A new algorithm for approximation of the number of linear extensions of a poset

Mark Huber

Departments of Mathematics and Statistical Science
Duke University

10 Dec, 2008

Example

Example (Who is the best tennis player?)

Jelena, Serena, Dinara, or Elena?

Unfortunately, they only have time to play 3 games

- Jelena beats Dinara
- Serena beats Dinara
- Jelena beats Elena

Who is most likely to be number 1?

A simple model

Prior distribution: All rankings equally likely
Condition on $J \preceq D$, $S \preceq D$, $J \preceq E$
Posterior distribution uniform on

- JSDE
- JSED
- JESD
- SJDE
- SJED

Posterior probability of being in first place:

- $\mathbb{P}(J = \text{first}) = 3/5, \mathbb{P}(S = \text{first}) = 2/5$
- $\mathbb{P}(D = \text{first}) = \mathbb{P}(E = \text{first}) = 0$

The problem

Posterior uniform over *linear extensions* of partial order

$$\mathbb{P}(A \text{ first}) = \frac{\# \text{ of lin. ext. } \sigma \text{ with } \sigma(1) = A}{\text{Total } \#\text{of lin. ext. } x}$$

The bad news: counting linear extensions is # P-complete[1]

Solution: develop approximation algorithms

[1] Brightwell & Winkler 1991

The results

Two main results

- A new method for random generation of linear extensions
- A new method for turning samples into counting algorithm

Improvement

- Sampler: Can be much faster on sparse partial orders
- Counting: $\Theta(n/(\ln n)^2)$ faster

Outline

1. Applications of linear extensions
   - Machine learning
   - Convex rank tests
2. Previous work
   - Approximating volume of convex bodies
   - Perfect simulation
   - The product estimator
3. The new algorithm
   - Reducing # of levels in product estimator
   - Retooling perfect sampler to handle new levels
   - Results

Partial orders

Notation:

$$\begin{aligned} [n] &= \{1, \ldots, n\} \\ P &= (\preceq, [n]) \text{ is a partial order on } [n] \end{aligned}$$

A partial order has three properties:

- Reflexive: $a \preceq a$
- Antisymmetric: $a \preceq b$ and $b \preceq a$ implies $a = b$
- Transitive: $a \preceq b$ and $b \preceq c$ implies $a \preceq c$

Linear extensions

Definition

A *linear extension* of a partial order is a permutation σ of $[n]$ that respects the partial order, so

$$i < j \to \sigma(i) \preceq \sigma(j) \text{ or } \sigma(i), \sigma(j) \text{ unrelated}$$

Example: Suppose $A \preceq B$ then permutation must have the form:

$$\star \cdots \star A \star \cdots \star B \star \cdots \star$$

Goal: count Ω, the set of linear extensions of P

Challenges

Problem:

- counting linear extensions is #P-complete
- #P complete problems are harder than NP-complete
- unlikely to find polynomial time algorithm

Solution:

- Develop Monte Carlo approximation algorithm
- Come within factor of $1 + \epsilon$ with probability at least $1 - \delta$
- Want run time $\text{poly}(n)[\ln(1/\delta)]/\epsilon^2$

Machine learning[2]

Learning a binary relation

- Example: Tennis example–ranked items with some order
- Example query: Does A outrank B?
- Goal is to predict answers from a few queries
- Step 1: Use queries to create partial order
- Step 2: Use random linear extension to make predictions

The problem here is to sample linear extensions

[2] Goldman, Rivest, & Shapire 1989

Convex rank tests[3]

Nonparametric statistical model: all permutations equally likely

- *Rank tests* used for ordered data
- Tests can be viewed as partition of symmetric group S_n
- Find p-values by counting size of equivalence classes
- Each equivalence class is set of linear extensions for a partial order

The problem here is to count linear extensions

[3] Morton, Pachet, Shiu, Sturmfels, & Wienand 2008

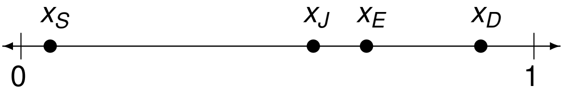
Outline

1. Applications of linear extensions
   - Machine learning
   - Convex rank tests
2. Previous work
   - Approximating volume of convex bodies
   - Perfect simulation
   - The product estimator
3. The new algorithm
   - Reducing # of levels in product estimator
   - Retooling perfect sampler to handle new levels
   - Results

Linear extensions as convex bodies

Example: embedding linear extensions in continuous space

$$(x_S, x_j, x_e, x_d) \in [0, 1]^4$$



SJED

Enforce $A \preceq B$ with $x_A \leq x_B$

Alternate encodings

Hit and run chain

- Choose random direction
- Choose random point inside convex body

For any convex body:[4]

- Steps per sample (amortized): $O(n^3(\ln n)^3)$
- Steps for approximately counting: $O(n^4(\ln n)^7)$
- Each step: $\Theta(\#\text{relations in partial order})$
- Total time: $\Theta(n^6(\ln n)^7)$
- Impractical: $O(\cdot)$ hides large constant (at least 1000)

[4] Lovász and Vempala 2006

Bounding chains and coupling from the past (CFTP)

CFTP[5] is a perfect simulation protocol

- Draws samples exactly from uniform distribution
- Running time is a random variable
- Requires extra construction such as: monotonicity, bounding chains[6]

Bounding chain for linear extensions[7]

- Uses adjacent transposition chain
- $O(n^3 \ln n)$ steps for sparse problems

[5] Propp & Wilson 1996
[6] H. 2004
[7] H. 2006

Adjacent transposition chain

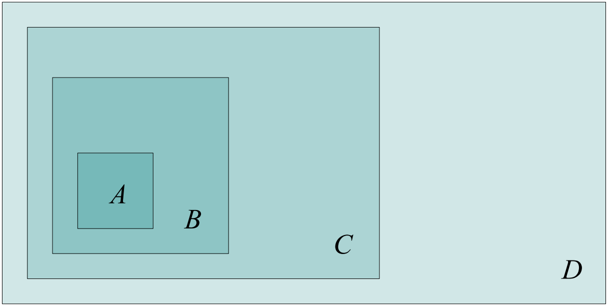
One step in chain

- Stay at same state with probability $1/2$
- Otherwise choose i uniformly from $\{1, \ldots, n\}$
  swap items at position i and $i + 1$

Examples

$$\begin{aligned} SJDE + \text{position } 1 + \text{move} &= JSDE \\ SJDE + \text{position } 3 + \text{stay} &= SJDE \\ SJDE + \text{position } 2 + \text{move} &= SJDE \text{ (since } J \preceq D) \end{aligned}$$

Product Estimator



Know: $\#A$

Want: $\#D$

$$\widehat{\#D} = \widehat{\left(\frac{\#D}{\#C}\right)} \widehat{\left(\frac{\#C}{\#B}\right)} \widehat{\left(\frac{\#B}{\#A}\right)} \#A$$

Total # of samples needed: Ck^2/ϵ^2

▸ General procedure

Naive Product Estimator

At each level *fix* one element of permutation

Example (* = wildcard)

- ****
- J***
- JS**
- JSD* = JSDE

Parameters:

- Number of levels: n
- $C = n$
- Total # of samples: $O(n^3)$

Previous work versus today

	Convex Bodies	Discrete	Today
Sampling	Hit and run $O(n^5(\ln n)^3)$	Adjacent Trans. $O(n^3 \ln n)$	Adj. Trans. Plus restrictions $O(n^3 \ln n)$
Counting	Advanced prod. est. $O(n^6(\ln n)^9)$	Naive prod. estimator $O(n^6 \ln n)$	Improved prod. est. $O(n^5(\ln n)^3)$

Outline

1. Applications of linear extensions
   - Machine learning
   - Convex rank tests
2. Previous work
   - Approximating volume of convex bodies
   - Perfect simulation
   - The product estimator
3. The new algorithm
   - Reducing # of levels in product estimator
   - Retooling perfect sampler to handle new levels
   - Results

Slicing state space more efficiently

Previously

- Ratio between levels: $C = n$
- Number of levels: $k = n$

Today

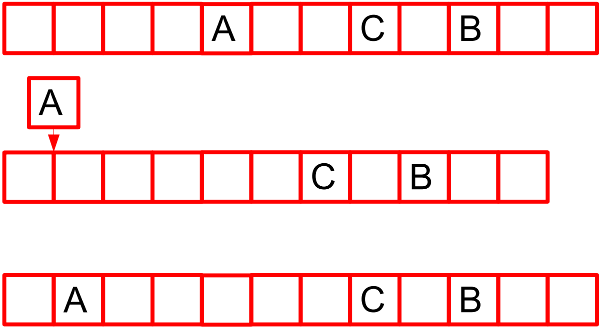
- Ratio between levels: $C = 2$
- Number of levels: $k = n \log_2 n$

Distribution of a maximal element

Stationary update function

- Remove item A from permutation
- Reinsert item A uniformly among available positions

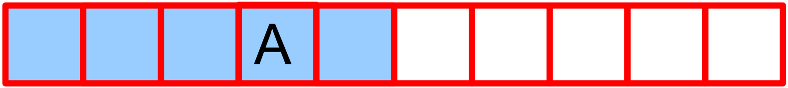
Example: $A \preceq B$, $A \preceq C$



Probability max element in left half at least 1/2

In general

- If no element precedes A
- $\mathbb{P}(\sigma^{-1}(A) \leq \lceil n/2 \rceil) \geq 1/2$



How to slice the state space in half

Restricted linear extensions

- Start with upper bound $\sigma^{-1}(A) = n_1$, $n_1 = n$
- Next $\sigma^{-1}(A) = n_2$, $n_2 = \lceil n_1/2 \rceil$
- Continues until $n_j = 1$

Example

- $A \preceq B$, $A \preceq C$
- Current state $* * * A * * B * * C$
- If A chosen, after one step A equally likely to be in first 6 positions

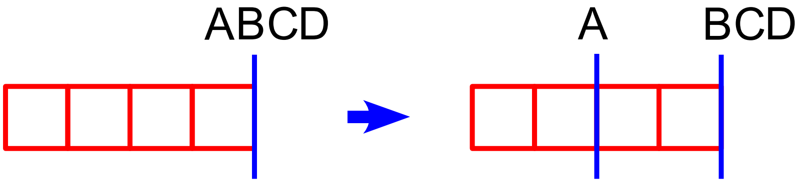
In general

- If no element precedes A
- $\mathbb{P}(\sigma^{-1}(A) \leq \lceil n/2 \rceil) \geq 1/2$

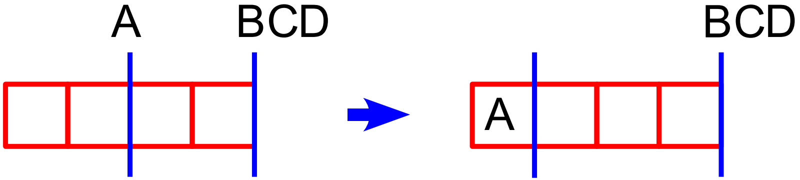
```
function n = product_estimator(A,epsilon,delta)

n = size(A); n = n(1);  % Find side length of matrix A;
k = ceil(log(n)/log(2));
est = zeros(n-1,k);
num_per_level = 4*(n - 1)*k / epsilon^2 * log(1/delta);
for move_item = 1:(n-1)
    move_from_loc = n;
    move_to = 0;
    while (move_from_loc > move_item)
        move_to = move_to + 1;
        move_to_loc = max(move_item,ceil(move_from_loc / 2));
        for samples = 1:num_per_level
           x = gler(A,move_item - 1,move_item,move_from_loc);
           est(move_item,move_to) = est(move_item,move_to) + ...
                (sum(x(1:move_to_loc) == move_item) > 0);
        end
        move_from_loc = move_to_loc;
        est(move_item,move_to)=log(est(move_item,move_to)/num_per_level);
    end
end
n = exp(-sum(sum(est)));
```

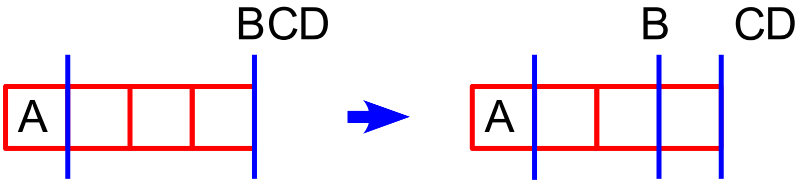
Levels with 4 states...



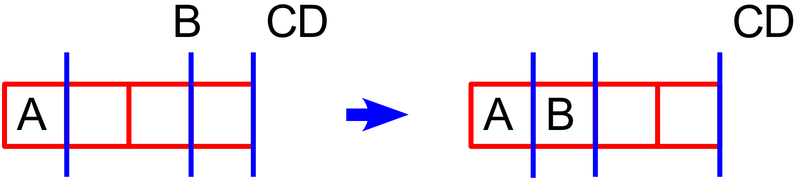
Levels with 4 states...



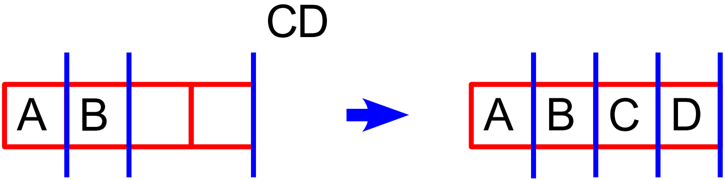
Levels with 4 states...



Levels with 4 states...



Levels with 4 states...



To generate samples

Key line in earlier code:

```
x = gler(A,move_item - 1,move_item,move_from_loc);
```

Need function `gler` to generate restricted linear extensions

- Need $A \preceq B \Rightarrow \sigma^{-1}(A) < \sigma^{-1}(B)$
- For all items α, $\sigma^{-1}(\alpha) \leq r(\alpha)$

How to get samples

Bounding chains	+	Couping From the Past	=	Perfect Simulation

Perfect simulation = draws exactly uniform over linear extensions

- Technique similar to [H. 2004]
- Bounding chain needs modification
- Running time same as for unrestricted
- Still $O(n^3 \ln n)$

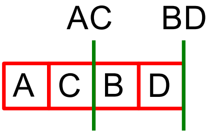
Bounding chains[8]

Bounding state tells what positions can be occupied

- Similar to levels in product estimator
- When upper bounds different, unique permutation bounded
- Modification to handle levels straightforward

Example of bounding state:

$$x = ACBD \qquad\qquad y = (2, 4, 2, 4)$$



▸ More details

[8] H. 2004, H. 2006

Further work

Other perfect simulation methods

- Monotonic Markov chain
- Embed permutations in $[0, 1]^n$
- Delete-reinsert Step 1: Choose an item
- Delete-reinsert Step 2: Remove item
- Delete-reinsert Step 3: Uniformly place item back in

Can be faster for sparse partial orders:

$$O(n \ln n) \text{ for empty } P$$

Advanced product estimator

Summary

Linear extensions arise in several statistical applications

- Counting exactly is difficult
- Fully polynomial randomized approximation scheme (fpras)

New product estimator, new perfect simulator

- Product estimator fewer levels
- Perfect simulator same speed as old
- Needs $3n^5(\ln n)^3\epsilon^{-2}\ln(1/\delta)$ expected number of uniforms
- Beats previous $1000n^6(\ln n/\epsilon^2)^9\epsilon^{-2}\ln(1/\delta)$

References

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

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

P. Mathews.
Generating a random linear extension of a partial order.
Ann. Probab., 19(3):1367–1392, 1991.

J. Morton, L. Pachter, A. Shiu, B. Sturmfels, and O. Wienand.
Convex rank tests and semigraphoids,
arXiv:math/0702564v2[math.CO], 2008.

Many ways to create convex body

Mathews[9] used

$$\Omega = \{x \in \mathbb{R}^d : x_1^2 + x_2^2 + \cdots x_d^2 \leq 1\}$$

Again, enforce partial order via half-space constraints:

$$A \preceq B \Rightarrow x_A \leq x_B$$

Mathews devised Markov chain step specialized to this space

Go back

[9] Mathews1991

General procedure

Given sets $A_1 \subset \cdots \subset A_k$

- k levels
- Create enough levels so $\#A_i/\#A_{i-1} \leq C$ for all i
- Estimate $\#A_{i-1}/\#A_i$ using N samples from A_i
- Estimate $\#A_k$ with telescoping product
- Call final estimate $\hat{p}$
- $\mathbb{E}[p] = \#A_k$
- For $\mathrm{SD}(p) = \epsilon\mathbb{E}[p]$, set $N = Ck/\epsilon^2$

Total # of samples needed: Ck^2/ϵ^2

Want to minimize C and minimize number of levels k

Go back

Bounding chains for adjacent transpositions[10]

Upper bounds positions

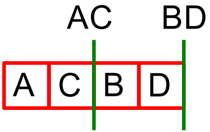
- Permuting items $A_1, \ldots, A_n$
- $x^{-1}(A_i) \leq y(A_i)$ for all items A_i
- Trick is to update (x, y) to next state (x', y') so that:

$$(\forall i)(x^{-1}(A_i) \leq y(A_i)) \to (\forall i)(x'^{-1}(A_i) \leq y'(A_i))$$

Example of bounding state:

$x = ACBD$, $x^{-1}(A) = 1, x^{-1}(B) = 3, x^{-1}(C) = 2, x^{-1}(D) = 4$

$y = (2, 4, 2, 4)$



[10] H. 2006

Bounding a single permutation

When only a single x for y:

- Say $y(i) \neq y(j)$ for all $i \neq j$
- Then $x^{-1} = y$ only state bounded by chain

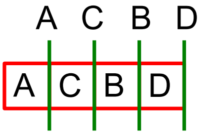
Example:

$$y = (2, 4, 2, 4)$$

$x^{-1} = (1, 3, 2, 4)$ or
$x^{-1} = (2, 3, 1, 4)$ or
$x^{-1} = (1, 4, 2, 3)$ or
$x^{-1} = (2, 4, 1, 3)$

$$y = (1, 3, 2, 4)$$

$$x^{-1} = (1, 3, 2, 4)$$



Typical run

Take a fixed number T of bounding chain steps:

$y = (4, 4, 4, 4)$
(bounds everything)

$y = (1, 3, 2, 4)$
(good outcome–one bounded state)

$y = (2, 4, 2, 4)$
(bad outcome–many bounded states)

What is it good for?

Two uses for bounding chains

- Expected time to bound single state gives an upper bound on mixing time of chain
- Together with coupling from the past, can generate samples exactly from uniform distribution

Go back