Polynomial tuning of multiparametric combinatorial samplers

Maciej Bendkowski[3] Olivier Bodini[1] Sergey Dovgal[1,2,4]

[1]Université Paris-13, [2]Université Paris-Diderot [3]Jagiellonian University [4]Moscow Institute of Physics and Technology

ALEA Days @CIRM 16.03.2018

Outline

Example problems for uniparametric sampling

- Regular languages

$$\mathcal{F} = \mathcal{A}\mathcal{F} + \mathcal{B}$$

- Algebraic specifications

$$\mathcal{F} = \mathbf{\Phi}(\mathcal{F}, \mathcal{Z})$$

- Pòlya structures

$$\mathcal{F}(z) = \mathbf{\Phi}(\mathcal{F}(z), \mathcal{F}(z^2), \mathcal{F}(z^3), \ldots, \mathcal{Z})$$

- Maps, differential specifications, DAG, trace monoids, lozenge tilings, walks, …

Example problems for uniparametric sampling

- Regular languages

$$\mathcal{F} = \mathcal{A}\mathcal{F} + \mathcal{B}$$

- Algebraic specifications

$$\mathcal{F} = \mathbf{\Phi}(\mathcal{F}, \boldsymbol{Z})$$

- Pòlya structures

$$\mathcal{F}(z) = \mathbf{\Phi}(\mathcal{F}(z), \mathcal{F}(z^2), \mathcal{F}(z^3), \ldots, \boldsymbol{Z})$$

- Maps, differential specifications, DAG, trace monoids, lozenge tilings, walks, ...

Example problems for uniparametric sampling

- Regular languages

$$\mathcal{F} = \mathcal{A}\mathcal{F} + \mathcal{B}$$

- Algebraic specifications

$$\mathcal{F} = \mathbf{\Phi}(\mathcal{F}, \boldsymbol{Z})$$

- Pòlya structures

$$\mathcal{F}(z) = \mathbf{\Phi}(\mathcal{F}(z), \mathcal{F}(z^2), \mathcal{F}(z^3), \ldots, \boldsymbol{Z})$$

- Maps, differential specifications, DAG, trace monoids, lozenge tilings, walks, ...

Example problems for uniparametric sampling

- Regular languages

$$\mathcal{F} = \mathcal{A}\mathcal{F} + \mathcal{B}$$

- Algebraic specifications

$$\mathcal{F} = \mathbf{\Phi}(\mathcal{F}, \boldsymbol{Z})$$

- Pòlya structures

$$\mathcal{F}(z) = \mathbf{\Phi}(\mathcal{F}(z), \mathcal{F}(z^2), \mathcal{F}(z^3), \ldots, \boldsymbol{Z})$$

- Maps, differential specifications, DAG, trace monoids, lozenge tilings, walks, …

Practical motivation for random sampling

Practical motivation for random sampling

- T-Shirt printing

Practical motivation for random sampling

- T-Shirt printing
- Monte-Carlo simulations
  - Biology (cell dynamics, RNA structures)
  - Statistical physics (random maps, Bose–Einstein condensate, Ising model, lozenge tilings, plane partitions)
  - Property-based testing (quickcheck, lambda terms, XML)
- Data analysis, computer science
  - Random graphs, community detection
  - Low-level programming (crypto primitives)
  - Concurrent process analysis, queue systems
  - Automata sampling

Contributions to uniparametric sampling

- Recursive method (exact size)
  [Nijenhuis, Wilf]
  [Bernardi, Denise, Flajolet, Giménez, Roques, Termier, Van Cutsem, Zimmerman, …]
- Boltzmann sampling (approximate size)
  [Duchon, Flajolet, Louchard, Schaeffer]
  [Bacher, Banderier, Bassino, Bernstein, Bodini, Bodirsky, David, Darasse, Dien, Fahrbach, Fontaine, Fusy, Gardy, Genitrini, Hwang, Jacquot, Kang, Lumbroso, Marchal, Nicaud, Pivoteau, Ponty, Randall, Rolin, Roussel, Salvy, Soria, Sportiello, Vigerske, …]

Contributions to uniparametric sampling

- Recursive method (exact size)
  [Nijenhuis, Wilf]
  [Bernardi, Denise, Flajolet, Giménez, Roques, Termier, Van Cutsem, Zimmerman, …]
- Boltzmann sampling (approximate size)
  [Duchon, Flajolet, Louchard, Schaeffer]
  [Bacher, Banderier, Bassino, Bernstein, Bodini, Bodirsky, David, Darasse, Dien, Fahrbach, Fontaine, Fusy, Gardy, Genitrini, Hwang, Jacquot, Kang, Lumbroso, Marchal, Nicaud, Pivoteau, Ponty, Randall, Rolin, Roussel, Salvy, Soria, Sportiello, Vigerske, …]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Computational cost of uniparametric sampling

- Recursive method without floating point optimization: $O(n^{3+\epsilon})$;
  [Nijenhuis and Wilf '78; Flajolet, Zimmerman, Van Cutsem '94]
- Recursive method: average $O(n \log n)$, worst case $O(n^2)$;
  [Denise, Zimmerman '99]
- Boltzmann exact: $O(n^2)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- approximate Boltzmann with $\sqrt{n}$ error: $O(1)$;
  [Duchon, Flajolet, Louchard, Schaeffer '02]
- Recursive sampler for rational grammars: $O(n)$;
  [Bernardi, Gimenéz '12]
- Boltzmann for rational grammars with $O(1)$ error: $O(n)$;
  Corollary of [Duchon, Flajolet, Louchard, Schaeffer '02]

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Tuning of uniparametric Boltzmann sampler

$$z\frac{\partial_z F(z)}{F(z)} = n, \quad z = z(n) = ?$$

- Assume that $F(z)$, $\partial_z F(z)$ can be computed.
  $z(n)$ is obtained by binary search on z with $O(\log n)$ iterations.
- How to find $F(z)$ and $\partial_z F(z)$?
  - Rational systems can be explicitly solved;
  - Algebraic systems: Newton oracle;
  - Pòlya systems: approximated by algebraic;
  - Differential equations:
    sometimes can be solved fast, sometimes not.

Outline

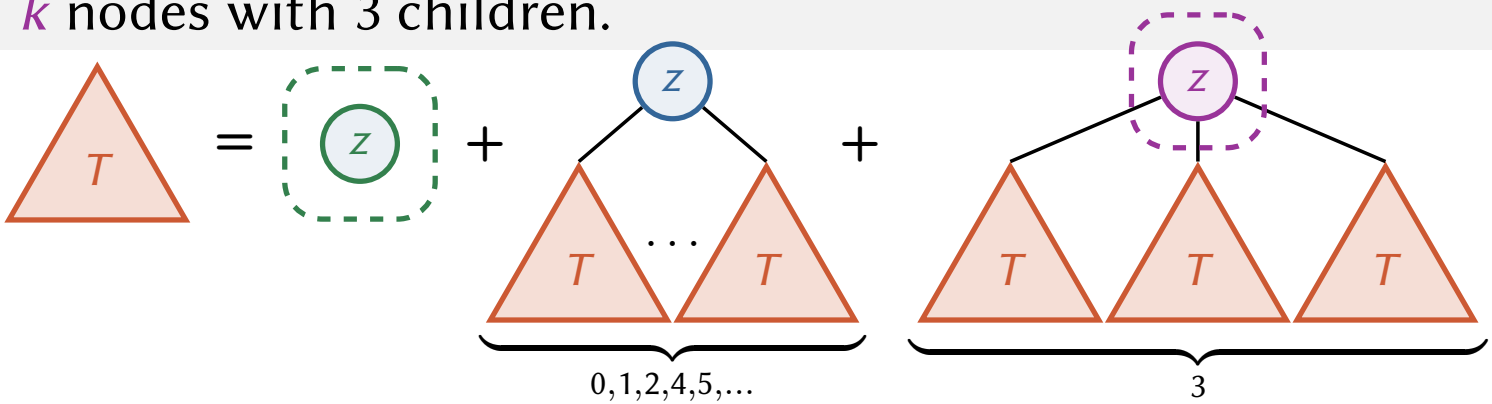
Concept of multiparametric sampling

Problem

Sample object of given size n and given parameters $m_1, m_2, \ldots, m_d$.

Example

Generate uniformly at random plane trees with n nodes, m leaves, and k nodes with 3 children.



$$T = zx + z\left(\frac{1}{1-T} - T^3\right) + zyT^3$$

Cost of multiparametric sampling

- Exact parameters $n_1, \ldots, n_d$ in $\Theta(n_1 n_2 \ldots n_d)$
  [Denise, Roques, Termier '00]
- Expected frequencies in $O(n)$, assuming tuning oracle provided.
  [Denise, Roques, Termier '00]
- Exact parameters $n_1, \ldots, n_d$ in $O(n^{d/2})$ using Boltzmann sampler and a tuning heuristic using local nonconvex optimisation [Bodini, Ponty '10]
- Exact parameters in $O(n^{3d/7} \log n)$ using mixture of Boltzmann sampling, novel tuning and recursive method
  [Bendkowski, Bodini, D. '18+]
- No hope to get polynomial(n, d) algorithm for exact multivariate sampling because otherwise $P = PH$.

Cost of multiparametric sampling

- Exact parameters $n_1, \ldots, n_d$ in $\Theta(n_1 n_2 \ldots n_d)$
  [Denise, Roques, Termier '00]
- Expected frequencies in $O(n)$, assuming tuning oracle provided.
  [Denise, Roques, Termier '00]
- Exact parameters $n_1, \ldots, n_d$ in $O(n^{d/2})$ using Boltzmann sampler and a tuning heuristic using local nonconvex optimisation [Bodini, Ponty '10]
- Exact parameters in $O(n^{3d/7} \log n)$ using mixture of Boltzmann sampling, novel tuning and recursive method
  [Bendkowski, Bodini, D. '18+]
- No hope to get polynomial(n, d) algorithm for exact multivariate sampling because otherwise $P = PH$.

Cost of multiparametric sampling

- Exact parameters $n_1, \ldots, n_d$ in $\Theta(n_1 n_2 \ldots n_d)$
  [Denise, Roques, Termier '00]
- Expected frequencies in $O(n)$, assuming tuning oracle provided.
  [Denise, Roques, Termier '00]
- Exact parameters $n_1, \ldots, n_d$ in $O(n^{d/2})$ using Boltzmann sampler and a tuning heuristic using local nonconvex optimisation [Bodini, Ponty '10]
- Exact parameters in $O(n^{3d/7} \log n)$ using mixture of Boltzmann sampling, novel tuning and recursive method
  [Bendkowski, Bodini, D. '18+]
- No hope to get polynomial(n, d) algorithm for exact multivariate sampling because otherwise $P = PH$.

Cost of multiparametric sampling

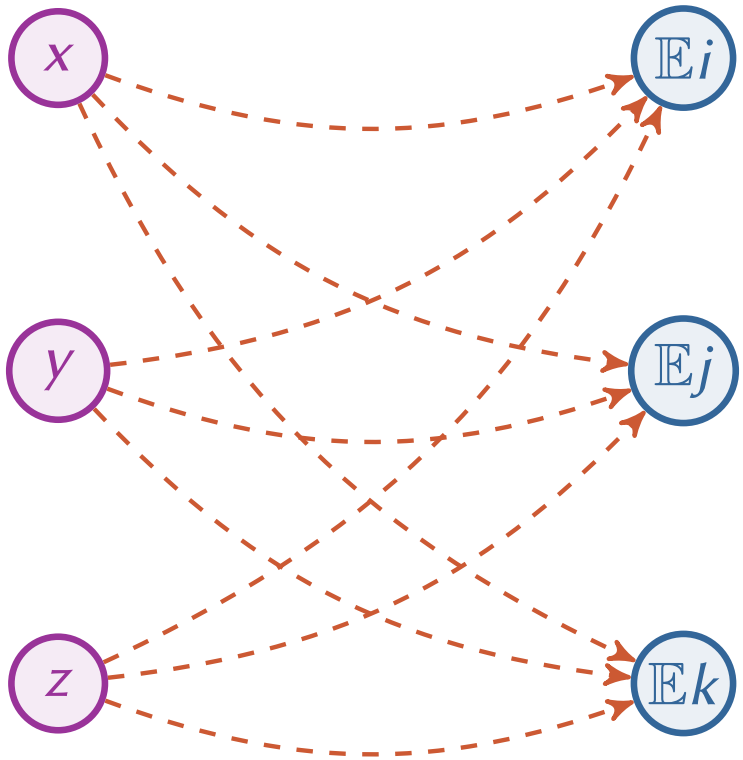
- Exact parameters $n_1, \ldots, n_d$ in $\Theta(n_1 n_2 \ldots n_d)$
  [Denise, Roques, Termier '00]
- Expected frequencies in $O(n)$, assuming tuning oracle provided.
  [Denise, Roques, Termier '00]
- Exact parameters $n_1, \ldots, n_d$ in $O(n^{d/2})$ using Boltzmann sampler and a tuning heuristic using local nonconvex optimisation [Bodini, Ponty '10]
- Exact parameters in $O(n^{3d/7} \log n)$ using mixture of Boltzmann sampling, novel tuning and recursive method
  [Bendkowski, Bodini, D. '18+]
- No hope to get polynomial(n, d) algorithm for exact multivariate sampling because otherwise $P = PH$.

Cost of multiparametric sampling

- Exact parameters $n_1, \ldots, n_d$ in $\Theta(n_1 n_2 \ldots n_d)$
  [Denise, Roques, Termier '00]
- Expected frequencies in $O(n)$, assuming tuning oracle provided.
  [Denise, Roques, Termier '00]
- Exact parameters $n_1, \ldots, n_d$ in $O(n^{d/2})$ using Boltzmann sampler and a tuning heuristic using local nonconvex optimisation [Bodini, Ponty '10]
- Exact parameters in $O(n^{3d/7} \log n)$ using mixture of Boltzmann sampling, novel tuning and recursive method
  [Bendkowski, Bodini, D. '18+]
- No hope to get polynomial(n, d) algorithm for exact multivariate sampling because otherwise $P = PH$.

Idea of multiparametric tuning

Handles $\Rightarrow$ Expectations



- Tuning is an inverse problem. How to solve it?

Cost of multiparametric tuning (recursive or Boltzmann)

$$n = \text{size}, \quad d = \#\text{ of parameters}$$

- Tuning in $O((\log n)^d)$ [folklore, nested binary searches]
- Asymptotic tuning when $n \to \infty$. Target point is expressed through alternative system involving singularities. (complexity not clear) [Bodini, Ponty '10]
- $O(Poly(d) \log\log n)$ if the initial argument is close enough to the target value. The search of initial value is done by dichotomy heuristic. [Bodini, Ponty '10]

Cost of multiparametric tuning (recursive or Boltzmann)

$$n = \text{size}, \quad d = \#\text{ of parameters}$$

- Tuning in $O((\log n)^d)$ [folklore, nested binary searches]
- Asymptotic tuning when $n \to \infty$. Target point is expressed through alternative system involving singularities. (complexity not clear) [Bodini, Ponty '10]
- $O(Poly(d) \log\log n)$ if the initial argument is close enough to the target value. The search of initial value is done by dichotomy heuristic. [Bodini, Ponty '10]

Cost of multiparametric tuning (recursive or Boltzmann)

$$n = \text{size}, \quad d = \#\text{ of parameters}$$

- Tuning in $O((\log n)^d)$ [folklore, nested binary searches]
- Asymptotic tuning when $n \to \infty$. Target point is expressed through alternative system involving singularities. (complexity not clear) [Bodini, Ponty '10]
- $O(Poly(d) \log\log n)$ if the initial argument is close enough to the target value. The search of initial value is done by dichotomy heuristic. [Bodini, Ponty '10]

Cost of multiparametric tuning (continued)

$n = \text{size}, \quad d = \#\text{ of parameters}$

- $O(Poly(d)\log n)$ for algebraic or rational systems + implementation on `github` [Bendkowski, Bodini, D. '17]
- $O(Poly(d) \times \{\text{oracle for } F(z,\vec{u}), \partial_z F(z,\vec{u}), \partial_{\vec{u}} F(z,\vec{u})\})$ for any generating function [Bendkowski, Bodini, D. '18+]

Cost of multiparametric tuning (continued)

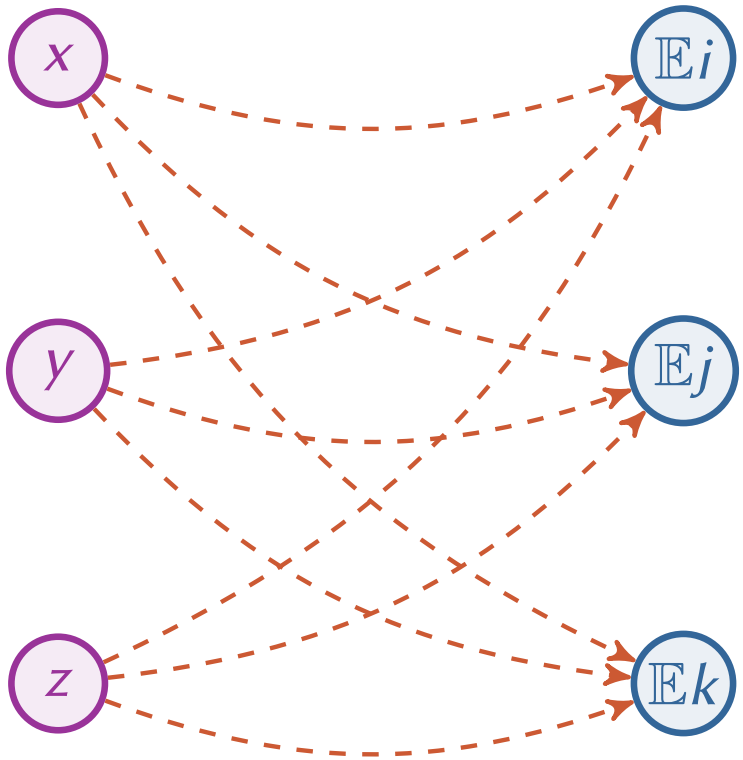
$n = \text{size}, \quad d = \#\text{ of parameters}$

- $O(Poly(d)\log n)$ for algebraic or rational systems + implementation on `github` [Bendkowski, Bodini, D. '17]
- $O(Poly(d) \times \{\text{oracle for } F(z,\vec{u}), \partial_z F(z,\vec{u}), \partial_{\vec{u}} F(z,\vec{u})\})$ for any generating function [Bendkowski, Bodini, D. '18+]

Outline

Idea of tuning

Handles $\Rightarrow$ Expectations



- Tuning is an inverse problem. How to solve it?

Convex optimisation approach idea

Given target expectations (n, j, k), the tuning problem is equivalent to

1. system of equations

$$x\frac{\partial_x A}{A} = n, \quad y\frac{\partial_y A}{A} = j, \quad z\frac{\partial_z A}{A} = k$$

2. Is equivalent to

$$\nabla_{x,y,z} \log A(e^x, e^y, e^z) = (n, j, k)$$

3. Is equivalent to

$$\nabla_{x,y,z} \left[\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k)\right] = 0$$

4. Is equivalent to a convex problem

$$\boxed{\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k) \to \min_{x,y,z}}$$

Convex optimisation approach idea

Given target expectations (n, j, k), the tuning problem is equivalent to

1. system of equations

$$x\frac{\partial_x A}{A} = n, \quad y\frac{\partial_y A}{A} = j, \quad z\frac{\partial_z A}{A} = k$$

2. Is equivalent to

$$\nabla_{x,y,z} \log A(e^x, e^y, e^z) = (n, j, k)$$

3. Is equivalent to

$$\nabla_{x,y,z} \left[\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k)\right] = 0$$

4. Is equivalent to a convex problem

$$\boxed{\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k) \to \min_{x,y,z}}$$

Convex optimisation approach idea

Given target expectations (n, j, k), the tuning problem is equivalent to

1. system of equations

$$x\frac{\partial_x A}{A} = n, \quad y\frac{\partial_y A}{A} = j, \quad z\frac{\partial_z A}{A} = k$$

2. Is equivalent to

$$\nabla_{x,y,z} \log A(e^x, e^y, e^z) = (n, j, k)$$

3. Is equivalent to

$$\nabla_{x,y,z} \left[\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k)\right] = 0$$

4. Is equivalent to a convex problem

$$\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k) \to \min_{x,y,z}$$

Convex optimisation approach idea

Given target expectations (n, j, k), the tuning problem is equivalent to

1. system of equations

$$x\frac{\partial_x A}{A} = n, \quad y\frac{\partial_y A}{A} = j, \quad z\frac{\partial_z A}{A} = k$$

2. Is equivalent to

$$\nabla_{x,y,z} \log A(e^x, e^y, e^z) = (n, j, k)$$

3. Is equivalent to

$$\nabla_{x,y,z} \left[\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k)\right] = 0$$

4. Is equivalent to a convex problem

$$\boxed{\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k) \to \min_{x,y,z}}$$

Interesting fact

$$\log A(e^x, e^y, e^z) - (x, y, z)^\top (n, j, k) \to \min_{x,y,z}$$

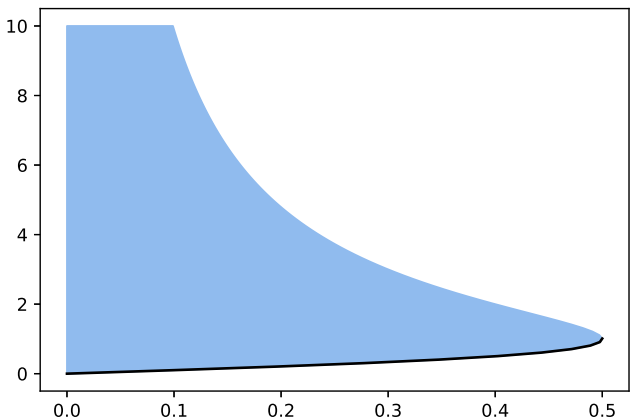
Interesting fact

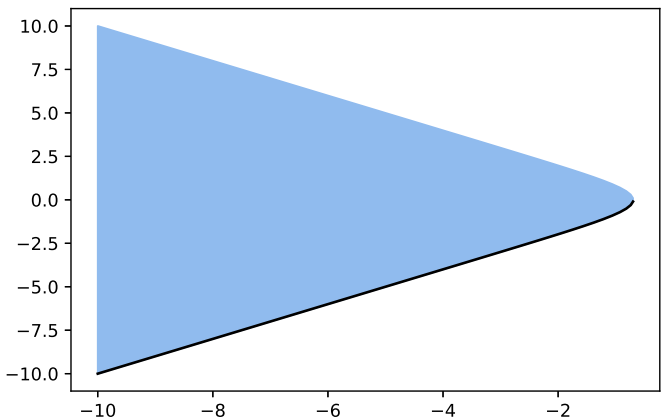
The convexity of the problem by itself doesn't give any guarantees that it can be solved efficiently.

Developing the algebraic case

Replace specification system by the system of inequalities

$$\begin{cases} z \to \max, \\ T \geq z + zT^2 \end{cases} \quad \Rightarrow \quad \begin{cases} \zeta \to \max, \\ b \geq \log(e^{\zeta} + e^{\zeta} e^{2b}) \end{cases}$$





Lemma

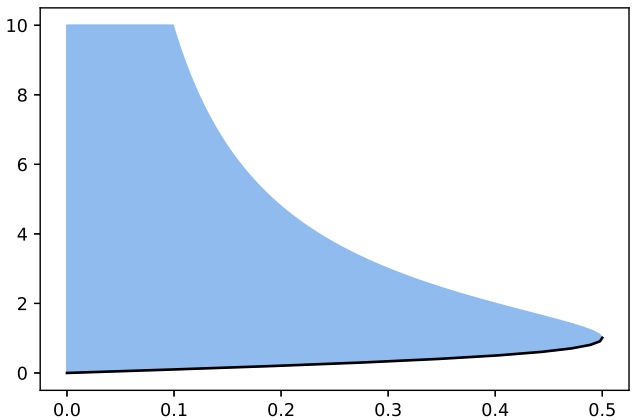
Log-exp transform will make the optimisation problem convex.

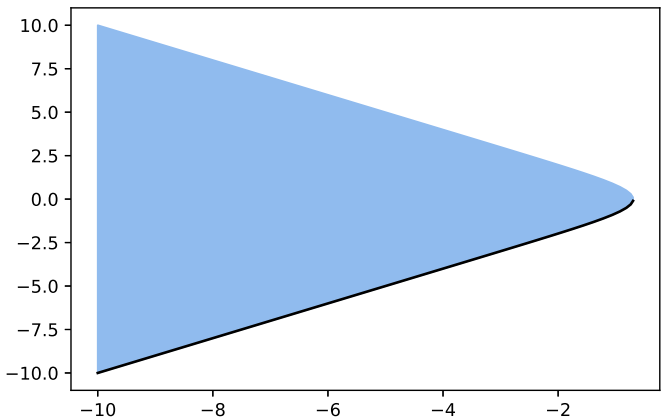
Developing the algebraic case

Replace specification system by the system of inequalities

$\vec{\mu}$ — vector of expected parameters (in class F_0),

$$\vec{F} = \vec{\Phi}(\vec{F}, \vec{z}) \quad \Rightarrow \quad \begin{cases} f_0 - \vec{\mu}^{\top}\vec{z} \to \min_{\vec{f},\vec{z}}, \\ \vec{f} \geq \log \vec{\Phi}(\exp(\vec{f}), \exp(\vec{z})) \end{cases}$$



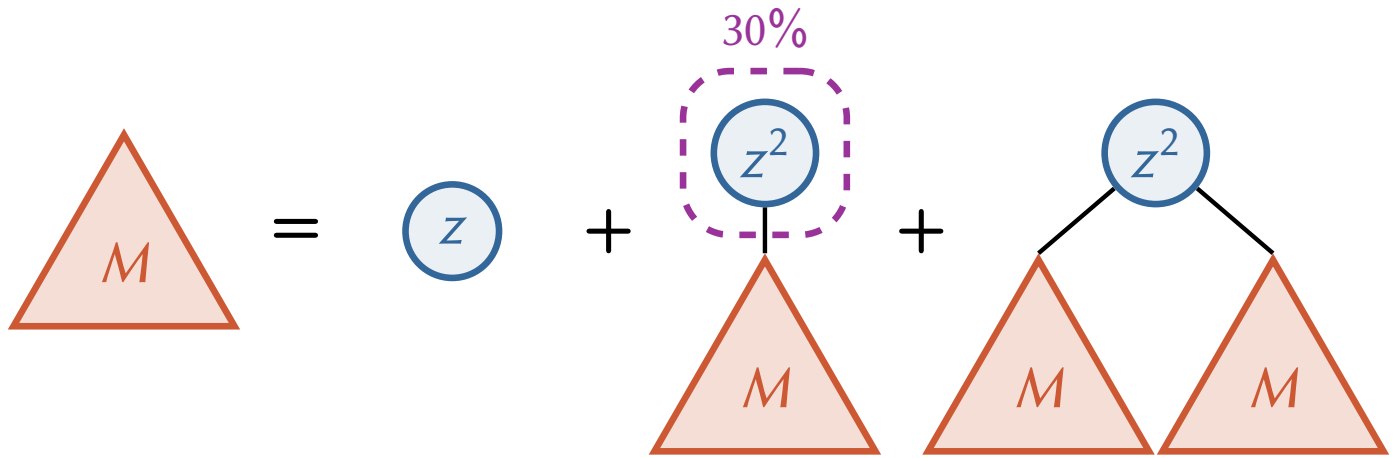


Bonus

Besides $\vec{z} = \vec{z}(\vec{\mu})$ we also obtain $\vec{F}(\vec{z})$.

Outline

Implementation. Boltzmann Brain



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

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

Generating Haskell module

1. Show me the contents of the input file

```
>cat motzkin.in
```

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

2. Generate a Haskell module

```
>bb motzkin.in > Motzkin.hs
```

3. Show me the code of the Haskell module

```
>vim Motzkin.hs
```

See next slide

Generating Haskell module

1. Show me the contents of the input file

```
>cat motzkin.in
```

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

2. Generate a Haskell module

```
>bb motzkin.in > Motzkin.hs
```

3. Show me the code of the Haskell module

```
>vim Motzkin.hs
```

See next slide

Generating Haskell module

1. Show me the contents of the input file

   ```
   >cat motzkin.in
   ```

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

2. Generate a Haskell module

   ```
   >bb motzkin.in > Motzkin.hs
   ```

3. Show me the code of the Haskell module

   ```
   >vim Motzkin.hs
   ```

   See next slide

Contents of Motzkin.hs

```haskell
-- | Compiler: Boltzmann brain v1.2
-- | Singularity: 0.523139341639112
-- | System type: algebraic
-- | System size: 1
-- | Constructors: 3
module Sampler
       (MotzkinTree(..), genRandomMotzkinTree, sampleMotzkinTree) where
import Control.Monad (guard)
import Control.Monad.Trans (lift)
import Control.Monad.Trans.Maybe (MaybeT(..), runMaybeT)
import Control.Monad.Random (RandomGen(..), Rand, getRandomR)

data MotzkinTree = Leaf
                 | Unary MotzkinTree
                 | Binary MotzkinTree MotzkinTree

randomP :: RandomGen g => MaybeT (Rand g) Double
randomP = lift (getRandomR (0, 1))

genRandomMotzkinTree ::
                       RandomGen g => Int -> MaybeT (Rand g) (MotzkinTree, Int)
genRandomMotzkinTree ub
  = do guard (ub > 0)
       p <- randomP
       if p < 0.37837770210556015 then return (Leaf, 1) else
         if p < 0.6216213625599037 then
           do (x0, w0) <- genRandomMotzkinTree (ub - 2)
              return (Unary x0, 2 + w0)
           else
           do (x0, w0) <- genRandomMotzkinTree (ub - 2)
              (x1, w1) <- genRandomMotzkinTree (ub - 2 - w0)
              return (Binary x0 x1, 2 + w1 + w0)

sampleMotzkinTree ::
                     RandomGen g => Int -> Int -> Rand g MotzkinTree
sampleMotzkinTree lb ub
  = do sample <- runMaybeT (genRandomMotzkinTree ub)
       case sample of
           Nothing -> sampleMotzkinTree lb ub
           Just (x, s) -> if lb <= s && s <= ub then return x else
                            sampleMotzkinTree lb ub
```

Sampling a combinatorial structure

Generate a random tree of size between 50 and 100

```
>ghci Motzkin.hs
*Sample> sampleMotzkinTree 50 100
```

```
Binary Leaf (Binary (Binary (Binary (Unary (Unary
(Unary (Binary (Unary Leaf) (Binary (Unary Leaf)
Leaf))))) (Unary (Unary (Binary (Binary (Unary
(Binary (Unary Leaf) Leaf)) (Unary (Binary (Binary
(Unary (Unary (Unary Leaf))) (Unary Leaf)) (Unary
Leaf)))) Leaf)))) (Unary (Unary (Binary (Binary
(Unary (Binary Leaf (Unary (Binary Leaf (Unary
(Binary Leaf Leaf)))))) Leaf) (Binary (Binary Leaf
Leaf) (Unary Leaf)))))) (Unary Leaf))
```

Code is available on github

- The core sampler and tuner
  (resp. Boltzmann Brain and Paganini)
  `github.com/maciej-bendkowski/boltzmann-brain`
- The examples (see next section):
  `github.com/maciej-bendkowski/multiparametric-combinatorial-samplers`

Outline

Random tilings

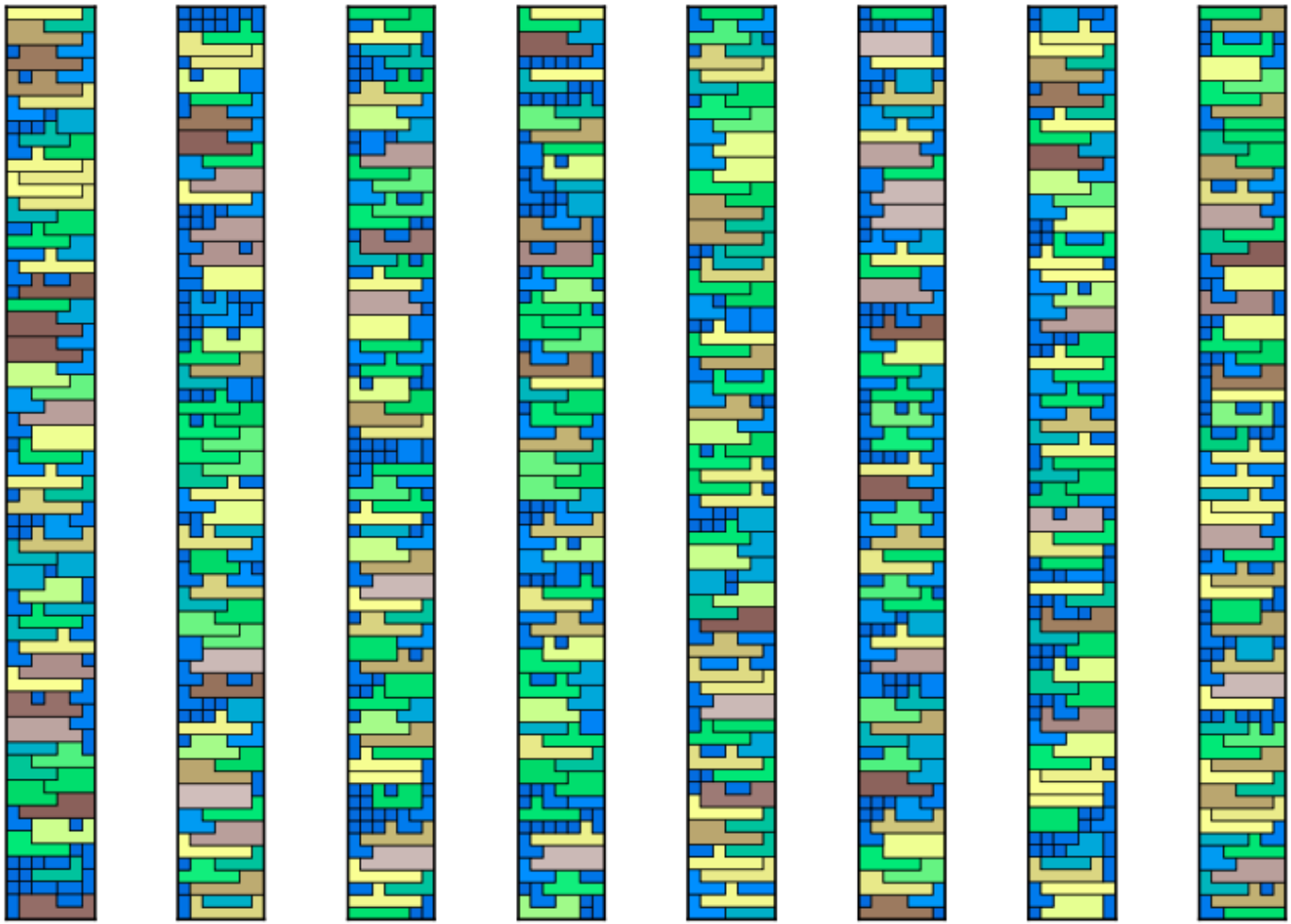
Problem

Tile a stripe $7 \times n$ with 126 different types of tiles such that the area covered by each tile is (approximately) uniform.

Tile construction principle

Attach a subset of unit squares to the base layer which is a single connected block.

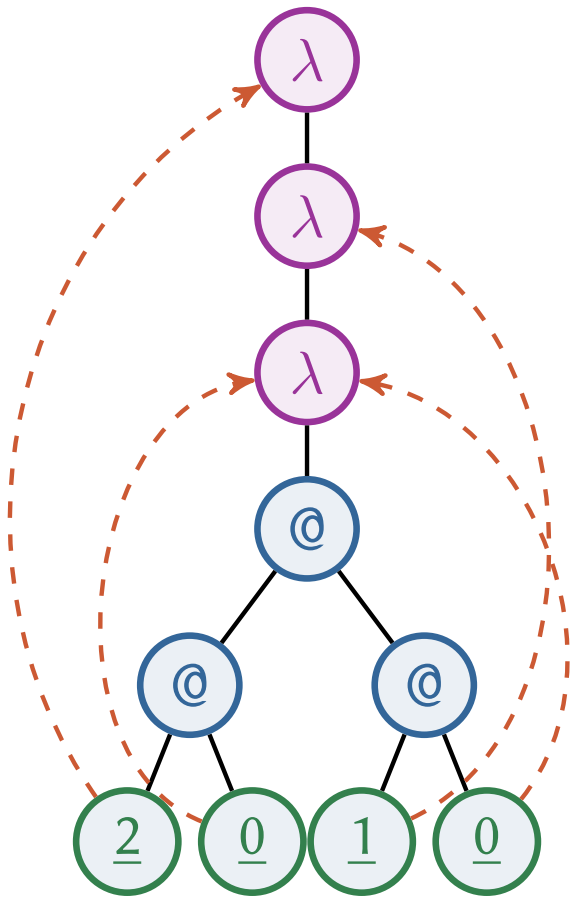
Generated tilings



Outline

Closed lambda terms for property testing

The specification can be approximated by algebraic



- Number of abstractions
- Number of variables
- De Bruijn index distribution
- Number of redexes
- Number of head abstractions
- Number of closed subterms

Application

Generate closed lambda terms (programs) with skewed distribution to find bugs in optimizing compilers.

Increasing trees with given degree profile

1. Maximum degree d is finite and fixed, but may be large.
   $\Omega(T; \vec{u}) = u_0 + u_1 T + u_2 T^2 + \ldots + u_d T^d$.

$$T' = z\Omega(T; \vec{u}), \quad \int_{T(0)}^{T(z)} \frac{d\tau}{\Omega(\tau; \vec{u})} = \frac{z^2}{2};$$

2. If $\vec{u}$ is fixed, the denominator is a polynomial with computable roots, hence the integral can be computed analytically.
3. Using binary search, $T(z; \vec{u})$ and $\partial_z T(z; \vec{u})$ can be computed.
4. $\nabla_{\vec{u}} T(z, \vec{u})$ can be also computed

$$\partial_z \nabla_{\vec{u}} T = z\partial_T \Omega(T, \vec{u}) \nabla_{\vec{u}} T + z\partial_{\vec{u}} \Omega(T, u)$$

Increasing trees with given degree profile

1. Maximum degree d is finite and fixed, but may be large. $\Omega(T; \vec{u}) = u_0 + u_1 T + u_2 T^2 + \ldots + u_d T^d$.

$$T' = z\Omega(T; \vec{u}), \quad \int_{T(0)}^{T(z)} \frac{d\tau}{\Omega(\tau; \vec{u})} = \frac{z^2}{2};$$

2. If $\vec{u}$ is fixed, the denominator is a polynomial with computable roots, hence the integral can be computed analytically.
3. Using binary search, $T(z; \vec{u})$ and $\partial_z T(z; \vec{u})$ can be computed.
4. $\nabla_{\vec{u}} T(z, \vec{u})$ can be also computed

$$\partial_z \nabla_{\vec{u}} T = z\partial_T \Omega(T, \vec{u}) \nabla_{\vec{u}} T + z\partial_{\vec{u}} \Omega(T, u)$$

Increasing trees with given degree profile

1. Maximum degree d is finite and fixed, but may be large.
   $\Omega(T; \vec{u}) = u_0 + u_1 T + u_2 T^2 + \ldots + u_d T^d$.

$$T' = z\Omega(T; \vec{u}), \quad \int_{T(0)}^{T(z)} \frac{d\tau}{\Omega(\tau; \vec{u})} = \frac{z^2}{2};$$

2. If $\vec{u}$ is fixed, the denominator is a polynomial with computable roots, hence the integral can be computed analytically.
3. Using binary search, $T(z; \vec{u})$ and $\partial_z T(z; \vec{u})$ can be computed.
4. $\nabla_{\vec{u}} T(z, \vec{u})$ can be also computed

$$\partial_z \nabla_{\vec{u}} T = z\partial_T \Omega(T, \vec{u}) \nabla_{\vec{u}} T + z\partial_{\vec{u}} \Omega(T, u)$$

Increasing trees with given degree profile

1. Maximum degree d is finite and fixed, but may be large. $\Omega(T; \vec{u}) = u_0 + u_1 T + u_2 T^2 + \ldots + u_d T^d$.

$$T' = z\Omega(T; \vec{u}), \quad \int_{T(0)}^{T(z)} \frac{d\tau}{\Omega(\tau; \vec{u})} = \frac{z^2}{2};$$

2. If $\vec{u}$ is fixed, the denominator is a polynomial with computable roots, hence the integral can be computed analytically.
3. Using binary search, $T(z; \vec{u})$ and $\partial_z T(z; \vec{u})$ can be computed.
4. $\nabla_{\vec{u}} T(z, \vec{u})$ can be also computed

$$\partial_z \nabla_{\vec{u}} T = z\partial_T \Omega(T, \vec{u}) \nabla_{\vec{u}} T + z\partial_{\vec{u}} \Omega(T, u)$$

Outline

Integer partitions

Example:

$$16 = 1 + 1 + 3 + 4 + 7$$

$$\mathcal{P} = \textsc{mset}(\textsc{mset}_{\geq 1}(\mathcal{Z}))$$

Generalisation from statistical physics (Bose–Einstein)

In d-dimensional anisotropic harmonic trap the number of states for particle with energy λ is $\binom{d+\lambda-1}{\lambda}$. Each state is represented as a multiset of λ elements having d different colours.

$$\mathcal{P} = \textsc{mset}(\textsc{mset}_{\geq 1}(d\mathcal{Z}))$$

d-dimensional quantum harmonic 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)
$\binom{d+\lambda-1}{\lambda}$	Number of particle states

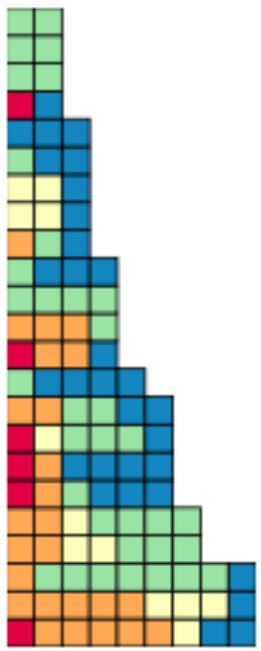
Problem

Generate random assemblies with given numbers of colours

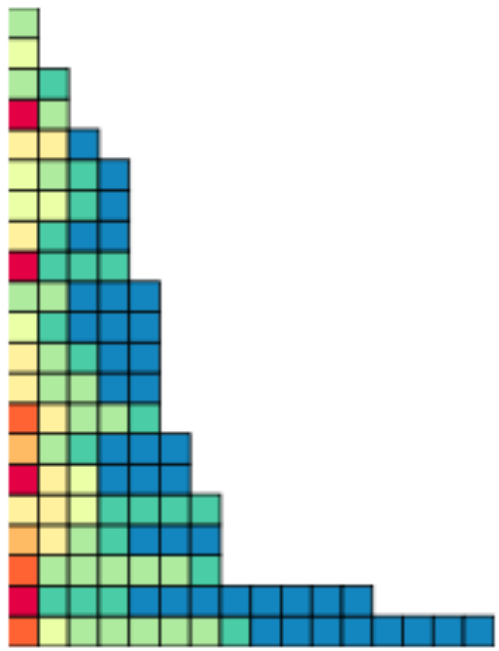
$$(n_1, n_2, \ldots, n_d)$$

Weighted integer partitions

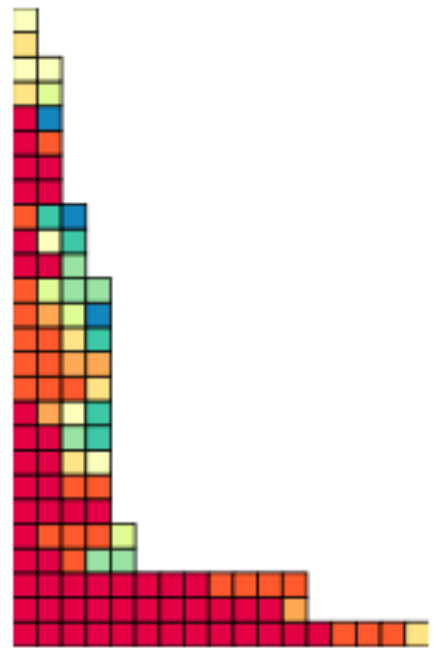
(small size objects for presentation purposes)



(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]

Conclusion

1. Polynomial efficient algorithm(s) for tuning recursive and Boltzmann samplers
2. Open source implementation: can be found by googling

   github boltzmann brain
3. Exact frequency sampler with smaller exponent[(in progress)]

That's all!

Thank you for your attention!