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 in Kraków
[4]Moscow Institute of Physics and Technology

Computational Logic and Applications,
Paris, May 25th 2018

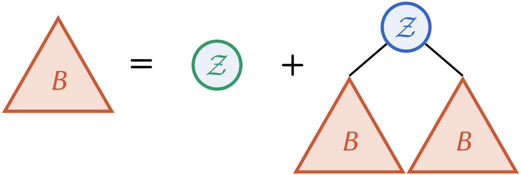
Outline

Combinatorial sampling techniques

Multivariate tuning

Applications

Warm-up — combinatorial samplers for binary trees



$$\boxed{B(z) = z + zB^2(z)}$$

<u>Recursive method</u>

exact-size sampling

(Nijenhuis and Wilf, 1978)

- if $n = 1$ return $\mathcal{Z}$,
- otherwise, sample k from probability distribution

$$\mathbb{P}(k) = \frac{b_k b_{n-k-1}}{b_n}$$

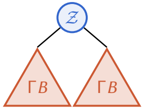
and recurse accordingly.

<u>Boltzmann samplers</u>

approximate-size sampling

(Duchon et al., 2004)

$$\Gamma B := \begin{cases} \mathcal{Z} & \mathbb{P}_{\mathcal{Z}} = \dfrac{z}{B(z)}, \end{cases}$$

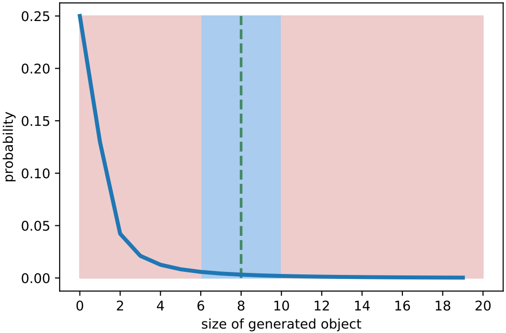


$$\mathbb{P}_{\Gamma B} = \frac{zB^2(z)}{B(z)}.$$

Univariate Boltzmann samplers — outcome distribution

- $\mathbb{P}(\Gamma B \text{ outputs tree } \tau) = \frac{z^{|\tau|}}{B(z)}$ (uniform conditioned on size);
- Expected output size is $z\frac{B'(z)}{B(z)}$ (increasing function of z).

How long do we wait for an object of size in $[n(1-\varepsilon), n(1+\varepsilon)]$?



$O_\varepsilon(n)$ for binary trees.

Combinatorial samplers — summary

For general, admissible combinatorial specifications, sampling objects of size n takes, respectively:

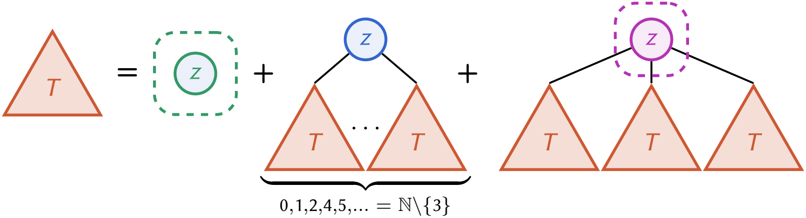
- $O(n^2)$ using the recursive method;
- $O(n^2)$ using exact-size Boltzmann sampling;
- $O(n)$ using approximate-size Boltzmann sampling.

Multiparametric sampling — example

Problem

Generate uniformly at random plane trees with

- n nodes in total,
- j leaves in total,
- and k nodes with 3 children.



Multiparametric sampling — state of the art

Problem

Given an admissible, algebraic[1] combinatorial specification $\mathcal{S}$ and an expectation vector X controlling k parameters of associated objects, generate a uniformly random object with expectation profile X.

Status (exact-parameter case):

- Dynamic programming using the recursive method $O(n^{k+1})$ (essentially Nijenhuis and Wilf, 1978);
- Boltzmann sampling using multidimensional oracles $O(n^{1+k/2})$ (Bodini and Ponty, 2010);

Our contributions:

- (target parameter expectations) $n \cdot \mathrm{Poly}(k)$;
- (exact-size, approximate-size parameters for strongly-connected rational languages) $n + O(1)$.

[1] extensions are also available

Outline

Combinatorial sampling techniques

Multivariate tuning

Applications

Multivariate Boltzmann model

Probability model

Let $C(\mathbf{z}) = \sum_{\mathbf{p} \geq \mathbf{0}} c_{\mathbf{p}} \mathbf{z}^{\mathbf{p}}$ be a multivariate generating function associated with the class $\boldsymbol{C}$. Then, the probability $\mathbb{P}_{\mathbf{z}}(\omega)$ of generating object ω of (composite-)size $\boldsymbol{p}$ takes the form

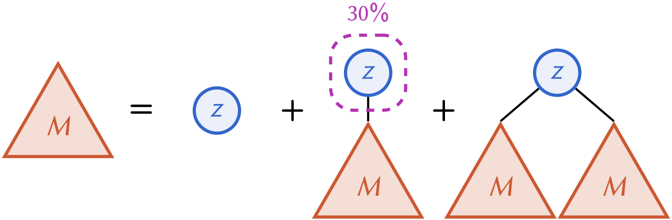
$$\mathbb{P}_{\mathbf{z}}(\omega) = \frac{\mathbf{z}^{\mathbf{p}}}{C(\mathbf{z})} = \frac{z_1^{p_1} \cdots z_k^{p_k}}{C(z_1, \ldots, z_k)}.$$

In consequence, the expectation of $\mathbb{E}_{\boldsymbol{z}}(N_i)$ of parameter z_i takes the form

$$\mathbb{E}_{\boldsymbol{z}}(N_i) = z_i \frac{\dfrac{\partial}{\partial z_i} C(\boldsymbol{z})}{C(\boldsymbol{z})} \quad .$$

Multivariate Boltzmann model — example

Q: How to quickly sample Motzkin trees, uniformly at random conditioned on size, with roughly 30% of unary nodes?



$$M(z,u) = z + uzM(z,u) + zM(z,u)^2$$

$$\mathbb{E}_{z,u}(N_u) = u\frac{\frac{\partial}{\partial u}M(z,u)}{M(z,u)} \quad .$$

Tuning as an optimisation problem

Let $\nabla_{\boldsymbol{z}} f(\boldsymbol{z}) = \left(\frac{\partial}{\partial z_1} f(\boldsymbol{z}), \ldots, \frac{\partial}{\partial z_k} f(\boldsymbol{z}) \right)^\top$ denote the gradient vector with respect to $\boldsymbol{z} = (z_1, \ldots, z_k)$. Then, solving

$$\mathbb{E}_{\boldsymbol{z}}(\boldsymbol{N}) = \boldsymbol{p}$$

is equivalent to

$$\nabla_{\boldsymbol{z}} \left[\log C(\boldsymbol{e}^{\boldsymbol{z}}) - \boldsymbol{z}^\top \boldsymbol{p} \right] = \boldsymbol{0}.$$

Theorem (B., Bodini, Dovgal)

Fix the expectations $\mathbb{E}_{\boldsymbol{z}}(\boldsymbol{N}) = \boldsymbol{p}$. Then, the associated tuning vector $\boldsymbol{z}$ is equal to $\boldsymbol{e}^{\boldsymbol{\xi}}$ where $\boldsymbol{\xi}$ comes from the following minimisation problem:

$$\log C(\boldsymbol{e}^{\boldsymbol{\xi}}) - \boldsymbol{p}^\top \boldsymbol{\xi} \to \min_{\boldsymbol{\xi}} \quad .$$

Log-exp transform for algebraic systems

Theorem (B., Bodini, Dovgal)

Let $\boldsymbol{\mathcal{C}} = \boldsymbol{\Phi}(\boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{Z}})$ be an algebraic system. Fix the expectations $\mathbb{E}_{\boldsymbol{z}}(\boldsymbol{N}) = \boldsymbol{p}$ of objects sampled from $\mathcal{C}_1$. Then, the corresponding tuning problem is equivalent to the following log-exp transformed convex minimisation problem:

$$\begin{cases} c_1 - \boldsymbol{p}^\top \boldsymbol{\xi} \to \min_{\boldsymbol{\xi}, \boldsymbol{c}} \quad , \\ \log \boldsymbol{\Phi}(e^{\boldsymbol{c}}, e^{\boldsymbol{\xi}}) - \boldsymbol{c} \leq 0. \end{cases}$$

Log-exp transform — example

$$\begin{cases} A \geq zxB^2 + zyC \\ B \geq zxB + zA \\ C \geq z + zA \end{cases} \Rightarrow \begin{cases} \alpha \geq e^\zeta e^\xi e^{2\beta} + e^\zeta e^\eta e^\gamma \\ \beta \geq e^\zeta e^\xi e^\beta + e^\zeta e^\alpha \\ \gamma \geq e^\zeta + e^\zeta e^\alpha \end{cases}$$

Associated convex minimisation problem:

$$\boxed{\begin{cases} \alpha - \mathbb{E}_z\zeta - \mathbb{E}_x\xi - \mathbb{E}_y\eta \to \min_{\zeta,\xi,\eta,\alpha} \\ \alpha \geq \log\left(e^\zeta e^\xi e^{2\beta} + e^\zeta e^\eta e^\gamma\right) \\ \beta \geq \log\left(e^\zeta e^\xi e^\beta + e^\zeta e^\alpha\right) \\ \gamma \geq \log\left(e^\zeta + e^\zeta e^\alpha\right) \end{cases}}$$

Convex optimisation complexity

Theorem (Nesterov and Nemirovskii, 1994)

Convex optimisation programs can be effectively optimised using polynomial-time interior-point methods. As a consequence, for multiparametric combinatorial systems with description length L, the tuning problem can be solved with precision ε in time

$$O\left(L^{3.5}\log\frac{1}{\varepsilon}\right) \quad .$$

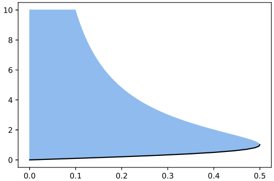
Singular samplers for algebraic systems

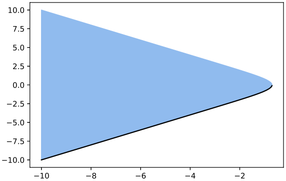
Theorem (B., Bodini, Dovgal)

Let $\boldsymbol{\mathcal{C}} = \boldsymbol{\Phi}(\boldsymbol{\mathcal{C}}, \mathcal{Z}, \boldsymbol{\mathcal{U}})$ be an algebraic system with $\mathcal{Z}$ marking the object size. Fix the expectations $\boldsymbol{p}$ of atoms $\boldsymbol{\mathcal{U}}$. Set $z = e^{\xi}$ and $\boldsymbol{u} = e^{\boldsymbol{\eta}}$. Then, $(z, \boldsymbol{u})$ from the following convex program tune the corresponding singular Boltzmann sampler:

$$\begin{cases} \xi + \boldsymbol{p}^{\top}\boldsymbol{\eta} \to \max_{\xi,\boldsymbol{\eta},\boldsymbol{c}} \\ \log \boldsymbol{\Phi}(e^{\boldsymbol{c}}, e^{\xi}, e^{\boldsymbol{\eta}}) - \boldsymbol{c} \leq 0\,. \end{cases}$$

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





Outline

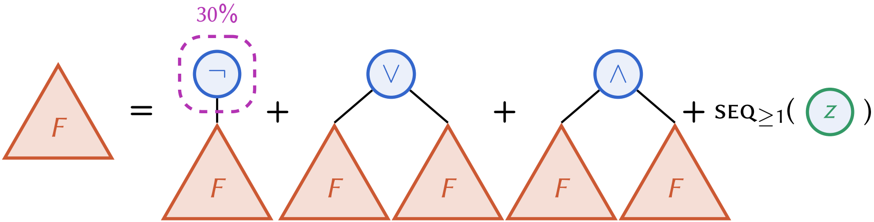
Combinatorial sampling techniques

Multivariate tuning

Applications

Open source implementation — Boltzmann Brain

Example – boolean formulae over $\{\neg, \vee, \wedge\}$:



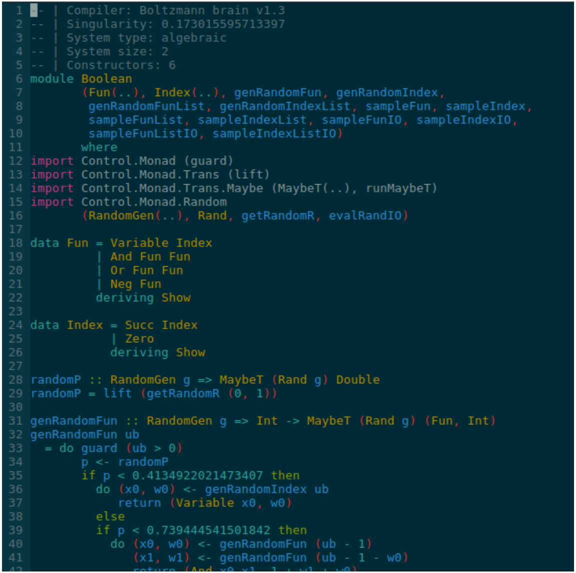
$$F(z) = uzF(z) + 2zF(z)^2 + \frac{z}{1-z}$$

```
Fun = Neg Fun [0.3]
    | And Fun Fun
    | Or Fun Fun
    | Variable Index (0)

Index = Zero
      | Succ Index
```

Code available at
https://github.com/maciej-bendkowski/boltzmann-brain
https://github.com/maciej-bendkowski/multiparametric-combinatorial-samplers

Boltzmann Brain — example output sampler



```haskell
-- | Compiler: Boltzmann brain v1.3
-- | Singularity: 0.173015595713397
-- | System type: algebraic
-- | System size: 2
-- | Constructors: 6
module Boolean
       (Fun(..), Index(..), genRandomFun, genRandomIndex,
        genRandomFunList, genRandomIndexList, sampleFun, sampleIndex,
        sampleFunList, sampleIndexList, sampleFunIO, sampleIndexIO,
        sampleFunListIO, sampleIndexListIO)
       where
import Control.Monad (guard)
import Control.Monad.Trans (lift)
import Control.Monad.Trans.Maybe (MaybeT(..), runMaybeT)
import Control.Monad.Random
       (RandomGen(..), Rand, getRandomR, evalRandIO)

data Fun = Variable Index
         | And Fun Fun
         | Or Fun Fun
         | Neg Fun
         deriving Show

data Index = Succ Index
           | Zero
           deriving Show

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

genRandomFun :: RandomGen g => Int -> MaybeT (Rand g) (Fun, Int)
genRandomFun ub
  = do guard (ub > 0)
       p <- randomP
       if p < 0.4134922021473407 then
         do (x0, w0) <- genRandomIndex ub
            return (Variable x0, w0)
         else
         if p < 0.739444541501842 then
           do (x0, w0) <- genRandomFun (ub - 1)
              (x1, w1) <- genRandomFun (ub - 1 - w0)
              return (And x0 x1, 1 + w1 + w0)
```

Toy example — random tilings

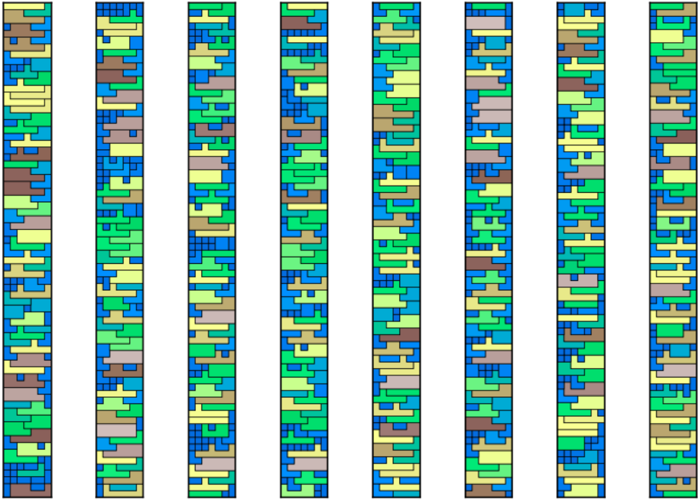
Problem

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

Note: The associated automaton has

- over $2,000$ states, and
- over $28,000$ transitions.

Random tilings — example



Simply-generated trees with node degree constraints

Simple varieties of plane trees with fixed sets of admissible node degrees, satisfying the general equation

$$y(z) = z\phi(y(z)) \quad \text{for some polynomial} \quad \phi \colon \mathbb{C} \to \mathbb{C}\,.$$

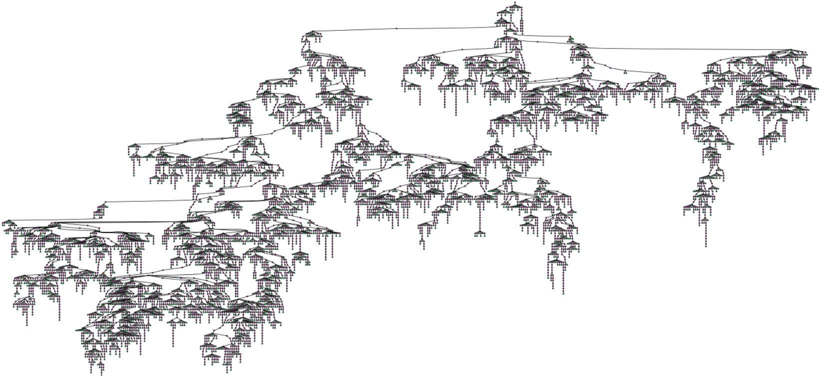
For instance, consider plane trees
$\phi(y(z)) = a_0 + a_1 y(z) + a_2 y(z)^2 + \cdots + a_9 y(z)^9$.

We tune the corresponding algebraic specification so to achieve a target frequency of 1% for all nodes of degrees $d \geq 2$. Frequencies of nodes with degrees $d \leq 1$ are left undistorted.

Node degree	0	1	2	3	4	5	6	7	8	9
Tuned frequency	- - -	- - -	1.00%	1.00%	1.00%	1.00%	1.00%	1.00%	1.00%	1.00%
Observed frequency	35.925%	56.168%	0.928%	0.898%	1.098%	0.818%	1.247%	0.938%	1.058%	0.918%
Default frequency	50.004%	24.952%	12.356%	6.322%	2.882%	1.984%	0.877%	0.378%	0.169%	0.069%

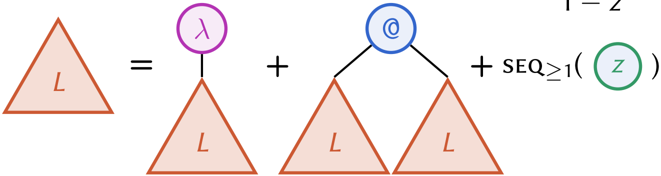
Table: Empirical frequencies of the node degree distribution.

Simply-generated trees with node degree constraints – example



λ-terms with given de Bruijn index distribution

$$L = zL + zL^2 + u_1 z + u_2 z^2 + \ldots + u_8 z^8 + \frac{z^9}{1 - z}$$



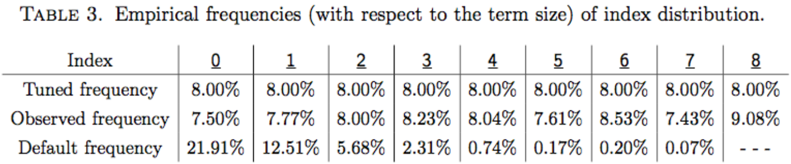
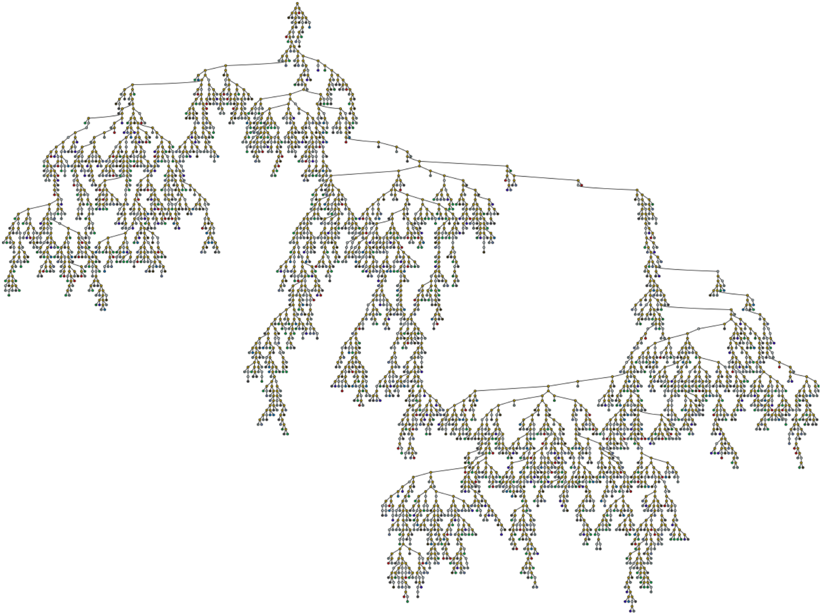


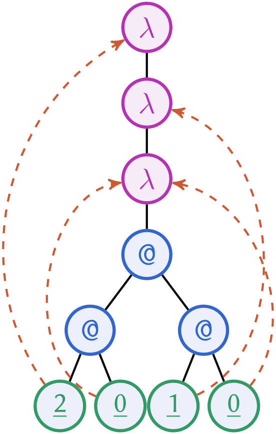
TABLE 3. Empirical frequencies (with respect to the term size) of index distribution.

Index	$\underline{0}$	$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$	$\underline{5}$	$\underline{6}$	$\underline{7}$	$\underline{8}$
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%	- - -

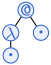
λ-terms with given de Bruijn index distribution – example



Closed λ-terms for property-based software testing



Some interesting parameters:

- number of variables,
- de Bruijn index distribution,
- Number of head lambdas,
- Number of redexes, i.e. 

- . . .

Motivation

Generate closed lambda terms (programs) with skewed distribution to find bugs in functional language compilers such as Haskell's GHC.

Integer partitions and the Bose–Einstein condensate

Example:

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

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

Generalisation from statistical physics (Bose–Einstein)

In the model of d-dimensional 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.

$$\mathcal{E} = \textsc{mset}\big(\textsc{mset}_{\geq 1}(\mathcal{Z}_1 + \mathcal{Z}_2 + \cdots + \mathcal{Z}_d)\big)$$

d-dimensional quantum harmonic oscillator

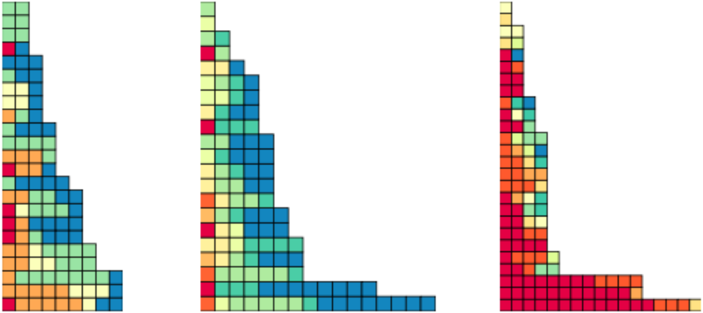
Weighted partition	Random particle assembly
Sum of numbers	Total energy
Number of colours	Dimension (d)
Row of Young table	Particle
Number of rows	Number of particles
Number of squares in the row	Energy of a particle (λ)

Problem

Generate random assemblies with given (expected) numbers $(n_1, n_2, \ldots, n_d)$ of colours (dimension).

Weighted integer partitions

(5,8, and 9 colours, respectively)



(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]

Thank you!

Thank you for your attention!