For $G = S_n$ for $n=3$ & $X = \{1,2,3\}$, then $\exists$ $3!$ permutations of it:

$$\{1,2,3\}, \{2,1,3\}, \{2,3,1\}, \{3,2,1\}$$
$$\{1,3,2\}, \{3,1,2\}$$

Call Comments

- <u>Note</u>: A group action that is free (only $g \in G$ that leaves an element $x \in X$ fixed st $g(x) = x$ is $g = e$) & transitive ($\exists x, y \in X, g \in G$ st $g \cdot x = y$ or $g(x) = y$) means the existence of only <u>one</u> orbit

- For degeneracy, instead of working w/ things that make E invariant (which is the identity), look at <u>degeneracy in vertex space</u> st $G' = G/\mathcal{G}$
  - ↳ Use of $G/\mathcal{G}$ should delete some columns in uniformity plot (remove degenerate multiplicity)
  - ↳ If all $\underbrace{\deg n_i = 1}_{\text{no degeneracy}}$, this reduces to $G' = S_n$

✓ + uniformity — & plot + uniformity

>> Make permutations in code general ✓, general, $\underbrace{G/\mathcal{G}}_{\text{Mathematica?}}$, show removal of degeneracy

—— x —— Removing Degeneracy

Consider the group action G on the set of nodes $\mathcal{N}$ which gives an equivalence class of orbits:

$$\mathcal{N}/G = \{ \underbrace{\mathrm{orb}(n)}_{[n]} : n \in \mathcal{N} \} = \{ \{ g(n) : g \in G \} : n \in \mathcal{N} \}$$

This partitions $\mathcal{N}$ into seperate pieces of possible values under G. Here two elements $n, m \in \mathcal{N}$ are equivalent $n \sim m$ iff $\mathrm{orb}(n) = \mathrm{orb}(m)$ [which implies $[n] = [m] \Rightarrow n \sim m$].

Now, there are actions of G which leaves the edge set E invariant which is precisely the identity $e \in G$

↳ Since we don't precisely know how this affects E, we consider actions of G that leaves $\mathcal{N}$ invariant ($g(n) = n$). In our case this would involve swapping nodes of equal degree

Consider $\mathcal{G} \subset G$ st $\forall h \in \mathcal{G}$ & $\forall n \in \mathcal{N}$, $h(n) = n$. We call $\mathcal{G}$ the isotropy subgroup. For a given $n \in \mathcal{N}$ we have:

or stabilizer subgroup

$$\mathcal{G}_n = \{ g \in G : g(n) = n \}$$

One could consider collecting all these subgroups as $\mathcal{G} = \bigcup_n \mathcal{G}_n$ & construct a fibre bundle $\mathcal{G} \to \mathcal{N}$, where a fibre of the base space is precisely $\mathcal{G}_n$

↳ Thus to remove degeneracy in our group action, we consider the reduced group as:

$$G' = G/\mathcal{G}$$ } Removes group elements which act trivially on elements of $\mathcal{N}$

This is the set of equivalence classes of G where two elements $g, g' \in G$ are equivalent if they differ by an element $h \in \mathcal{G}$:

$$g \sim g' = h^{-1} g h$$

Thus we need only modify our transition set to:

$$\mathrm{orb}(n) \cap \mathcal{N} = \{ g(n) : g \in G/\mathcal{G} \} \cap \mathcal{N}$$

When it comes to implementation in Python (as opposed to Mathematica / Sage) is that there is no well-defined way to efficiently define $G/\mathcal{G}$

↳ Instead at each step of the shuffle, we compute the stabilizer subgroup $\mathcal{G}_n$ & select a group action that does not use a stabilizer group element.

A rough first implementation is selecting from a new transition set:

$$\mathrm{orb}_G(n) \cap \mathcal{N} \cancel{\cap}\, \underbrace{\mathrm{orb}_{\mathcal{G}_n}(n)}_{\text{orbit wrt stabilizer group } \mathcal{G}_n}$$

Alternatively, we can write this using set minus notation:

$$\left(\mathrm{orb}_G(n) \setminus \mathrm{orb}_{\mathcal{G}_n}(n)\right) \cap \mathcal{N}$$

=> We must update our algorithm to exclude elements of $\mathrm{orb}_{\mathcal{G}_n}(n)$ which trivially permutes elements to the same position.

Note: → Does not work for permutation group where in the first step of the shuffle $\mathrm{orb}_G(n) \setminus \mathrm{orb}_{\mathcal{G}_n}(n) = \emptyset$

Comments

- Sym ⊃ Perm, Cyclic
  ↗ all elements change
  ↳ Reduced orbits: disallowed transition
- Why probability doesn't agree w/ pink ✓
  ↳ Forgot to include ✓
- Consensus shuffle no pre-built Python package ⟶ tests
  ↗ Consensus bundle: $\mathrm{Sym}(\mathcal{N})/\mathcal{G} \longrightarrow \mathcal{N}$

⋮--> $j \longleftarrow$ Random (t) } Using secure sense of randomness (global key)

---> Random function (NOT np.random.randint) (select a random element of transition set t)

- ↳ Have n-bits of pseudo-random seed from secure random source

↳ n-bits maps to 2^n possible outcomes. → element of t } If $|t| = 2^n$, trivially n-bits maps to all 2^n possibilities (n-bits directly index choice of t)

↓

global key eq 3.8 } Gives ∞ long bit string

---> For now we use generic random (guarantees random bit)

$\boxed{x = \text{bitstring} \quad n = |x|}$

⟶ If $|t| < 2^n$, what happens then?

↳ If $x > |t|$ (out of bounds) } Discard n-bits. Take new n-bits from bit stream (no wrap around as it destroys uniformity)

↳ Check in bounds

↳ Repeat

select from top of stack (global key)

- → Furthermore $n = \lceil \log_2 |t| \rceil$ } Minimum number of bits to address all elements of t (index)

. If $x \leq |t|$, then x is the index of t.

Repeat until success

⟶ Before: Random = numpy random

⟶ Now: Random = f(stuff; $\underbrace{\text{numpy random}}_{\text{For random bit}}$) } + Signal Screenshot

- Implementation: Swift language