Multiparametric Boltzmann sampling: on the crossroad of probability and convex optimisation

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

[a]Jagellonian University, Krakow

[b]LIPN, Université Sorbonne Paris Nord

SPOC Seminar, 13/10/2021

COFINANCÉ PAR L'UNION EUROPÉENNE

REGION BOURGOGNE FRANCHE COMTE

Plan

1. Introduction to random sampling
2. Unambiguous context-free grammars
3. Exact multiparametric sampling is NP-hard
4. Boltzmann sampling
5. Tuning a Boltzmann sampler
6. Convex optimisation complexity
7. Implementation
8. Applications and examples

Introduction

Random sampling

Problem

Let $\boldsymbol{\theta} \in \mathbb{R}^d$, $\mathbb{P}_{\boldsymbol{\theta}}$ be a given probability distribution on strings*.

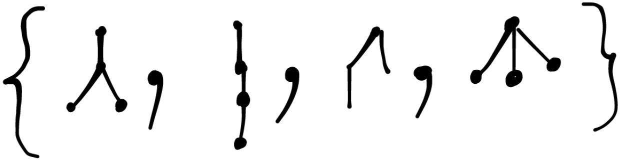
$$\boxed{\text{Sample } X \sim \mathbb{P}_{\boldsymbol{\theta}}}$$

$*$ All *discrete objects* can be encoded by strings!

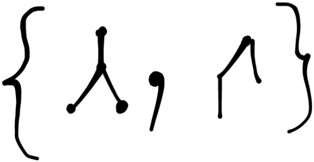
Some examples

Uniform sampling of rooted trees with 4 vertices

Uncontrolled sampling: $X \sim \mathbb{P}(X)$

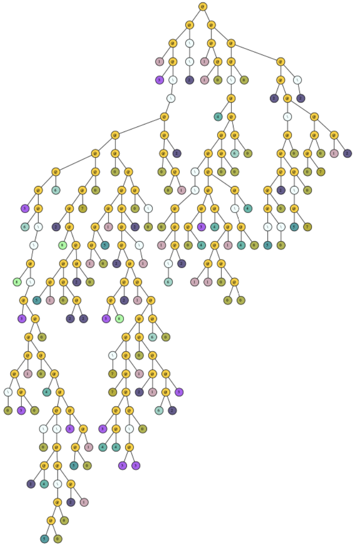


Controlled (parametric) sampling:
$X \sim \mathbb{P}(X \mid \text{number of leaves } = 2)$



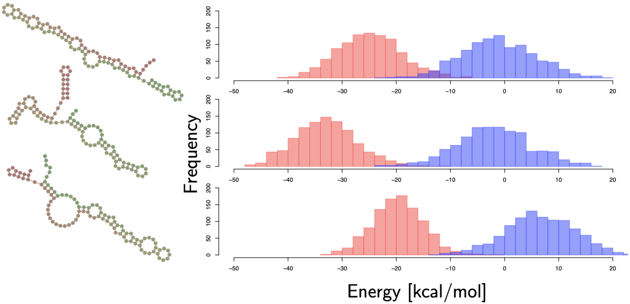
Some more examples

Closed lambda terms



Some more examples

RNA with given secondary structure energies



Some examples

- Trees with n nodes $\{\{\bullet\bullet\}\{\bullet\}\{\bullet\}\}$
- Graphs, networks (with given parametric properties)
- Tilings (with given number of tiles of each color)
- RNA sequences (with given pairing frequencies)
- lambda terms (with given proportion of beta-redexes)
- ...music with a given amount of sadness...

Why random sampling?

Motivations for random sampling

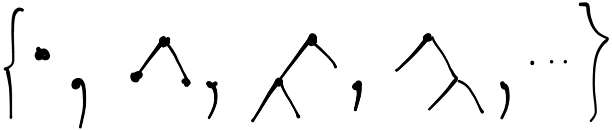
- Art and entertainment
  - T-shirt printing
  - Paintings, decorations, tilings
  - Music composition
  - Artificial intelligence artwork
- Monte-Carlo simulations
  - Property-based software testing (QuickCheck, lambda terms)
  - Biology (cell dynamics, RNA structures)
  - Statistical physics (random 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

Unambiguous context-free grammars

Unambiguous context-free grammars

Example

Binary trees



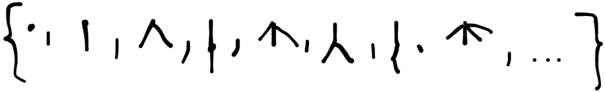
$$\{\bullet, \quad \bullet(\bullet)(\bullet), \quad \bullet(\bullet(\bullet)(\bullet))(\bullet), \quad \bullet(\bullet)(\bullet(\bullet)(\bullet)), \ldots\}$$

$$\boxed{T = \bullet \mid \bullet\,(T)(T)}$$

Weighted unambiguous context-free grammars

Example

Trees with $\leqslant 4$ children



$$\{(\bullet),\ (\bullet(\bullet)),\ (\bullet(\bullet)(\bullet)),\ (\bullet(\bullet(\bullet))),\ (\bullet(\bullet)(\bullet)(\bullet)), \ldots\}$$

Weighted grammar

$$\boxed{T = (\bullet_0) \mid (\bullet_1 T) \mid (\bullet_2 TT) \mid (\bullet_3 TTT) \mid (\bullet_4 TTTT)}$$

Color of the node reflects how many children it has.

Controlled sampling with rooted trees

Example

$$\boxed{T = (\bullet_0) \mid (\bullet_1 T) \mid (\bullet_2 TT) \mid (\bullet_3 TTT) \mid (\bullet_4 TTTT)}$$

- Randomly sample rooted trees with N nodes
- Quantity of $\bullet_1$ nodes is n_1
- Quantity of $\bullet_2$ nodes is n_2
- Quantity of $\bullet_3$ nodes is n_3
- Quantity of $\bullet_4$ nodes is n_3
- $n_1 + n_2 + n_3 + n_4 < N$

Exact multiparametric sampling

Let S_i be defined by an unambiguous context-free grammar (CFG)

$$S_i \to \Big|_j T_{ij}(S_1, \ldots, S_n, \bullet_1, \bullet_2, \bullet_3, \ldots, \bullet_d)$$

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

Problem

Given positive integers $(n_1, n_2, \ldots, n_d)$, 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 NP-hard

Exact sampling is NP-hard: reduction from #2-SAT

- [Jerrum, Valiant, Vazirani '86] If there is a fully polynomial *almost uniform generator* for $\mathcal{R}$, then there is a fully polynomial *randomized approximation scheme* (FPRAS) for $N_{\mathcal{R}}$.
- [Welsh, Gale '2001] Unless $NP = RP$, there is no FPRAS for #2-SAT
- [Bendkowski, Bodini, D. '2020] #2-SAT can be reduced to counting the number of words in unambiguous context-free grammars

$\Rightarrow$ exact sampling cannot be approximated unless $NP = RP$

Exact sampling is NP-hard: reduction from #2-SAT

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

Example. Consider a 2-CNF formula

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

Step 1.

Consider a grammar with an initial state A:

- $A = (X_1 + \overline{X}_1) \ldots (X_4 + \overline{X}_4)$
- $X_1 = c_1 c_2, \quad \overline{X}_1 = 1, \quad X_2 = 1, \quad \overline{X}_2 = c_1 c_3 c_4, \cdots$

Then, $\#2SAT(F) = \#\left\{w \leftarrow A \,\middle|\, |w|_{c_1} \geqslant 1, \ldots, |w|_{c_5} \geqslant 1\right\}$

reduction from #2-SAT (continuation)

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

- $A = (X_1 + \overline{X}_1) \dots (X_4 + \overline{X}_4)$
- $X_1 = c_1 c_2, \quad \overline{X}_1 = 1, \quad X_2 = 1, \quad \overline{X}_2 = c_1 c_3 c_4, \cdots$

$$\#2SAT(F) = \#\left\{w \leftarrow A \,\middle|\, |w|_{c_1} \geqslant 1, \dots, |w|_{c_5} \geqslant 1\right\}$$

Step 2.

- $B = A(1 + c_1) \dots (1 + c_5)$

$$\#2SAT(F) = \#\left\{w \leftarrow B \,\middle|\, |w|_{c_1} = 2, \dots, |w|_{c_5} = 2\right\}$$

What does it all mean?

Weighted unambiguous context-free grammars can encode a lot of things...

We can take that to our advantage!

But we cannot sample efficiently – very hard! What to do?

Boltzmann sampling

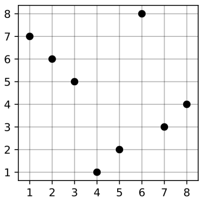
Generating functions and the symbolic method

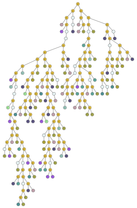
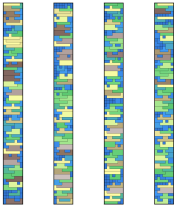
Framework

- 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:

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



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 $w \in \mathcal{S}$:

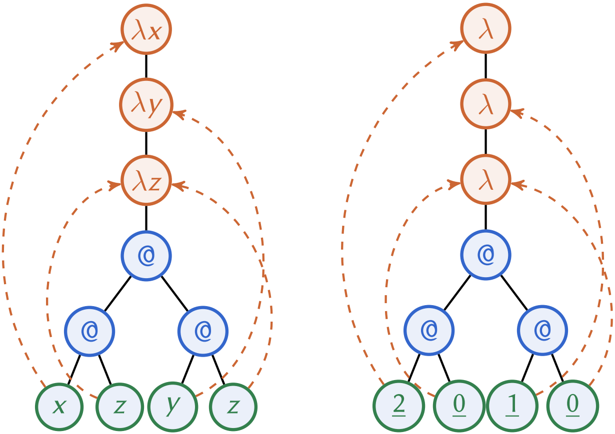
- conditioned on word length $|w| = n$, the distribution is uniform
- length distribution follows Gibbs law

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

- expected word length:

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

Example: lambda terms in de Bruijn notation

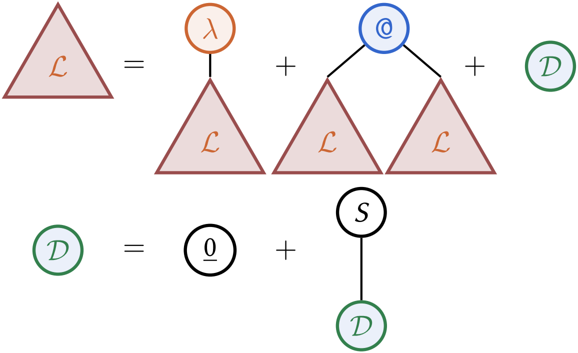


$\lambda x.\lambda y.\lambda z.(xz)(yz)$	$\longrightarrow$	$\lambda\lambda\lambda(\underline{2}\underline{0})(\underline{1}\underline{0})$
Classical notation	$\longrightarrow$	de Bruijn notation

Example: lambda terms in de Bruijn notation

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

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



Boltzmann sampling

Algorithm 1: 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 $X \in \{\lambda, @, \underline{n}\}$ with weights

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

if $X = \lambda$ **then**

  return λ `Generate`$(n-1)$ // abstraction ;

if $X = @$ **then**

  $L :=$ `Generate`(z) ;

  $R :=$ `Generate`(z) ;

  return (LR) // application ;

if $X = \underline{n}$ **then**

  return $\underline{\mathrm{Geom}(z)}$ // de Bruijn index ;

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

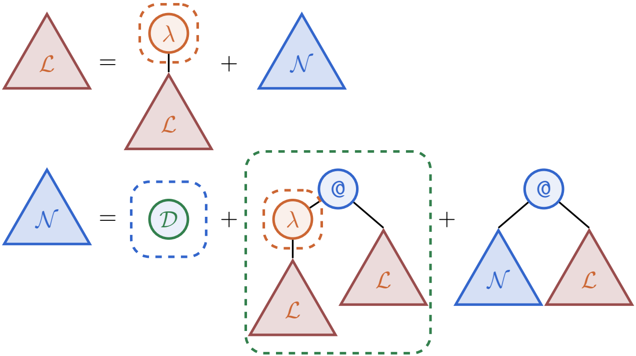
$$\mathbb{P}(n_1, n_2, n_3, n_4 \mid z_1, z_2, z_3, z_4) = \frac{a_{n_1,n_2,n_3,n_4} z_1^{n_1} z_2^{n_2} z_3^{n_3} z_4^{n_4}}{S(z_1, z_2, z_3, z_4)}$$

Example: lambda terms and their parameters

Abstractions, variables, successors and redexes marked separately:

$$L(z, \vec{u}) = u_{(\text{abs})} z L(z, \vec{u}) + N(z, \vec{u})$$

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



Multiparametric Boltzmann sampling

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

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

Algorithm 2: 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 $\Gamma L\left(z, u_{(abs)}, u_{(var)}, u_{(suc)}, u_{(red)}\right)$**:**

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

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

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

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

$$\mathbb{P}(X=2) = \frac{zN(z,\vec{u})L(z,\vec{u})}{N(z,\vec{u})}$$

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

Tuning a Boltzmann sampler as convex optimisation

Mathematical formulation

$$S(z_1, z_2, z_3, \ldots, z_d) = \sum_{n \geqslant 0} a_{n_1, n_2, n_3, \ldots, n_d} z_1^{n_1} z_2^{n_2} z_3^{n_3} \cdots z_d^{n_d} \quad .$$

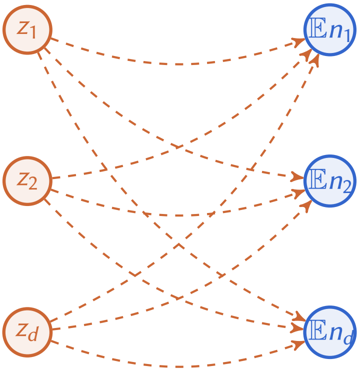
$$\mathbb{E}_{z_1, z_2, z_3, \ldots, z_d}(\xi_1) = z_1 \frac{\partial_{z_1} S(z_1, z_2, z_3, \ldots, z_d)}{S(z_1, z_2, z_3, \ldots, z_d)} = N_1,$$

$$\vdots$$

$$\mathbb{E}_{z_1, z_2, z_3, \ldots, z_d}(\xi_d) = z_d \frac{\partial_{z_d} S(z_1, z_2, z_3, \ldots, z_d)}{S(z_1, z_2, z_3, \ldots, z_d)} = N_d.$$

Tuning of a multiparametric Boltzmann sampler

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



‼ The handles cannot be turned independently ‼

Convex optimisation formulation

Universal version

Let $z_i = e^{\zeta_i}$. Tuning is equivalent to convex optimisation problem

$$\boxed{\begin{cases} \varphi - \boldsymbol{N}^\top \zeta \to \min_{\zeta, \varphi}, \\ \varphi \geqslant \log S(e^\zeta) \end{cases}}$$

Idea. $\log \sum e^{t_i}$ is a convex function.

$$\nabla_\zeta \left(\log S(e^\zeta) - \boldsymbol{N}^\top \zeta \right) = 0$$

if and only if

$$z_1 \frac{\partial_{z_1} S(z_1, z_2, z_3, \ldots, z_d)}{S(z_1, z_2, z_3, \ldots, z_d)} = N_1,$$

$$\vdots$$

$$z_d \frac{\partial_{z_d} S(z_1, z_2, z_3, \ldots, z_d)}{S(z_1, z_2, z_3, \ldots, z_d)} = N_d.$$

Case study: unambiguous context-free grammars

Let $\boldsymbol{\mathcal{C}} = \Phi(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{Z}})$ be a multi-parametric CFG:

- $\boldsymbol{\mathcal{C}} = (\mathcal{C}_1, \ldots, \mathcal{C}_m)$, sampling from the state $\mathcal{C}_1$
- $\Phi = (\Phi_1, \ldots, \Phi_m)$ is a transition matrix
- $\boldsymbol{\mathcal{Z}} = (\mathcal{Z}_1, \ldots, \mathcal{Z}_d)$ are distinct terminals
- $\boldsymbol{N} = (N_1, \ldots, N_d)$ is a tuning vector

Let $\boldsymbol{z} = e^{\boldsymbol{\xi}}$. The solution comes from the convex problem:

$$\begin{cases} c_1 - \boldsymbol{N}^\top \zeta \to \min_{\zeta, \boldsymbol{c}}, \\ \boldsymbol{c} \geqslant \log \Phi(e^{\boldsymbol{c}}, e^{\zeta}) \end{cases}$$

Convex optimisation complexity

Interior point method

Convex optimisation programs

$$\begin{cases} \boldsymbol{c}^\top \boldsymbol{z} \to \min_{\boldsymbol{z}} \\ f_i(\boldsymbol{z}) \leqslant 0 \text{ for } i = 1, \dots, m \end{cases}$$

Nesterov and Nemirovskii IPM:

$$\mathcal{O}\left(\sqrt{\nu}\log\left(\frac{\nu\mu_0}{\varepsilon}\right)\right) \text{ Newton iterations,}$$

where

- ν is the *self-concordance* parameter
- μ_0 is related to the choice of the starting point
- ε is the target precision (in the solution space)

Self-concordant functions

Just if you are curious to see the most used criterion

If $f(x)$ is a three times continuously differentiable real-valued convex function such that

$$|f'''(x)| \leqslant 3\beta x^{-1} f''(x), \quad x > 0$$

for some $\beta > 0$, then

$$F(t, x) = -\log(t - f(x)) - \max[1, \beta^2] \log x$$

is a self-concordant barrier with $\nu = 1 + \max[1, \beta^2]$.

- Positive linear combinations are also self-concordant with $\nu = \sum \alpha_i \nu_i$

Barriers for combinatorial constructions

- Context-free unambiguous: $\nu = O(\#\text{ of terms})$

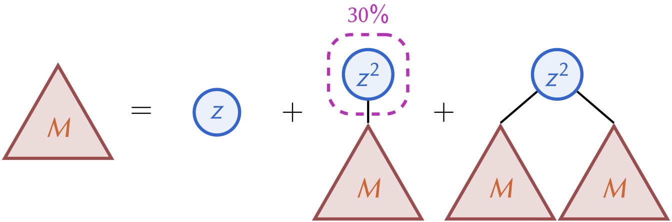
- Other constructions: cycles, sets, restricted cycles and sets

- More constructions: unlabelled cycles, multisets, ...

Implementation

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).
```

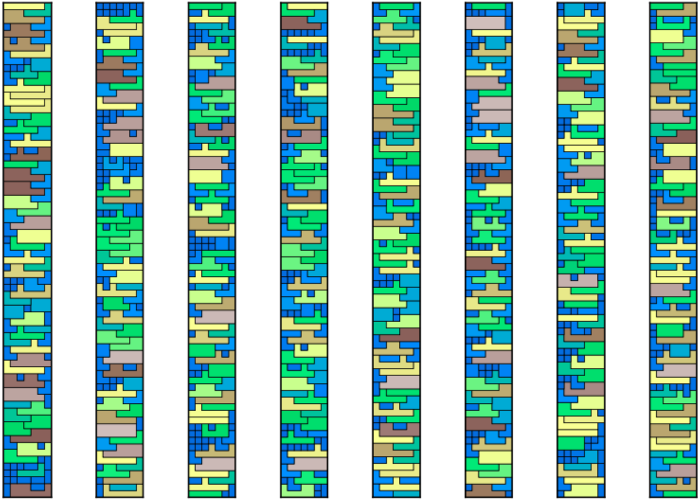
Tiling example, practical benchmark

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.

Tiling example, practical benchmark



Applications and examples

1. Combinatorial learning
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: Combinatorial learning

Application 1: Combinatorial learning

Example: hidden parameter estimation

Maximum likelihood estimate for Boltzmann distribution.

$$L(X_1, \ldots, X_n | z) = \sum_{i=1}^{n} \log \mathbb{P}(|X_i| = n \mid z) = \log \frac{a_{n_i} z^{n_i}}{F(z)}$$

$$= \sum \log a_{n_i} + \sum n_i \log z - n \log F(z) \to \max_z$$

We obtain the tuning equation:

$$\frac{\sum_{i=1}^{n} n_i}{n} = z \frac{F'(z)}{F(z)}$$

- **Hidden parameter estimation.** Objects are sampled from multivariate Boltzmann distribution $\boldsymbol{z} = (z_1, \ldots, z_k)$. We observe only a part of the parameters $(n_1^*, \ldots, n_\ell^*)$. Estimate $\boldsymbol{z}$.

Application 1: Combinatorial learning

Hidden parameter estimation

- **Hidden parameter estimation.** Objects are sampled from multivariate Boltzmann distribution $\boldsymbol{z} = (z, u)$. We observe only the parameter n corresponding to z. Estimate $\boldsymbol{z} = (z, u)$.
- Maximising the log-likelihood we obtain:

$$L(X_1, \ldots, X_n \mid z, u) = \sum_i \log \frac{\sum_k a_{n_i,k} z^{n_i} u^k}{F(z, u)} \to \max_{z,u}$$

- Multiparametric #P-complete problem:

$$\sum_{i=1}^{n} n_i - n\frac{\partial_z F}{F} = 0$$

$$\sum_{i=1}^{n} \frac{\partial_u [z^{n_i}] F(z, u)}{[z^{n_i}] F(z, u)} - n\frac{\partial_u F(z, u)}{F(z, u)} = 0$$

Application 1: Combinatorial learning

Hidden parameter estimation

- Multiparametric #P-complete problem:

$$\sum_{i=1}^{n} n_i - n\frac{\partial_z F}{F} = 0$$

$$\sum_{i=1}^{n} \frac{\partial_u [z^{n_i}] F(z,u)}{[z^{n_i}] F(z,u)} - n\frac{\partial_u F(z,u)}{F(z,u)} = 0$$

- Boltzmann relaxation:

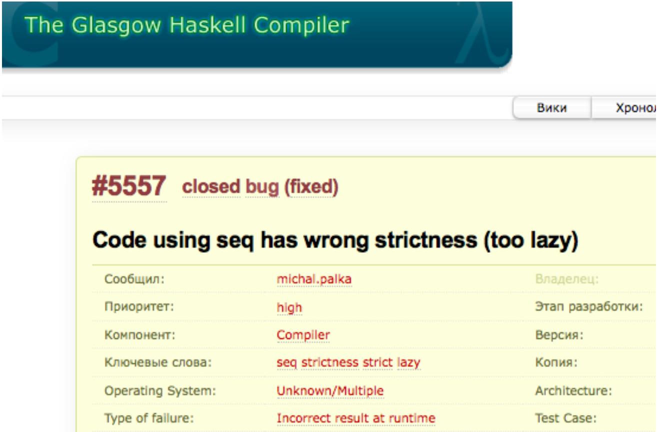
$$\frac{\partial_u [z^{n_i}] F(z,u)}{[z^{n_i}] F(z,u)} \approx \frac{\partial_u F(z^*(n_i),u)}{F(z^*(n_i),u)}$$

The parameter $z^*(n_i)$ can be found by the tuning procedure

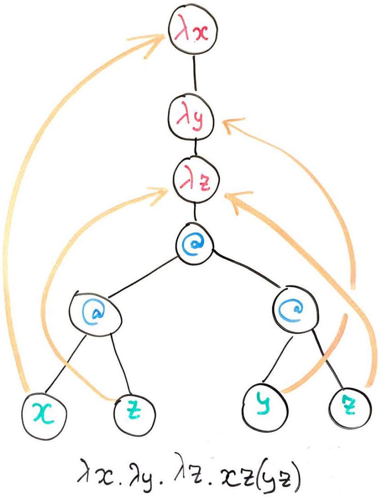
Application 2. Software testing using lambda calculus

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

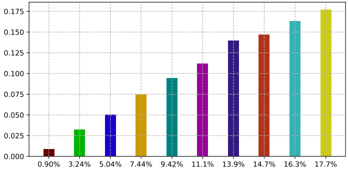
Application 3. Models of random trees

Application 3. Models of random trees

Model 1: Multi-partite rooted labelled trees

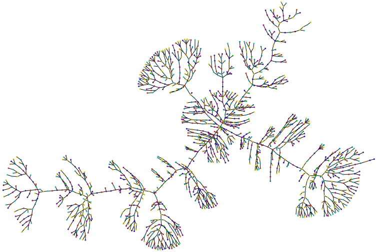
Target expectation tuning $(0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.13, 0.15, 0.17, 0.19)$

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



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



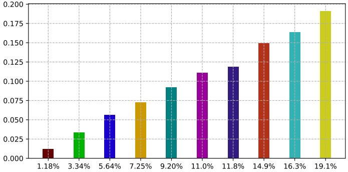
Application 3. Models of random trees

Model 2: Otter trees with coloured leaves

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

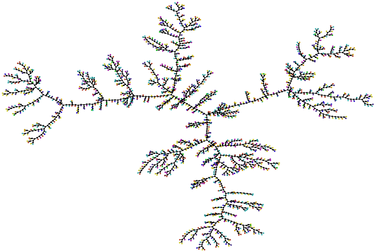
$$T(z, u_1, \ldots, u_d) = z \sum_{i=1}^{d} u_i + \mathrm{MSet}_2(T(z, u_1, \ldots, u_d))$$

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



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

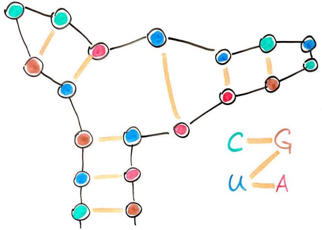
Table 2: Numerical values for arguments



Application 4. RNA folding design

Application 4: RNA folding design

[Hammer, Ponty, Wang, Will '2019]

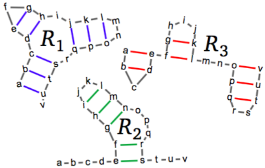


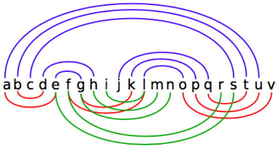
- **Problem.** Given the set of allowed secondary structures $(s_1, \cdots, s_k)$, sample uniformly at random RNA satisfying each of those structures.
- **Proposition.** The problem is equivalent to enumerating independent sets in bipartite graphs

Application 4: RNA folding design

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

Step 1: construct a graph based on secondary structures

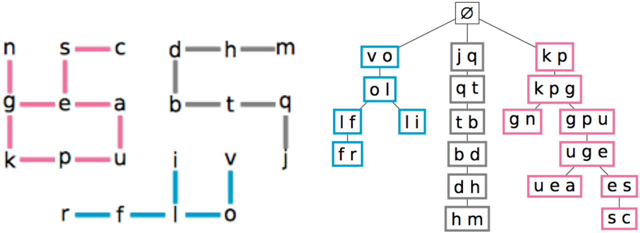




Application 4: RNA folding design

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

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



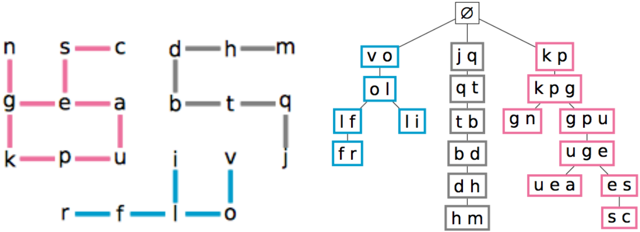
$$m_{\{uge\}\to\{pgu\}}(x_g, x_u) = \sum_{\text{allowed } x_e} \left(m_{\{uea\}\to\{uge\}}(x_u, x_e)\right) \left(m_{\{es\}\to\{uge\}}(x_e)\right)$$

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



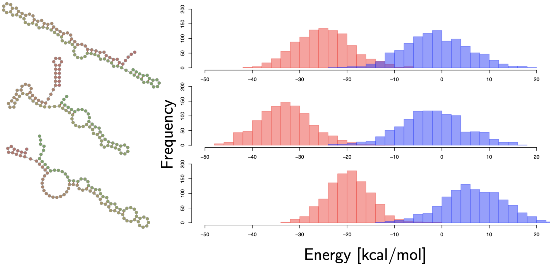
$$m_{u \to v}(x) = \sum_{\widetilde{x}} \prod_{w \to u} m_{w \to u}(x, \widetilde{x}) \times u_c \text{—energy of added edge}$$

Application 4: RNA folding design

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

Conclusion:

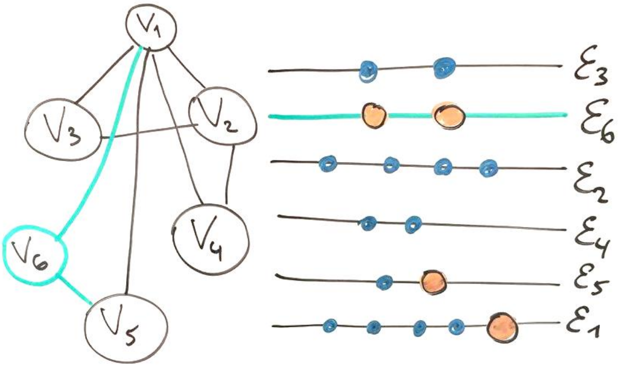
- The energies of the secondary structures and letter frequencies can be tuned
- This can be subsequently refined to energies of adjacent pairs in RNA sequence, triples, etc.
- Empirically observed energy distributions are Gaussian



Application 5: Bose–Einstein condensate in quantum harmonic oscillator

Bianconi–Barabási model

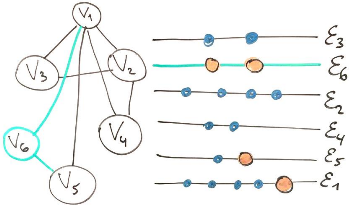
An evolving network can be compared to a diluted gas at low temperature



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

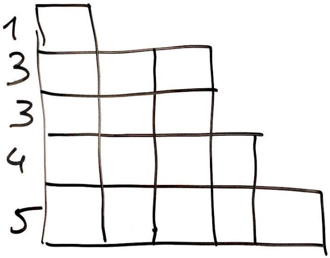


In this model, the number of particles on the energy level ε follows the Bose statistics $n(\varepsilon) = \frac{1}{e^{\beta(\varepsilon-\mu)}-1}$ which also represents the number of edges linking to nodes with energy ε.

Application 5: Bose–Einstein condensate in quantum harmonic oscillator

Integer partitions $\leftrightarrow$ 1-dimensional quantum oscillator

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



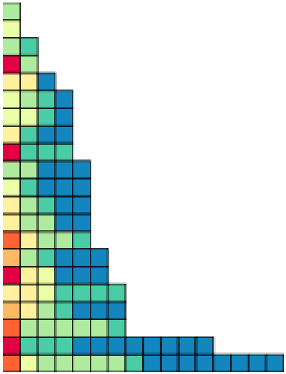
$$\text{partitions} = \text{multiset}(\mathbb{N}) = \text{multiset}(\text{multiset}(1))$$

Application 5: Bose–Einstein condensate in quantum harmonic oscillator

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

Coloured partitions $\leftrightarrow$ d-dimensional quantum oscillator

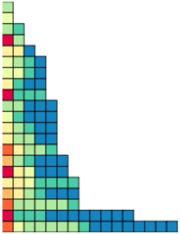
$$\text{coloured partitions} = \text{multiset}\binom{\mathbb{N}+d-1}{\mathbb{N}} = MSet(MSet(d \cdot 1))$$



Application 5: Bose–Einstein condensate in quantum harmonic oscillator

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 $(n_1, n_2, \ldots, n_d)$.

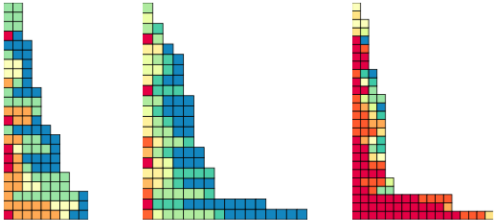
Application 5: Bose–Einstein condensate in quantum harmonic oscillator

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]

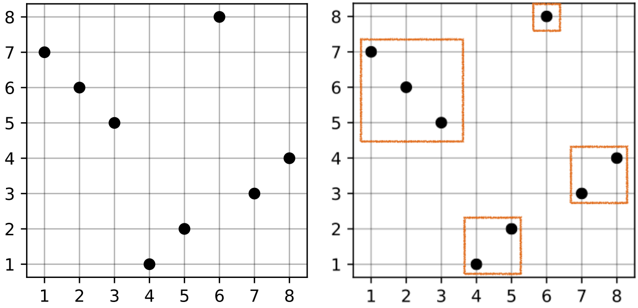
Application 6: Substitution-closed permutation classes

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*

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

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 $\pi \in \mathcal{S}$

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

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 relaxation of exact-parameter sampling. It samples in linear time and can be tuned very efficiently now (using a polynomial algorithm).
2. A wide variety of combinatorial classes can be reduced to unambiguous context-free grammar
3. To incorporate new exotic classes, convex optimisation and combinatorics should work together

Thank you for your attention