Using TPA to count linear extensions

Jacqueline Banks
University of California, Riverside
jbank003@student.ucr.edu

Scott Garrabrant
Pitzer College
scott@garrabrant.com

Mark L. Huber
Claremont McKenna College
mhuber@cmc.edu

Anne Perizzolo
Columbia University
aperizzolo11@students.claremont.edu

Abstract

A linear extension of a poset P is a permutation of the elements of the set that respects the partial order. Let $L(P)$ denote the number of linear extensions. It is a #P complete problem to determine $L(P)$ exactly for an arbitrary poset, and so randomized approximation algorithms that draw randomly from the set of linear extensions are used. In this work, the set of linear extensions is embedded in a larger state space with a continuous parameter β. The introduction of a continuous parameter allows for the use of a more efficient method for approximating $L(P)$ called TPA. Our primary result is that it is possible to sample from this continuous embedding in time that as fast or faster than the best known methods for sampling uniformly from linear extensions. For a poset containing n elements, this means we can approximate $L(P)$ to within a factor of $1 + \epsilon$ with probability at least $1 - \delta$ using an expected number of random bits and comparisons in the poset which is at most $O(n^3(\ln n)(\ln L(P))^2\epsilon^{-2}\ln\delta^{-1})$.

Keywords: perfect simulation, posets, counting, #P complete

MSC Classification: Primary: 65C05; 06A07

1 Introduction

Consider the set $[n] = \{1, \ldots, n\}$ and a partial order $P = ([n], \preceq)$ on this set. Then a *linear extension* of the poset P is a permutation σ of $[n]$ such that for all $i < j$, $\sigma(i) \preceq \sigma(j)$. Say that such a permutation respects the partial order.

Our goal here is to efficiently count the number of linear extensions, which we denote $L(P)$. In general, finding $L(P)$ is a #P complete problem [2], and so instead of an exact deterministic method, we develop a randomized approximation method.

There are many applications of this problem. Morton et al. [9] have shown that a particular type of convex rank test for nonparametric models can be reduced to counting linear extensions. Many data sets such as athletic competitions or product comparisons do not have results for every possible pairing, but instead have an incomplete set of comparisons. Counting linear extensions can be used to develop estimates of the actual rank of the items involved (see [3].)

Previous results Previous methods for this problem ([1, 6]) concentrated on sampling from the set of linear extensions where some of the permutation values are fixed ahead of time. Generating a single uniform sample from the set of linear extensions takes $O(n^3 \ln n)$ expected number of random bits, using a number of comparisons that is at most the number of random bits [6]. Using the self-reducibility method of Jerrum et al. [8], this can be used to estimate $L(P)$ to within a factor of $1+\epsilon$ with probability at least $1-\delta$ in time $O(n^5(\ln n)^3\epsilon^{-2}\ln(1/\delta))$.

Here we take a different approach. Instead of sampling uniformly from the set of permutations, a weighted distribution is used that has a parameter β. The weight assigned to an element varies continuously with β, and this allows us to use a new method for turning samples from our weighted distribution into an approximation for $L(P)$ called the Tootsie Pop Algorithm (TPA). The use of TPA gives us an algorithm that is $O((\ln L(P))^2 n^3 (\ln n)\epsilon^{-2}\ln(1/\delta))$. In the worse case, $\ln L(P)$ is $O(n \ln n)$ and the complexity is the same as the older algorithm, however, if $L(P)$ is small compared to $n!$, this algorithm can be much faster. Even in the worst case, the constant hidden by the big-O notation is much smaller for the new algorithm (see Theorem 4 of Section 6.)

Organization In the next section, we describe the self-reducibility method and TPA in detail. Section 3 illustrates the use of TPA on a simple example, and then Section 4 shows how it can be used on the linear extensions problem by adding the appropriate weighting. Section 5 then shows how the non-Markovian coupling from the past method introduced in [6] can also be used for this new embedding, and Section 6 collects results concerning the running time of the procedure, including an explicit bound on the expected number of random bits and comparisons used by the algorithm.

2 The Tootsie Pop Algorithm

In [8], Jerrum et al. noted that for self-reducible problems, an algorithm for generating from a set could be used to build an approximation algorithm for finding the size of the set. Informally, a problem is self-reducible if the set of solutions can be partitioned into the solutions of smaller instances of the problem (for precise details, see [8].)

For example, in linear extensions once $\sigma(1)$ is determined, the problem of drawing $\sigma(2), \ldots, \sigma(n)$ is just a smaller linear extension generation problem.

While a theoretical tour de force, as a practical matter using self-reducibility to build algorithms is difficult. The output of a self-reducibility algorithm is a scaled product of binomials, not the easiest distribution to work with or analyze precisely.

The Tootsie Pop Algorithm (TPA) [7] is one way to solve this difficulty. Roughly speaking, TPA begins with a large set (the *shell*) containing a smaller set (the *center*). At each step, TPA draws a sample X randomly from the shell, and reduces the shell as much as possible while still containing X. The process then repeats, drawing samples and contracting the shell. This continues until the sample drawn lands in the center. The number of samples drawn before one falls in the center has a Poisson distribution, with parameter equal to the natural logarithm of the ratio of the size of the shell to the center.

To be precise, TPA requires the following ingredients

(a) A measure space $(\Omega, \mathcal{F}, \mu)$.

(b) Two finite measurable sets B and B' satisfying $B' \subset B$. The set B' is the *center* and B is the *shell*.

(c) A family of nested sets $\{A(\beta) : \beta \in \mathbb{R}\}$ such that $\beta < \beta'$ implies $A(\beta) \subseteq A(\beta')$. Also $\mu(A(\beta))$ must be a continuous function of β, and $\lim_{\beta \to -\infty} \mu(A(\beta)) = 0$.

(d) Special values β_B and $\beta_{B'}$ that satisfy $A(\beta_B) = B$ and $A(\beta_{B'}) = B'$.

With these ingredients, TPA can be run as follows.

Algorithm 2.1 TPA$(r, \beta_B, \beta_{B'})$

```
Input: Number of runs $r$, initial index $\beta_B$, final index $\beta_{B'}$
Output: $\hat{L}$ (estimate of $\mu(B)/\mu(B')$)
1: $k \leftarrow 0$
2: for $i$ from 1 to $r$ do
3:     $\beta \leftarrow \beta_B$, $k \leftarrow k - 1$
4:     while $\beta > \beta_{B'}$ do
5:         $k \leftarrow k + 1$, $X \leftarrow \mu(A(\beta))$, $\beta \leftarrow \inf\{\beta' \in [\beta_{B'}, \beta_B] : X \in A(\beta')\}$
6:     end while
7: end for
8: $\hat{L} \leftarrow \exp(k/r)$
```

Let $A = \ln(\mu(B)/\mu(B'))$, so that $\exp(A)$ is what we are trying to estimate. Then each run through the for loop in the algorithm requires on average $A + 1$ samples, making the total expected number of samples $r(A + 1)$. The value of k in line 7 of the algorithm is Poisson distributed with parameter rA. This means that r should be set to about A so that k/r is tightly concentrated around A.

But we do not know A ahead of time! This leads to the need for a two-phase algorithm. In the first phase r is set to be large enough to get a rough approximation of A, and then in the second phase r is set based on our estimate from the first run. That is:

1. Call TPA with $r_1 = 2\ln(2/\delta)$ to obtain $\hat{L}_1$, and set $\hat{A}_1 = \ln(\hat{L}_1)$.
2. Call TPA with $r_2 = 2(\hat{A}_1 + \sqrt{\hat{A}_1} + 2)[\ln(1+\epsilon)^2 - \ln(1+\epsilon)^3]^{-1}\ln(4/\delta)$ to obtain the final estimate.

The result is output $\hat{L}_2$ that is within a factor of $1+\epsilon$ of $L(P)$ with probability at least $1-\delta$. This is shown in Section 6.

3 Continuous embedding: simple example

To illustrate TPA versus the basic self-reducibility approach, consider a simple problem that will serve as a building block for our algorithm on linear extensions later. In this problem, we estimate the size of the set $\{1, 2, \ldots, n\}$ given the ability to draw samples uniformly from $\{1, 2, \ldots, b\}$ for any b.

In the self-reducibility approach, begin by setting $\beta_1 = \lceil n/2 \rceil$ and drawing samples from $\{1, \ldots, n\}$. Count how many fall into $\{1, \ldots, \beta_1\}$ and use this number $\hat{a}_1$ (divided by the number of samples) as an estimate of β_1/n. Now repeat, letting $\beta_2 = \lceil \beta_1/2 \rceil$ and estimating $\hat{a}_2 = \beta_2/\beta_1$ until $\beta_k = 1$. Note that

$$\mathbb{E}[\hat{a}_1 \hat{a}_2 \cdots \hat{a}_{k-1}] = \frac{\beta_1}{n}\frac{\beta_2}{\beta_1} \cdots \frac{\beta_k}{\beta_{k-1}} = \frac{\beta_k}{n}.$$

Since the final estimate $\hat{a}$ of $\frac{1}{n}$ is the product of $k-1$ estimates, Fishman called this algorithm the *product estimator* [4]. The problem with analyzing the output of the product estimator, is that it is the product of k scaled binomials.

To use TPA on this problem, it needs to be embedded in a continuous setting. Consider the state space $[0, n]$. The family of sets needed for TPA will be $[0, \beta]$, where $\beta_B = n$ and $\beta_{B'} = 1$. This makes the ratio of the measure of $[0, \beta_B]$ to $[0, \beta_{B'}]$ equal to n.

Note that you can draw uniformly from $[0, \beta]$ in the following two step fashion. First draw $X \in \{1, 2, \ldots, \lceil \beta \rceil\}$ so that $\mathbb{P}(X = i) = 1/\beta$ for $i < \beta$ and $\mathbb{P}(X = \beta) = (1 + \beta - \lceil \beta \rceil)/\beta$. If $X < \beta$, draw Y uniform on $[0, 1]$, otherwise draw Y uniform on $[0, 1 + \beta - \lceil \beta \rceil]$. The final draw is $W = X - 1 + Y$.

TPA starts with $\beta_0 = n$, then draws W as above. The infimum over all β such that $W \in [0, \beta]$ is just $\beta = W$. So β_1 just equals W. Next, redraw W from $[0, \beta_1]$. Again, the infimum of β satisfying $W \in [0, \beta]$ is just W, so β_2 equals this new value of W.

This process repeats until W falls into $[0, 1]$. The estimate k for $\ln n$ is just the number of steps needed before the final step into $[0, 1]$. Note that k can equal 0 if the very first step lands in $[0, 1]$. This random variable k will be Poisson distributed with parameter $\ln n$. Recall that the sum of Poisson random variables is also Poisson with parameter equal to the sum of the individual parameters, so repeating the process r times and summing the results yields a Poisson random variable with parameter $r \ln n$. Dividing by r and exponentiating then yields an estimate of n.

4 Continuous embedding: linear extensions

This approach can be extended to the problem of linear extensions as follows. A permutation can be viewed as a vector in $[n]^n$ space, where if $\sigma(i) = j$, then $\sigma(i') \neq j$ for all $i' \neq i$.

In the continuous embedding, our vectors x are in $(0, n]^n$, and if $\lceil x(i) \rceil = j$, then $\lceil x(i') \rceil \neq j$ for all $i' \neq i$. Such a vector x induces a unique permutation σ by taking the ceiling of all its elements, moreover, the Lebesgue measure of the set of x that map to a particular σ is always 1.

Therefore the measure of the set of x that map to linear extensions of the poset P is just the number of linear extensions. The state space $(0, n]^n$ can be restricted in a continuous fashion, leading to the use of TPA.

Let σ_{home} be any valid linear extension. For a vector x, define the distance from x to the home position to be:

$$d(x, \sigma_{\text{home}}) = \max_{i \in [n]} \lceil x(i) - \sigma_{\text{home}}(i) \rceil.$$

If the distance is 0, then no element i is to the right of the home position. The only way that can happen is if $\lceil x(i) \rceil = \sigma_{\text{home}}(i)$ for all i. Let $A(\beta)$ be vectors x in $(0, n]^n$ that map to linear extensions of P and $d(x, x_{\text{home}}) \leq \beta$. Then $\mu(A(n)) = L(P)$ and $\mu(A(0)) = 1$.

In order to use this family for TPA, it is necessary to draw samples from $A(\beta)$ for any $\beta \in [0, n]$. As in the previous section, a two stage process will be used. In the first stage, a linear extension is drawn nonuniformly from the set of linear extensions. Any linear extension where $\sigma(i) - \sigma_{\text{home}} > \beta$ has zero chance of being drawn. Any i where $\sigma(i) - \sigma_{\text{home}} = \lceil \beta \rceil$ contributes a penalty factor of $1 + \beta - \lceil \beta \rceil$ to the probability of $S(i)$. To be precise, let $\mathbb{P}(S = \sigma) = w(\sigma)/Z(\beta)$, where

$$w(\sigma) = \prod_{i \in [n]} ((1 + \beta - \lceil \beta \rceil)\mathbf{1}(\sigma(i) - \sigma_{\text{home}}(i) = \lceil \beta \rceil) + \mathbf{1}(\sigma(i) - \sigma_{\text{home}} < \lceil \beta \rceil)), \qquad (1)$$

and $Z(\beta) = \sum_\sigma w(\sigma)$.

For instance, suppose $\sigma_{\text{home}} = (1, 2, 3, 4)$, and $\beta = 1.3$. The permutation $\sigma_1 = (3, 2, 4, 1)$ (if this was a valid linear extension) would have probability $.3/Z(1.3)$ of occurring. Note the first component is $2 = \lceil 1.3 \rceil$ to the right of its home position, the second component is only 1 of the right of its home position, and the last two components are to the left of their home position. The permutation $\sigma_2 = (3, 4, 1, 2)$ would have probability $.3^2/Z(1.3)$ of occurring, and so on.

Once the linear extension S has been drawn, draw X as follows. For each i, let $X(i) \sim \text{Unif}((S(i)-1, S(i)]$ if $S(i) - \sigma_{\text{home}} < \lceil \beta \rceil$, otherwise let $X(i) \sim \text{Unif}((S(i)-1, S(i)+\beta-\lceil \beta \rceil])$. This is set up so that the normalizing constants in the uniform for $X(i)$ cancels the $1+\beta-\lceil \beta \rceil$ factor in the weight of σ if $\sigma(i) - \sigma_{\text{home}} = \lceil \beta \rceil$. Therefore X is a uniform choice over $A(\beta)$.

5 Sampling from the continuous embedding

For the continuous embedding to be useful for TPA, it must be possible to sample from the set of linear extensions with weights in (1). Once the linear extension has been created, sampling the continuous version is easy.

To sample from the set of weighted linear extensions, first build a Markov chain whose stationary distribution matches the target distribution. This is done by using the Metropolis-Hastings approach with a proposal chain that chooses two adjacent elements uniformly at random and transposes them (if such a transposition obeys the partial order) with probability 1/2. This is encoded in algorithm 5.1. In line 1, Unif denotes the uniform distribution, and Bern the Bernoulli distribution.

From this chain, it is possible to build a method for obtaining samples exactly from the target distribution. The method of coupling from the past (CFTP) was developed by Propp and Wilson [10] to draw samples exactly from the stationary distribution of Markov chains. For this problem, an extension called non-Markovian CFTP [6] is needed.

Algorithm 5.1 ChainStep(σ)

Input: current linear extension Markov chain state σ

Output: next linear extension Markov chain state σ

```
1: draw $i \leftarrow \text{Unif}(\{1,\ldots,n-1\})$, $C_1 \leftarrow \text{Bern}(1/2)$, $C_2 \leftarrow \text{Bern}(1+\beta-\lceil\beta\rceil)$
2: if $C_1 = 1$ and not $\sigma(i) \preceq \sigma(i+1)$ then
3:     if $\sigma(i) - \sigma_{\text{home}}(i) \neq \lceil\beta\rceil - 1$ or $C_2 = 1$ then
4:         $a \leftarrow \sigma(i+1)$, $\sigma(i+1) \leftarrow \sigma(i)$, $\sigma(i) \leftarrow a$
5:     end if
6: end if
```

The method works as follows. First, a bounding chain [5] is constructed for the chain in question. A bounding chain is an auxiliary chain on the set of subsets of the original state space. That is, $\Omega_{\text{bound}} = 2^{\Omega}$, where Ω is the state space of the original chain. Moreover, there is a coupling between the original chain $\{\sigma_t\}$ and the bounding chain $\{S_t\}$ such that σ_t evolves according to the kernel of the original bounding chain, and $\sigma_t \in S_t \to \sigma_{t+1} \in S_{t+1}$.

For us, the state of the bounding chain is indexed by a vector $B \in \{1,\ldots,n,\theta\}^n$. Let

$$S(B) = \{\sigma : (\forall i)((B(j) = i) \wedge (\sigma(j') = i) \Rightarrow j' \leq j)\}$$

For instance, if $B(3) = 4$, then $\sigma \in S(B)$ requires that $\sigma(1) = 4$ or $\sigma(2) = 4$ or $\sigma(3) = 4$. In this setup θ is a special symbol: if $B(i) = \theta$, then there is no restriction on σ whatsoever. To visualize what is happening with the state and bounding state, it will be useful to have a pictorial representation. For instance, if $\sigma = (4,2,3,1)$ and $B = (\theta,4,3,\theta)$ this can be represented by:

$$\underline{\ 4\ }|_{\theta}\underline{\ 2\ }|_{4}\underline{\ 3\ }|_{3}\underline{\ 1\ }|_{\theta}.$$

Note that the vertical line with subscript i ($|_i$) always appears to the right of the underlined i. This is how the bounding state works, it keeps track of the right most position of the item in the underlying state.

This bounding state has two other properties needed to use non-Markovian CFTP. First, there are several states that contains everything, for example $B = (\theta,\ldots,\theta,\sigma_{\text{home}}(1))$. Second, when all the θ values are gone, there is exactly one underlying state. That state must be equal to B. For instance, if $B = (4,3,1,2)$, then from the definition of S, the only possible underlying state is $\sigma = (4,3,1,2)$.

We are now ready to state the procedure for updating the current state and the bounding state simultaneously. This operates as in Algorithm 5.2. Note that if the inputs to the Algorithm have $i \sim \text{Unif}(\{1,2,\ldots,n\})$ and $C_1 \sim \text{Bern}(1/2)$, then the state σ is updated using the same probabilities as the previous chain step. The key difference between how σ and B are updated is that if $\sigma(i) = B(i+1)$, then B is updated using $C_3 = 1 - C_1$, otherwise $C_3 = C_1$. In any case, since $C_1 \sim \text{Bern}(1/2)$, $C_3 \sim \text{Bern}(1/2)$ as well.

Note that σ is being updated as in Algorithm 5.1. The only different is the bounding state update. Our main result is:

Theorem 1. *If $\sigma \in S(B)$, then running one step of Algorithm 5.2 leaves $\sigma \in S(B)$.*

Proof. Since only $\sigma(i), \sigma(i+1), B(i)$ and $B(i+1)$ are changing, they are the only states that need be considered. There are several cases. To describe the cases more succinctly, use $*$ to

Algorithm 5.2 BoundingChainStep(σ, B, i, C_1, C_2)

```
Input: current state and bounding state $(\sigma, B)$
Output: next state and bounding state $(\sigma, B)$
 1: $C_3 \leftarrow (1 - C_1)\mathbf{1}(\sigma(i) = B(i+1)) + C_1\mathbf{1}(\sigma(i) \neq B(i+1))$
 2: if $C_1 = 1$ and not $\sigma(i) \preceq \sigma(j)$ then
 3:     if $\sigma(i) - \sigma_{\text{home}}(i) \neq \lceil\beta\rceil - 1$ or $C_2 = 1$ then
 4:         $a \leftarrow \sigma(i+1)$, $\sigma(i+1) \leftarrow \sigma(i)$, $\sigma(i) \leftarrow a$
 5:     end if
 6: end if
 7: if $C_3 = 1$ and not $B(i) \preceq B(j)$ then
 8:     if $B(i) - \sigma_{\text{home}}(i) \neq \lceil\beta\rceil - 1$ or $C_2 = 1$ then
 9:         $a \leftarrow B(i+1)$, $B(i+1) \leftarrow B(i)$, $B(i) \leftarrow a$
10:     end if
11: end if
12: if $B(n) = \theta$ then
13:     $p \leftarrow$ the number of $i$ such that $B(i) \neq \theta$
14:     $B(n) \leftarrow \sigma_{\text{home}}(p+1)$
15: end if
```

denote a value of $\sigma(i)$ or $\sigma(i+1)$ that is neither $B(i)$ nor $B(i+1)$. For instance, the case $\underline{\ *\ }|_a\underline{\ \text{b}\ }|_b$ indicates $B(i) = a$, $B(i+1) = b$, $\sigma(i+1) = b$, and $\sigma(i)$ is neither a nor b. These cases are given in the table below.

First note if $\{\sigma(i), \sigma(i+1)\} \cap \{B(i), B(i+1)\} = \emptyset$, then if $\sigma(i') = B(i)$ then $i' < i$ and switching $B(i)$ and $B(i+1)$ leaves $B(i)$ bounded. Similarly, if $\sigma(i') = B(i+1)$ then $i' < i$ and switching $B(i)$ and $B(i+1)$ leaves $B(i+1)$ bounded. Hence the only interesting cases are when the sets $\{\sigma(i), \sigma(i+1)\}$ and $\{B(i), B(i+1)\}$ overlap. These cases are given below.

Case 1: $\underline{\ \text{a}\ }|_a\underline{\ *\ }|_\theta$. For a to move right, it must be true that $C_1 = 1$ and it must be false that $a \preceq \sigma(i+1)$. If $a - \sigma_{\text{home}}(i) = \lceil\beta\rceil$ then it is also necessary that $C_2 = 1$. Under these conditions, the $|_a$ also always moves right, so continues to bound a.

Case 2: $\underline{\ \text{a}\ }|_a\underline{\ *\ }|_b$. That b is bounded just follows from the fact that if $\sigma(i') = b$, then $i' < i$. This further implies that it is not true that $a \preceq b$. Hence (just as in Case 1) if a swaps, then $|_a$ must swap as well.

Case 3: $\underline{\ \text{a}\ }|_a\underline{\ \text{b}\ }|_b$. Since $C_1 = C_3$ and the swap for a and b and $|_a$ and $|_b$ are determined in the same fashion, either they both swap or neither pair swaps.

Case 4: $\underline{\ \text{a}\ }|_\theta\underline{\ *\ }|_a$. For a to move right, it must be true that $C_1 = 1$, which makes $C_3 = 0$. Hence the $|_a$ continues to bound a.

Case 5: $\underline{\ \text{a}\ }|_b\underline{\ *\ }|_a$. Same as Case 4.

Case 6: $\underline{\ *\ }|_\theta\underline{\ \text{b}\ }|_b$. There are three conditions under which b stays in location $i+1$: 1) $C_1 = 0$. 2) $\sigma(i) \preceq b$. 3) $\sigma(i) - \sigma_{\text{home}}(i) = \lceil\beta\rceil$ and $C_2 = 0$. If 1) occurs, then $|_b$ does not move either. It turns out that 2) cannot occur. To see this, let c be any element that precedes b in the partial order. Then it also precedes b in the home permutation σ_{home}. That means that in lines 12 through 15, $|_c$ was introduced before the $|_b$ bound was introduced. Since the $|_c$ bound is always to the left of the $|_b$ bound, and is not in position i, it must be somewhere to the left of i. Hence c is in a location to the left of i, that is, $\sigma(i') = c \rightarrow i' < i$. Since c

was an arbitrary element that precedes b, situation 2) never occurs.

Finally, if 3) occurs, then $|_b$ also does not move. Hence the only times b stays put, $|_b$ will also stay, and lines 2 through 11 maintain the property that σ is bounded by B.

The last lines of the algorithm (12 through 15) deal with the situation where $B(n) = \theta$. In this case switching $B(n)$ to any nonzero number will not hurt the bounding property. Therefore we switch to the "next" number in line, which is found by using the values in order σ_{home}. □

Hence if σ is bounded by B, it will still be bounded after taking one step in the bounding chain step. With this established, samples from the target distribution can be generated as in Algorithm 5.3 [6, 10] using non-Markovian CFTP.

Algorithm 5.3 Generate(t)

Input: t number of steps to use to generate a sample
Output: σ drawn from the weighted distribution

```
1: $\sigma \leftarrow \sigma_{\text{home}}$, $B \leftarrow (\theta, \ldots, \theta, \sigma_{\text{home}}(1))$
2: for $j$ from 1 to $t$ do
3:     draw $i(j) \leftarrow \text{Unif}(\{1, \ldots, n-1\})$, $C_1(j) \leftarrow \text{Bern}(1/2)$,
       $C_2(j) \leftarrow \text{Bern}(1 + \beta - \lceil \beta \rceil)$
4:     $(\sigma, B) \leftarrow \text{BoundingChainStep}(\sigma, B, i(j), C_1(j), C_2(j))$
5: end for
6: if for all $i$, $B(i) \neq \theta$ then
7:     $\sigma \leftarrow B$
8: else
9:     $\sigma \leftarrow \text{Generate}(2t)$
10:    for $j$ from 1 to $t$ do
11:        $(\sigma, B) \leftarrow \text{BoundingChainStep}(\sigma, B, i(j), C_1(j), C_2(j))$
12:    end for
13: end if
```

6 Analysis

In this section we prove several results concerning the running time of the procedure outlined in the previous section.

Theorem 2. *The non-Markovian coupling from the past in Algorithm 5.3 requires an expected number of random bits bounded by* $4.3n^3(\ln n)(\lceil \log_2 n \rceil + 3)$ *and a number of comparisons bounded by* $8.6n^3 \ln n$.

Theorem 3. *For* $\epsilon \leq 1$, *the two-phase* TPA *approach outlined at the end of Section 2 generates output* $\hat{L}_2$ *such that*

$$\mathbb{P}((1+\epsilon)^{-1} \leq \hat{L}_2 / L(P) \leq 1 + \epsilon) \geq 1 - \delta.$$

Theorem 4. *The expected number of random bits needed to approximate $L(P)$ to within a factor of $1 + \epsilon$ with probability at least $1 - \delta$ is bounded above by*

$$4.3n^3(\ln n)(\lceil \log_2 n \rceil + 3)[2(A+1)\ln(2/\delta) + (A+1)(A+3\sqrt{2A}+2)(\ln(1+\epsilon)^2 - \ln(1+\epsilon)^3)\ln(4/\delta)].$$

Proof of Theorem 2. Lemma 10 of [6] showed that when there is no β parameter, the expected number of steps taken by non-Markovian CFTP was bounded above by $4.3n^3 \ln n$.

So the question is: once the β parameter falls below n, does the bound still hold? The bound was derived by considering how long it takes for the $|_\theta$ values in the bounding state to disappear. Each time a $|_\theta$ reaches position n, it is removed and replaced by something of the form $|_a$. When all the $|_\theta$ disappear, the process in Algorithm 5.3 terminates.

When there is no β, the probabilities that a particular $|_\theta$ bound moves to the left or the right are equal: both $1/(2n)$. (This does not apply when the bound is at position 1, in which case the bound cannot move to the left.) The result in [6] is really a bound on the number of steps in a simple random walk necessary for the $|_\theta$ bounds to all reach state n.

Now suppose that $\beta \in (0, n)$. The probability that a $|_\theta$ bound moves to the right is still $1/(2n)$, but now consider when the state is of the form $\ldots \underline{\quad}|_a \underline{\quad}|_\theta \ldots$. For $|_\theta$ to move left the $|_a$ has to move right, and this could occur with probability $(1 + \beta - \lceil \beta \rceil)/(2n)$. That is, with $\beta \in (0, n)$, the chance that the $|_\theta$ moves left can be below $1/(2n)$.

This can only reduce the number of moves necessary for the $|_\theta$ bounds to reach the right hand side! That is, the random variable that is the number of steps needed for all the $|_\theta$ bounds to reach position n and disappear is dominated by the same random variable for $\beta = n$. Hence the bound obtained by Lemma 10 of [6] still holds.

Now to the random bits. Drawing a uniform number from $\{1, \ldots, n\}$ takes $\lceil \log_2 n \rceil$ bits, while drawing from $\{0, 1\}$ for coin C_1 takes one bit. The expected number of bits needed to draw a Bernoulli random variable with parameter not equal to $1/2$ is two, and so the total bits needed for one step of the process (in expectation) is $\lceil \log_2 n \rceil + 3$. Each step in the bounding chain and state uses at most two comparisons. □

It will be helpful in proving Theorem 3 to have the following bound on the tail of the Poisson distribution.

Lemma 1. *For $X \sim \text{Pois}(\mu)$ and $a \leq \mu$, $\mathbb{P}(X \geq \mu + a) \leq \exp(-(1/2)a^2/\mu + (1/2)a^3/\mu^2)$ and for $a \leq \mu$, $\mathbb{P}(X \leq \mu - a) \leq \exp(-a^2/(2\mu))$.*

Proof. These follow from Chernoff Bounds which are essentially Markov's inequality applied to the moment generating function of the random variable. The moment generating function of X is $\mathbb{E}[\exp(tX)] = \exp(\mu(e^t - 1))$. So for $a > 0$

$$\mathbb{P}(X \geq \mu + a) = \mathbb{P}(\exp(tX) \geq \exp(t(\mu + a)) \leq \frac{\exp(\mu(e^t - 1))}{\exp(t(\mu + a))}.$$

Setting $t = \ln(1 + a/\mu)$ minimizes the right hand side, and yields:

$$\mathbb{P}(X \geq \mu + a) \leq \exp(a - (\mu + a)\ln(1 + a/\mu)).$$

For $a \leq \mu$, $-\ln(1 + a/\mu) \leq -a/\mu + (1/2)a^2/\mu^2$, so $\mathbb{P}(X \geq \mu + a) \leq \exp(-(1/2)a^2/\mu + (1/2)a^3/\mu^2)$ as desired. For the second result:

$$\mathbb{P}(X \leq \mu - a) = \mathbb{P}(\exp(-tX) \geq \exp(-t(\mu - a)) \leq \frac{\exp(\mu(e^{-t} - 1))}{\exp(-t(\mu - a))}.$$

Setting $t = -\ln(1 - a/\mu)$ then yields the next result.

For $a \leq \mu$, $-\ln(1 - a/\mu) \leq a/\mu + (1/2)(a/\mu)^2$. So

$$-a - (\mu - a)\ln(1 - a/\mu) \leq -a + (\mu - a)((a/\mu) + (1/2)(a/\mu)) = -(1/2)(a^2/\mu) - (1/2)(a^3/\mu^2).$$

The right hand side is at most $-(1/2)a^2/\mu$, which completes the proof. □

Proof of Theorem 3. Consider the first phase of the algorithm, where TPA is run with $r_1 = 2\ln(2/\delta)$. Consider the probability of the event $\{\hat{A}_1 + \sqrt{\hat{A}_1} + 2 < A\}$. This event cannot happen if $A \leq 2$. If $A > 2$, then this event occurs when $\hat{A}_1 < A - (3/2) - \sqrt{A - 7/4}$. Since $r_1\hat{A}_1 \sim \text{Pois}(rA)$, Lemma 1 can be used to say that

$$\begin{aligned}\mathbb{P}(r_1\hat{A}_1 < r_1A - r_1(3/2 + \sqrt{A - 7/4})) &\leq \exp(-(1/2)(r_1(3/2 + \sqrt{A - 7/4}))^2/(r_1A)^2 \\ &\leq \exp(-(1/2)r_1(9/4 + A - 7/4)/A^2) \\ &\leq \exp(-(1/2)r_1/A) \\ &\leq 2/\delta.\end{aligned}$$

In other words, with probability at least $1 - \delta/2$, $\hat{A}_1 + \sqrt{\hat{A}_1} + 2 \geq A$.

Now consider the second phase. To simplify the notation, let $\epsilon' = \ln(1 + \epsilon)$, and $\hat{A}_2 = \exp(\hat{L}_2)$ where $\hat{L}_2$ is the output from the second phase. Then from the first phase $r_2 \geq A(\epsilon'^2 - \epsilon'^3)^{-1}\ln(4/\delta)$ with probability at least $1 - \delta/2$.

So from Lemma 1,

$$\begin{aligned}\mathbb{P}(r_2\hat{A}_2 \geq r_2A + r_2\epsilon') &\leq \exp(-(1/2)(r_2\epsilon')^2/(r_2A) + (1/2)(r_2\epsilon')^3/(r_2A)^2) \\ &= \exp(-(1/2)r_2\epsilon'^2/A + (1/2)r_2\epsilon'^3/A^2) \\ &\leq \exp(-\ln(4/\delta)).\end{aligned}$$

A similar bound holds for the left tail:

$$\mathbb{P}(r_2\hat{A}_2 \leq r_2A - r_2\epsilon') \leq \exp(-(1/2)r_2^2\epsilon'^2/(r_2A)) \leq \delta/4.$$

Therefore, the total probability that failure occurs in either the first phase or the second is at most $\delta/2 + \delta/4 + \delta/4 = \delta$. If $r_2\hat{A}_2$ is within additive error $r_2\epsilon' = r_2\ln(1 + \epsilon)$ of r_2A, then $\hat{L}_2 = \exp(\hat{A}_2/r)$ is within a factor of $1 + \epsilon$ of $\exp(A)$, showing the result. □

To bound the expected running time, the following loose bound on the expected value of the square root of a Poisson random variable is useful.

Lemma 2. *For $X \sim \text{Pois}(\mu)$, $\mathbb{E}[\sqrt{X}] \leq 3\sqrt{2\mu}$.*

Proof. First note $\mathbb{E}[\sqrt{X}] = \mathbb{E}[\sqrt{X}\mathbf{1}(X \leq 2\mu)] + \mathbb{E}[\sqrt{X}\mathbf{1}(X > 2\mu)]$. Then $\mathbb{E}[\sqrt{X}\mathbf{1}(X \leq 2\mu)] \leq \sqrt{2\mu}$, while the second term is:

$$\mathbb{E}[\sqrt{X}\mathbf{1}(X > 2\mu)] = \sum_{i=\lfloor 2\sqrt{\mu}\rfloor+1}^{\infty} \sqrt{i}\mathbb{P}(X = i).$$

The ratio between successive terms in this series is:

$$\frac{\sqrt{i+1}\exp(-\mu)(\mu)^{i+1}/(i+1)!}{\sqrt{i}\exp(-\mu)(\mu)^{i}/(i)!} = \frac{\mu}{\sqrt{i(i+1)}} \leq 1/2.$$

The first term in the series is at most $\sqrt{2\mu}$, so the series sums to at most $2\sqrt{2\mu}$. □

Proof of Theorem 4. From Theorem 2, the expected number of bits per sample is bounded by $4.3n^3(\ln n)(\lceil \log_2 n\rceil + 3)$ and does not depend on the sample. Hence the total number of expected bits can be bounded by the expected number of bits per samples times the expected number of samples. The first phase of TPA uses $r_1 = 2\ln(2/\delta)$ runs, each with an expectation of $A+1$ samples per run to make $r_1(A+1)$ expected samples. The second phase uses $r_2 = (\hat{A}_1 + \sqrt{\hat{A}_1} + 2)[\ln(1+\epsilon)^2 - \ln(1+\epsilon)^3]\ln(4/\delta)$ runs, where $r_1\hat{A}_1 \sim \text{Pois}(r_1 A)$. So from Lemma 2,

$$\mathbb{E}[\sqrt{\hat{A}_1}] = r_1^{-1/2}\mathbb{E}[\sqrt{r_1 A}] \leq r_1^{-1/2}3\sqrt{2r_1 A} = 3\sqrt{2A}.$$

Using $A = \ln L(P)$ and then combining these factors yields the result. □

7 Conclusion

TPA is a sharp improvement on the self-reducibility method of Jerrum et al. for estimating the size of a set. At first glance, the continuity requirement of TPA precludes its use for discrete problems such as linear extensions. Fortunately, discrete problems can usually be embedded in a continuous space to make the use of TPA possible. Here we have shown how to accomplish this task in such a way that the time needed to take samples is the same as for uniform generation. The result is an algorithm that is much faster at estimating the number of linear extensions than previously known algorithms.

References

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

[2] G. Brightwell and P. Winkler. Counting linear extensions. *Order*, 8(3):225–242, 1991.

[3] P.C. Fishburn and W.V. Gehrlein. A comparative analysis of methods for constructing weak orders from partial orders. *J. Math. Sociology*, 4:93–102, 1975.

[4] G. S. Fishman. Choosing sample path length and number of sample paths when starting in the steady state. *Oper. Res. Letters*, 16:209–219, 1994.

[5] M. Huber. Perfect sampling using bounding chains. *Annals of Applied Probability*, 14(2):734–753, 2004.

[6] M. Huber. Fast perfect sampling from linear extensions. *Discrete Mathematics*, 306:420–428, 2006.

[7] M. L. Huber and S. Schott. Using TPA for Bayesian inference. *Bayesian Statistics 9*, 2010.

[8] M. Jerrum, L. Valiant, and V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoret. Comput. Sci.*, 43:169–188, 1986.

[9] J. Morton, L. Pachter, A. Shiu, B. Sturmfels, and O. Wienand. Convex rank tests and semigraphoids. *SIAM J. Discrete Math.*, 23(2):1117–1134, 2009.

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