Multiparametric Boltzmann sampling and applications

Maciej Bendkowski [a] *Sergey Dovgal* [b] Olivier Bodini [c]

[a]Jagellonian University, Krakow

[b]LaBRI, Université de Bordeaux

[c]LIPN, Université Sorbonne Paris Nord

MOCQUA Seminar, 25/03/2021 (online)

Random sampling

Problem

Suppose that some implicit description of $\mathbb{P}$ is provided.

$$\text{Let } X \sim \mathbb{P}, \ \text{sample } X$$

Framework

- Which descriptions of $\mathbb{P}$ are admitted?
- If $\mathbb{P}$ is parametric, what is pre- and computation complexity?
- What error margin is allowed?

Why random sampling?

Motivations for random sampling

- Art and entertainment
  - T-shirt printing
  - Paintings, decorations, tilings
  - Music composition
  - Artificial intelligence artwork
- Monte-Carlo simulations
  - Property-based software testing (QuickCheck, lambda terms)
  - Biology (cell dynamics, RNA structures)
  - Statistical physics (random maps, Bose–Einstein condensate, Ising model, tilings, plane partitions)
- Theoretical computer science
  - Random permutations, sorting algorithms, cellular automata
  - Random graphs and community detection
  - Crypto primitives and low-level programming
  - Concurrent process analysis, queueing systems
  - Automata sampling

Outline of the current talk

$$\underbrace{\text{Multiparametric Boltzmann sampling}}_{\text{Part I}} \text{ and } \underbrace{\text{applications}}_{\text{Part II}}$$

Part I. Generating functions and Boltzmann samplers

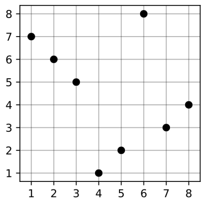
Generating functions and the symbolic method

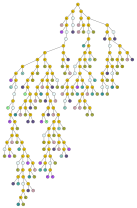
Framework

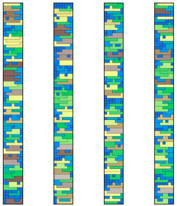
- Discrete objects are represented by *words* in a finite alphabet.
- The *size* of the object is the number of its letters.
- Let a_n be the number of words of length n

Generating function of the counting sequence:

$$A(z) = \sum_{n=0}^{\infty} a_n z^n$$

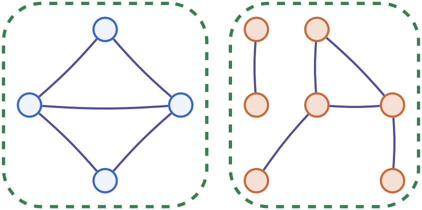






The cartesian product

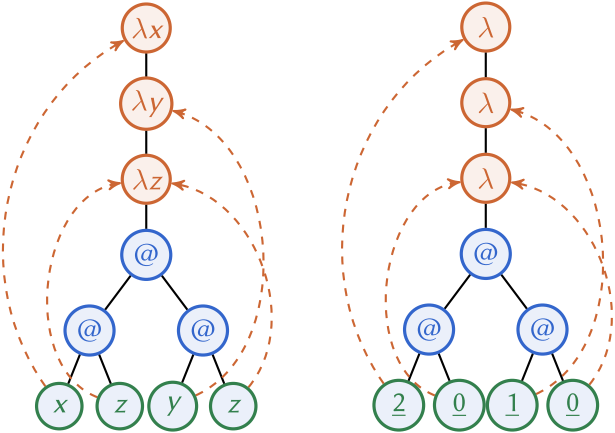
$$\left(a_0 + a_1 z + a_2 z^2 + \ldots\right)\left(b_0 + b_1 z + b_2 z^2 + \ldots\right) = c_0 + c_1 z + c_2 z^2 + \ldots$$



The convolution rule corresponding to EGF:

$$c_n = \sum_{k=0}^{n} a_k b_{n-k}$$

Example: lambda terms in de Bruijn notation



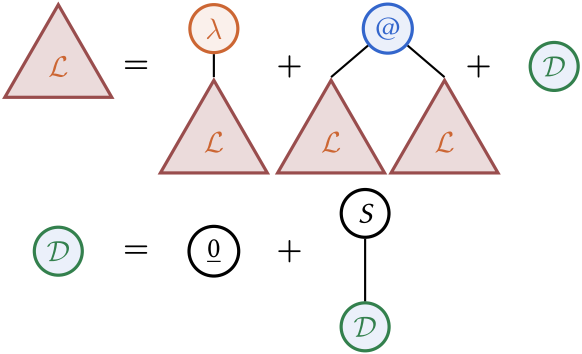
$$\lambda x.\lambda y.\lambda z.(xz)(yz) \longrightarrow \lambda\lambda\lambda(\underline{2}\underline{0})(\underline{1}\underline{0})$$

Classical notation ⟶ de Bruijn notation

Example: lambda terms in de Bruijn notation

$$\mathcal{L} ::= \lambda\mathcal{L} \mid (\mathcal{L}\mathcal{L}) \mid \underline{\mathsf{n}} \qquad L(z) = zL(z) + zL(z)^2 + \frac{z}{1-z}$$

$$\underline{\mathsf{n}} ::= \underline{0} \mid \mathsf{S}\underline{\mathsf{n}}. \qquad T_n = T_{n-1} + \sum_{k=1}^{n} T_k T_{n-k-1} + 1$$



Recursive sampling

Algorithm 1: Recursive algorithm for plain lambda terms

Input: Integer n
Output: Plain lambda term of size n
begin

Precompute array $(T_k)_{k=0}^{n}$ using the recurrence

$$T_n = T_{n-1} + \sum_{k=1}^{n} T_k T_{n-k-1} + 1 \quad , \qquad T_0 = 0$$

For each n, precompute the probability distribution $\mathcal{P}_n$:

$$p_\lambda^{(n)} = \frac{T_{n-1}}{T_n}, \qquad p_k^{(n)} = \frac{T_k T_{n-k-1}}{T_n}, \qquad p_{\underline{n}} = \frac{1}{T_n}$$

Function `Generate`(n):

if $n = 1$ **then**
  return $\underline{0}$ // minimal de Bruijn index ;

Sample index k from the probability distribution $\mathcal{P}_n$;

if $k = \lambda$ **then**
  return λ `Generate`($n-1$) // abstraction ;

if $k = \underline{n}$ **then**
  return $\underline{n}$ // de Bruijn index ;

$L :=$ `Generate`(k) ;
$R :=$ `Generate`($n-k-1$) ;
return (LR) // application ;

Boltzmann sampling

Algorithm 2: Boltzmann sampler for plain lambda terms

Input: Integer number n
Output: Random term of variable size, target expected size n
begin

Precompute z as a function of n // stay tuned

Function `Generate`(z):

Carefully look at the equation

$$L(z) = zL(z) + zL(z)^2 + \frac{z}{1-z}$$

Flip a weighted coin $X \in \{\lambda, @, \underline{\mathrm{n}}\}$ with weights

$$\mathbb{P}_\lambda = \frac{zL(z)}{L(z)}, \quad \mathbb{P}_@ = \frac{zL^2(z)}{L(z)}, \quad \mathbb{P}_{\underline{\mathrm{n}}} = \frac{\frac{z}{1-z}}{L(z)}$$

if $X = \lambda$ **then**
  return λ `Generate`$(n-1)$ // abstraction ;

if $X = @$ **then**
  $L :=$ `Generate`(z) ;
  $R :=$ `Generate`(z) ;
  return (LR) // application ;

if $X = \underline{\mathrm{n}}$ **then**
  return $\underline{\mathrm{Geom}(z)}$ // de Bruijn index ;

Boltzmann distribution

Probability output of the Boltzmann samplers

Let $S(z)$ be the generating function of the language $\mathcal{S}$:

$$S(z) = \sum_{n \geqslant 0} a_n z^n$$

Consider a distribution $\mathbb{P}_z$ on words $w \in \mathcal{S}$:

- conditioned on word length $|w| = n$, the distribution is uniform
- length distribution follows Gibbs law

$$\mathbb{P}_z(|w| = n) = \frac{a_n z^n}{S(z)}$$

- expected word length:

$$\mathbb{E}_z(n) = z\frac{S'(z)}{S(z)}$$

Multivariate generating functions

Consider a language $\mathcal{S} \subset \Sigma^*$ where $\Sigma = \{\bullet_1, \bullet_2, \bullet_3, \bullet_4\}$ is finite.
Let a_{n_1,n_2,n_3,n_4} count the number of words $w \in \mathcal{S}$ containing

- n_1 letters $\bullet_1$,
- n_2 letters $\bullet_2$,
- n_3 letters $\bullet_3$,
- n_4 letters $\bullet_4$;

Its multivariate generating function is

$$S(z_1, z_2, z_3, z_4) = \sum_{n \geqslant 0} a_{n_1,n_2,n_3,n_4} z_1^{n_1} z_2^{n_2} z_3^{n_3} z_4^{n_4} \quad .$$

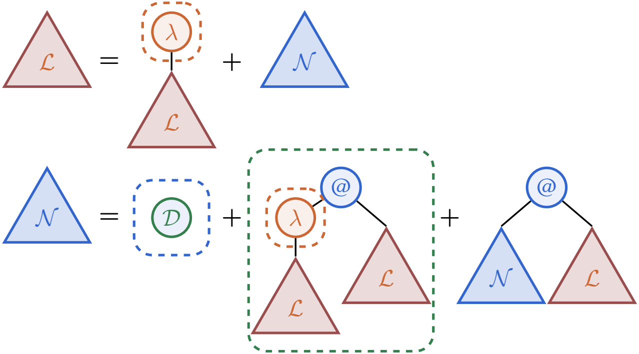
Boltzmann distribution

$$\mathbb{P}(n_1, n_2, n_3, n_4 \mid z_1, z_2, z_3, z_4) = \frac{a_{n_1,n_2,n_3,n_4} z_1^{n_1} z_2^{n_2} z_3^{n_3} z_4^{n_4}}{S(z_1, z_2, z_3, z_4)}$$

Example: lambda terms and their parameters

Abstractions, variables, successors and redexes marked separately:

$$
\begin{aligned}
L(z,\vec{u}) &= u_{(\mathrm{abs})} z L(z,\vec{u}) + N(z,\vec{u})\\
N(z,\vec{u}) &= \frac{u_{(\mathrm{var})} z}{1 - u_{(\mathrm{suc})} z} + u_{(\mathrm{red})} u_{(\mathrm{abs})} z^2 L(z,\vec{u})^2 + zN(z,\vec{u}) L(z,\vec{u}).
\end{aligned}
$$



Multiparametric Recursive sampling

Algorithm 3: Recursive algorithm for plain lambda terms with parameters

Input: Target integer parameters $N, n_{(\mathrm{abs})}, n_{(\mathrm{var})}, n_{(\mathrm{suc})}, n_{(\mathrm{red})}$

Output: Random term with target size N, and given parameter values

begin

Precompute array $(T_{K,k_1,k_2,k_3,k_4})_{k,k_1,k_2,k_3,k_4}$ using the recurrence

$$\ldots$$

Complexity

Memory complexity is now $\mathcal{O}(n^5)$ and grows exponentially with the number of parameters.

Multiparametric Boltzmann sampling

Plain lambda terms with given portions of abstractions, variables, successors and redexes

$$L(z,\vec{u}) = u_{(\mathrm{abs})} z L(z,\vec{u}) + N(z,\vec{u})$$
$$N(z,\vec{u}) = \frac{u_{(\mathrm{var})} z}{1 - u_{(\mathrm{suc})} z} + u_{(\mathrm{red})} u_{(\mathrm{abs})} z^2 L(z,\vec{u})^2 + z N(z,\vec{u}) L(z,\vec{u}).$$

Algorithm 4: Boltzmann sampler for plain lambda terms

Input: Target expectations $N, n_{(\mathrm{abs})}, n_{(\mathrm{var})}, n_{(\mathrm{suc})}, n_{(\mathrm{red})}$
Output: Random term with target expected size N, and given expected parameters

begin

Precompute $\left(z, u_{(\mathrm{abs})}, u_{(\mathrm{var})}, u_{(\mathrm{suc})}, u_{(\mathrm{red})}\right)$ as functions of $\left(N, n_{(\mathrm{abs})}, n_{(\mathrm{var})}, n_{(\mathrm{suc})}, n_{(\mathrm{red})}\right)$

// stay tuned;

Function $\Gamma L\,(z, u_{(abs)}, u_{(var)}, u_{(suc)}, u_{(red)})$**:**

Generate $X \in \{0,1\}$ such that

$$\mathbb{P}(X=0) = \frac{u_{(\mathrm{abs})} z L(z,\vec{u})}{L(z,\vec{u})},$$
$$\mathbb{P}(X=1) = \frac{N(z,\vec{u})}{L(z,\vec{u})}$$

$X = 0 \Rightarrow$ **return** $\lambda \Gamma L(z,\vec{u})$;
$X = 1 \Rightarrow$ **return** $\Gamma N(z,\vec{u})$;

Function $\Gamma N\,(z, u_{(abs)}, u_{(var)}, u_{(suc)}, u_{(red)})$**:**

Generate $X \in \{0,1,2\}$ such that

$$\mathbb{P}(X=0) = \frac{\frac{u_{(\mathrm{var})} z}{1 - u_{(\mathrm{suc})} z}}{N(z,\vec{u})},$$
$$\mathbb{P}(X=1) = \frac{u_{(\mathrm{red})} u_{(\mathrm{abs})} z^2 L(z,\vec{u})^2}{N(z,\vec{u})},$$
$$\mathbb{P}(X=2) = \frac{z N(z,\vec{u}) L(z,\vec{u})}{N(z,\vec{u})}$$

$X = 0 \Rightarrow$ **return** $\underline{\mathrm{Geom}(z u_{(suc)})}$;
$X = 1 \Rightarrow$ **return** $(\lambda \Gamma L(z,\vec{u}))\Gamma L(z,\vec{u})$;
$X = 1 \Rightarrow$ **return** $(\Gamma N(z,\vec{u}) \Gamma L(z,\vec{u}))$;

Why Boltzmann sampling?

Exact multiparametric sampling

Suppose that S_i are defined by an unambiguous context-free grammar

$$S_i \to \sum_j T_{ij}(S_1, \dots, S_n, \bullet_1, \bullet_2, \bullet_3, \bullet_d)$$

where $(T_{ij})_{ij}$ are transitions, and $(\bullet_1, \bullet_2, \bullet_3, \bullet_d)$ are alphabet letters.

Problem

Given positive integers $(n_1, n_2, \dots, n_d)$, sample a word w with n_k literals of color k from a context-free grammar uniformly at random;

Complexity

Exact multiparametric sampling from CFG is $\#P$-complete, i.e. higher in the complexity hierarchy than NP-complete.

Exact sampling is #P-complete: reduction from #2-SAT

[Welsh, Gale '2001] + [Jerrum, Valiant, Vazirani '86] + [Bendkowski, Bodini, D. '2020]

Consider a 2-CNF formula

$$F = \underbrace{(x_1 \vee \bar{x}_2)}_{c_1} \underbrace{(x_1 \vee \bar{x}_4)}_{c_2} \underbrace{(\bar{x}_2 \vee \bar{x}_3)}_{c_3} \underbrace{(\bar{x}_2 \vee \bar{x}_4)}_{c_4} \underbrace{(\bar{x}_3 \vee x_4)}_{c_5}$$

Construct a system of algebraic equations

$$A(c_1, \ldots, c_5) = (x_1 + \bar{x}_1) \ldots (x_4 + \bar{x}_4)(1 + c_1) \ldots (1 + c_5)$$

where

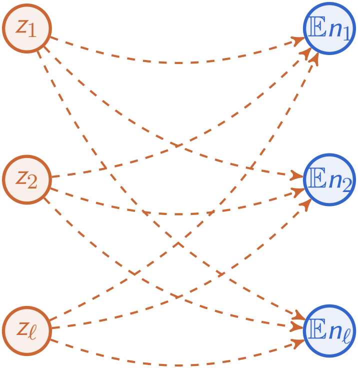
$$x_1 = c_1 c_2, \quad \bar{x}_1 = 1, \quad x_2 = 1, \quad \bar{x}_2 = c_1 c_3 c_4, \quad \bar{x}_3 = c_3 c_5, \quad \cdots$$

Then, using the notation $[\boldsymbol{z}^{\boldsymbol{n}}]F(\boldsymbol{z}) = \boldsymbol{n}$-th coefficient of $F(\boldsymbol{z})$,

$$\#2SAT(F) = [c_1^2 c_2^2 \ldots c_5^2] A(c_1, \ldots, c_5)$$

Tuning of a multiparametric Boltzmann sampler

Handles $\boldsymbol{z}$ $\Rightarrow$ Expectations $\mathbb{E}n_k$



‼ The handles cannot be tuned independently ‼

Multiparametric tuning complexity

Theorem (Bendkowski, Bodini, D. '2020)

For context-free grammars with L states and transitions, d parameters and target size parameter n, there is a tuning algorithm running in time

$$\mathcal{O}(d^{3.5} L \log n)$$

based on convex optimisation with barriers.

Proof idea: log-sum-exp with non-negative coefficients is convex

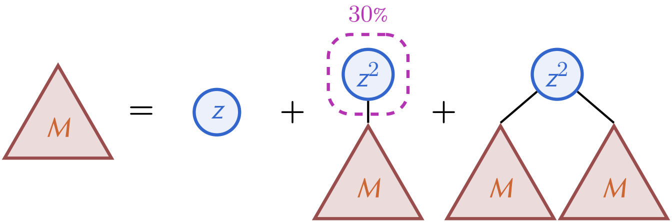
$$f(x_1, \ldots, x_n) = \log \sum_{i=1}^{n} e^{x_i}$$

generating functions are reduced to log-sum-exp by variable change

Part II. Applications

Boltzmann Brain + Paganini

Grammar example: Motzkin trees with non-uniform weights



$$M(z) = z + uz^2 M(z) + z^2 M^2(z)$$

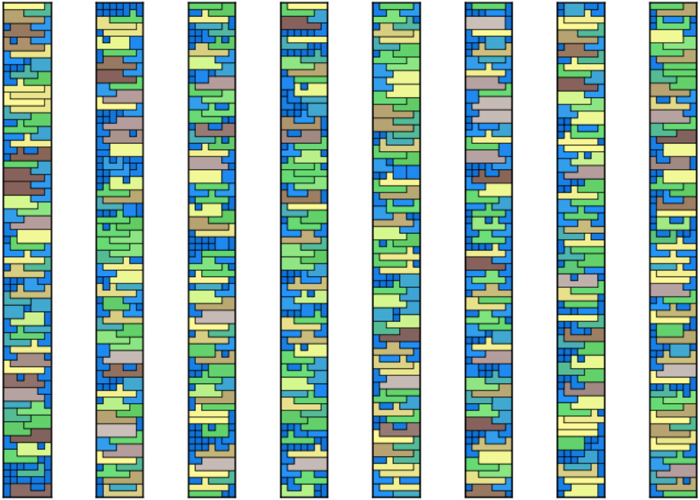
```
-- Motzkin trees
MotzkinTree = Leaf
            | Unary MotzkinTree (2) [0.3]
            | Binary MotzkinTree MotzkinTree (2).
```

Applications and examples

1. Polyomino tilings
2. Software testing using lambda calculus
3. Models of random trees
4. RNA folding design
5. Bose–Einstein condensate in quantum harmonic oscillator
6. Permutation classes

Application 1. Tilings and performance benchmarks

Tiling example, practical benchmark



Tiling example, practical benchmark

Tilings $9 \times n$ form a regular grammar with

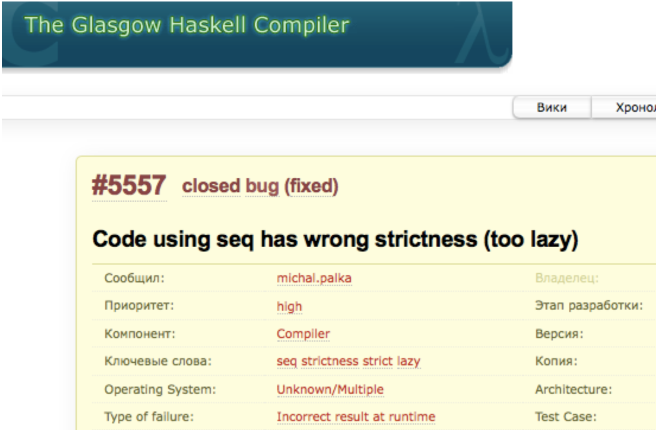
- 1022 tuning parameters
- 19k states
- 357k transitions

We tune for a uniform distribution for tile frequency.
This results in **few hours** of tuning.

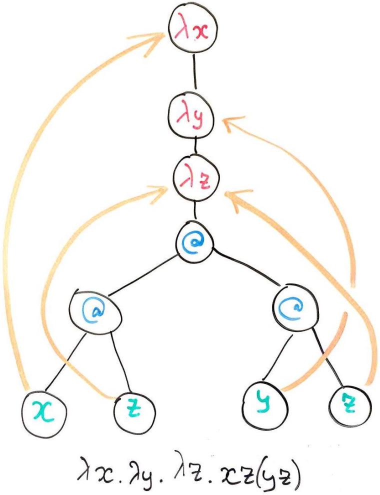
Application 2. Software testing using lambda calculus

Application 2: software testing

Goal: finding bugs in optimising compilers using **corner-case** random sampling of simply typed lambda terms



Application 2: software testing



- **Plain lambda terms**: Motzkin trees whose leaves contain non-negative integers.
- **Closed lambda terms**: Plane lambda terms whose leaf values do not exceed their unary height.
- **Holy grail:** simply typed lambda terms (in progress)

Application 2: software testing

Tuning uniform leaf index frequencies from 0 to 8:

TABLE 3. Empirical frequencies (with respect to the term size) of index distribution.

Index	0	1	2	3	4	5	6	7	8
Tuned frequency	8.00%	8.00%	8.00%	8.00%	8.00%	8.00%	8.00%	8.00%	8.00%
Observed frequency	7.50%	7.77%	8.00%	8.23%	8.04%	7.61%	8.53%	7.43%	9.08%
Default frequency	21.91%	12.51%	5.68%	2.31%	0.74%	0.17%	0.20%	0.07%	- - -

Can be also tuned:

- number of atomic nodes of distinguished colors
- number of **redexes** (i.e. patterns necessary to perform a computation step in lambda calculus)
- number of head abstractions
- number of closed subterms
- number of any tree-like patterns

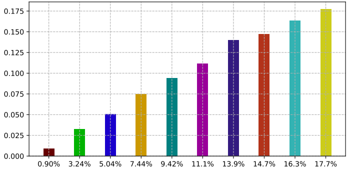
Application 3. Models of random trees

Application 3. Models of random trees

Model 1: Multi-partite rooted labelled trees

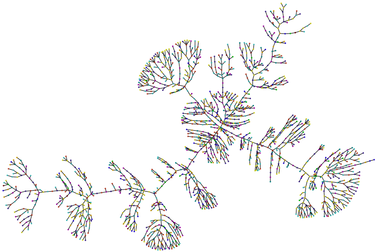
Target expectation tuning $(0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.13, 0.15, 0.17, 0.19)$

$$\begin{aligned} T_1(z, u_1, \ldots, u_d) &= z u_1 e^{T_2(z, u_1, \ldots, u_d)} \\ T_2(z, u_1, \ldots, u_d) &= z u_2 e^{T_3(z, u_1, \ldots, u_d)} \\ &\vdots \\ T_d(z, u_1, \ldots, u_d) &= z u_d e^{T_1(z, u_1, \ldots, u_d)} \end{aligned}$$



z	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8	u_9	u_{10}
0.3	0.009	1.88	1.37	1.29	1.26	1.25	1.24	1.23	1.23	3.52

Table 1: Numerical values for arguments



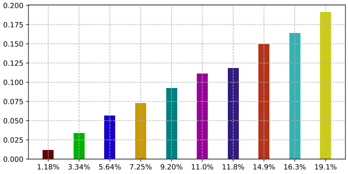
Application 3. Models of random trees

Model 2: Otter trees with coloured leaves

Target expectation tuning $(0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.13, 0.15, 0.17, 0.19)$

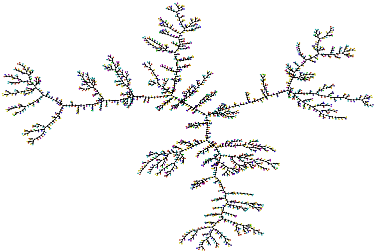
$$T(z, u_1, \ldots, u_d) = z \sum_{i=1}^{d} u_i + \mathrm{MSet}_2(T(z, u_1, \ldots, u_d))$$

$$\mathrm{MSet}_2(T(z, u_1, \ldots, u_d)) = \frac{T(z, u_1, \ldots, u_d)^2 + T(z^2, u_1^2, \ldots, u_d^2)}{2}$$



$u_1 z$	$u_2 z$	$u_3 z$	$u_4 z$	$u_5 z$	$u_6 z$	$u_7 z$	$u_8 z$	$u_9 z$	$u_{10} z$
0.005	0.015	0.025	0.035	0.044	0.054	0.063	0.072	0.081	0.09

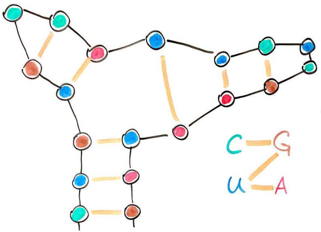
Table 2: Numerical values for arguments



Application 4. RNA folding design

Application 4: RNA folding design

[Hammer, Ponty, Wang, Will '2019]

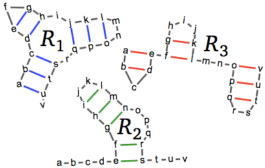


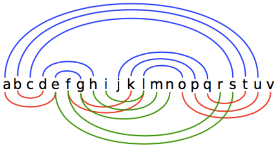
- **Problem.** Given the set of allowed secondary structures $(s_1, \cdots, s_k)$, sample uniformly at random RNA satisfying each of those structures.
- **Proposition.** The problem is equivalent to enumerating independent sets in bipartite graphs

Application 4: RNA folding design

image taken from [Hammer, Ponty, Wang, Will '2019]

Step 1: construct a graph based on secondary structures

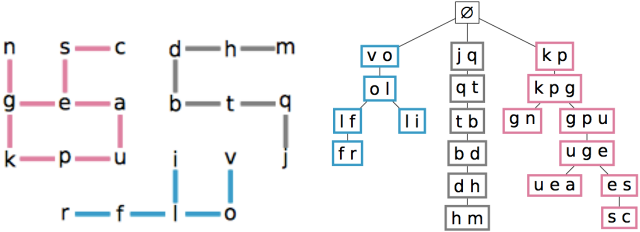




Application 4: RNA folding design

image taken from [Hammer, Ponty, Wang, Will '2019]

Step 2: construct a suitable tree decomposition and a context-free grammar



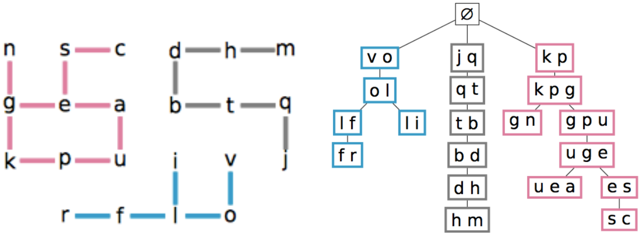
$$m_{\{uge\}\to\{pgu\}}(x_g, x_u) = \sum_{\text{allowed } x_e} \left(m_{\{uea\}\to\{uge\}}(x_u, x_e)\right)\left(m_{\{es\}\to\{uge\}}(x_e)\right)$$

Application 4: RNA folding design

image taken from [Hammer, Ponty, Wang, Will '2019]

Step 3: add the parameters

- each secondary structure energy (marked by u_c)
- letter frequency



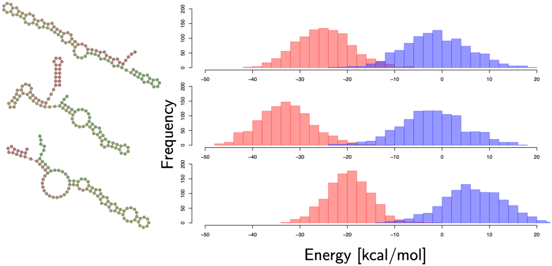
$$m_{u \to v}(x) = \sum_{\widetilde{x}} \prod_{w \to u} m_{w \to u}(x, \widetilde{x}) \times u_c^{-\text{energy of added edge}}$$

Application 4: RNA folding design

image taken from [Hammer, Ponty, Wang, Will '2019]

Conclusion:

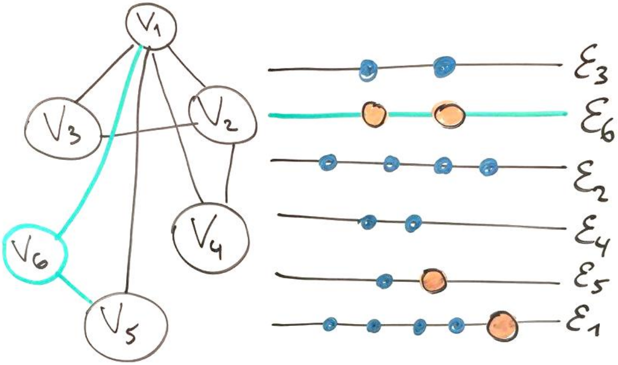
- The energies of the secondary structures and letter frequencies can be tuned
- This can be subsequently refined to energies of adjacent pairs in RNA sequence, triples, etc.
- Empirically observed energy distributions are Gaussian



Application 5: Bose–Einstein condensate in quantum harmonic oscillator

Bianconi–Barabási model

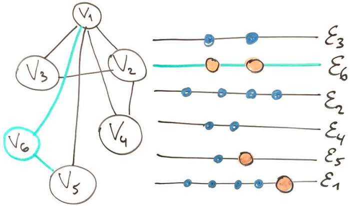
An evolving network can be compared to a diluted gas at low temperature



Bose–Einstein condensation in evolving networks

Bianconi–Barabási model

Bose gas	**network evolution**
temperature	temperature
energy	energy
particle	half-edge
number of energy levels	$\leqslant$ number of nodes
Bose–Einstein condensation	topological phase transition



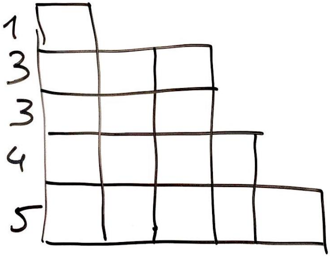
In this model, the number of particles on the energy level ε follows the Bose statistics $n(\varepsilon) = \frac{1}{e^{\beta(\varepsilon-\mu)}-1}$ which also represents the number of edges linking to nodes with energy ε.

Application 5: Bose–Einstein condensate in quantum harmonic oscillator

[Bernstein, Fahrbach, Randall], [Bendkowski, Bodini, D.]

Integer partitions $\leftrightarrow$ 1-dimensional quantum oscillator

$$16 = 1 + 3 + 3 + 4 + 5$$



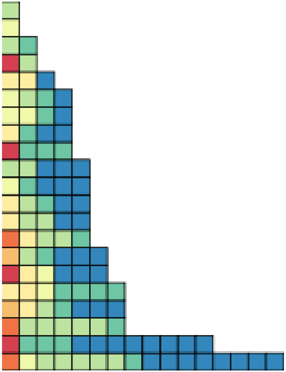
$$\text{partitions} = \text{multiset}(\mathbb{N}) = \text{multiset}(\text{multiset}(1))$$

Application 5: Bose–Einstein condensate in quantum harmonic oscillator

[Bernstein, Fahrbach, Randall], [Bendkowski, Bodini, D.]

Coloured partitions $\leftrightarrow$ d-dimensional quantum oscillator

$$\text{coloured partitions} = \text{multiset}\binom{\mathbb{N} + d - 1}{\mathbb{N}} = \mathrm{MSet}(\mathrm{MSet}(d \cdot 1))$$

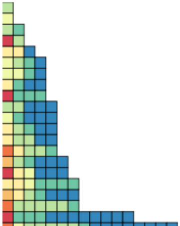


Application 5: Bose–Einstein condensate in quantum harmonic oscillator

[Bernstein, Fahrbach, Randall], [Bendkowski, Bodini, D.]

Coloured partitions $\leftrightarrow$ d-dimensional quantum oscillator

Weighted partition	Random particle assembly
Sum of numbers	Total energy
Number of colours	Dimension (d)
Row of Young table	Particle
Number of rows	Number of particles
Number of squares in the row	Energy of a particle (λ)
Partition limit shape	Bose–Einstein condensation
$\binom{d+\lambda-1}{\lambda}$	Number of particle states



Problem: generate random assemblies with given numbers of colours $(n_1, n_2, \ldots, n_d)$.

Application 5: Bose–Einstein condensate in quantum harmonic oscillator

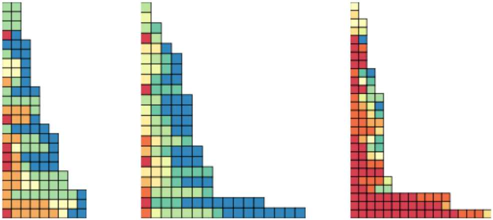
[Bernstein, Fahrbach, Randall], [Bendkowski, Bodini, D.]

Challenge: express the inner generating function

$$MSET(\bullet_1, \bullet_2, \cdots, \bullet_\ell) = \frac{1}{1 - z_1} \cdot \frac{1}{1 - z_2} \cdot \cdots \cdot \frac{1}{1 - z_\ell} - 1$$

in DCP rules using only polynomial number of additions and multiplications.

Solution: convexity proof of length $\Theta(\ell^2)$ using dynamic programming.



(A) [5,10,15,20,25] (B) [4,4,4,4,10,20,30,40] (C) [80,40,20,10,9,8,7,6,5]

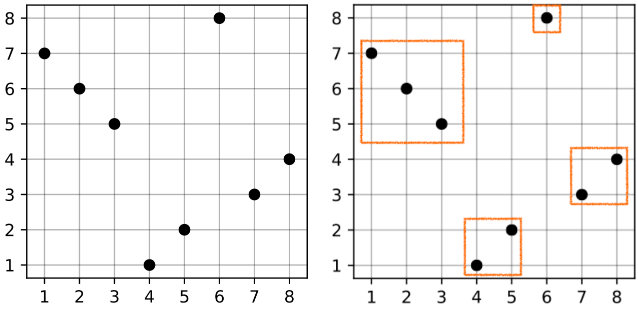
Application 6: Substitution-closed permutation classes

Simple permutations and inflations

- *Simple* permutation: does not contain *intervals*

$$\{a, a+1, ..., b\} \to \{c, c+1, ..., d\}$$

  of length strictly between 1 and n. Permutation from the figure is not simple because it contains an interval $\{1, 2, 3\} \to \{5, 6, 7\}$.
- *Inflation* is obtained by replacing each entry by interval



Substitution-closed classes

Theorem (Albert, Atkinson '2005)

Let $\mathcal{C}$ be substitution-closed and contain 12 and 21. Let $\mathcal{S}$ be the class of all simple permutations contained in $\mathcal{C}$. Then, $\mathcal{C}$ satisfies

$$\begin{aligned}
\mathcal{C} &= \{\bullet\} + 12[\mathcal{C}^+, \mathcal{C}] + 21[\mathcal{C}^-, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}] \\
\mathcal{C}^+ &= \{\bullet\} + 21[\mathcal{C}^-, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}] \\
\mathcal{C}^- &= \{\bullet\} + 12[\mathcal{C}^+, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}].
\end{aligned}$$

Remark

Algorithm for computing specifications of permutation classes containing finitely many simple permutations is given in [Bassino, Bouvel, Pierrot, Pivoteau, Rossin '2017]

Substitution-closed classes

Expected number of simple permutations $\pi \in \mathcal{S}$

$$\mathcal{C} = \{\bullet\} + 12[\mathcal{C}^+, \mathcal{C}] + 21[\mathcal{C}^-, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} u_\pi \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}]$$

$$\mathcal{C}^+ = \{\bullet\} + 21[\mathcal{C}^-, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} u_\pi \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}]$$

$$\mathcal{C}^- = \{\bullet\} + 12[\mathcal{C}^+, \mathcal{C}] + \sum_{\pi \in \mathcal{S}} u_\pi \pi[\mathcal{C}, \mathcal{C}, \ldots, \mathcal{C}].$$

By tuning the expectations attached to $(u_\pi)_{\pi \in \mathcal{S}}$, we can alter the expected frequencies of inflation used during the construction of a permutation.

Conclusion

Conclusion

1. Boltzmann sampler is a fundamental tool for multiparametric sampling. The tuning procedure is very natural in many contexts.
2. Context-free unambiguous grammars are ubiquitous in many areas of mathematics, physics and computer science.
3. Behind the scenes:
   - An $\mathcal{O}(n^{d/2})$ algorithm with $\mathcal{O}(\log n)$ memory for exact multiparametric sampling
   - The tuning algorithm can be interpreted in terms of Maximum Likelihood Estimation for combinatorial objects
   - Other frameworks: unlabelled structures and finite differential equations
   - Other applications: multiclass queues, hidden parameter estimation, random graphs with weighted degrees and patterns
   - Theory of self-concordant barriers for interior-point optimisation
   - Precise expected complexity and fine-tuning for rejections

Thank you for your attention