Iz Tecúm

Hi! My name is Iz, and I am a student at Columbia University pursuing a degree in (pure) mathematics. I am from rural Kansas, but I have spent my previous summer as a pure probability researcher with my school's mathematics department here in New York. I learned a lot during these last few months with HeadStart and look forward to meet with you!

Let's connect!
https://www.linkedin.com/in/israel-tramos/

Overview

What? A python simulator for a Markov chain (Glauber dynamics) on colorings considering a 4-vertex cycle. Each step repaints one vertex while keeping the colors proper, traction over iteratively many independent trials.

How long does it take for this random process to "forget" its starting state and look uniform? First, assume worst-case variation to the uniform distribution against time for coloration numbers q.

Why? I started this in the summer as a way to connect theory with programming. It was my first full pipeline from probabilistic modeling to an actual simulation and taught me how Markov chains behave in practice. Not perfect, but certainly a massive undertaking I am proud of having taken.

Demo

Comparing Classical and Path Coupling Bounds Finite-State Markov Chains: A Graph-Coloring Case Study

Israel Chigüil Tecúm-Ramos
Department of Mathematics
Columbia University
New York, NY 10027 USA
ilt2109@columbia.edu

B. Results

Figure 1 plots the worst-start total-variation distance $\max_{x\in\Omega} \|P^t(x,\cdot)-\pi\|_{TV}$ estimated from 10^3 trajectories per start state. The dashed line marks $\varepsilon = 0.05$.

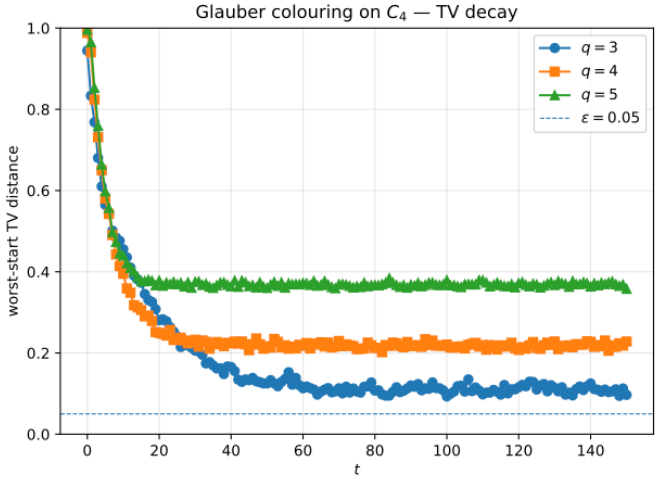

Fig. 1: Empirical TV decay for Glauber coloring on C_4. Larger q mixes more slowly, consistent with the contraction factor $\alpha(q) = 1 - \frac{1}{2(q-1)}$.

Listing 1: 20-line Python script that recomputes Fig. 1.

```python
# tv_minimal.py : exact worst-start TV on C4, q = 3
import itertools, numpy as np
V = 4
def proper(col):
    return all(col[i] != col[(i+1) % V] for i in
        range(V))

states = [c for c in itertools.product(range(3),
    repeat=V) if proper(c)]
idx    = {c: i for i, c in enumerate(states)}
n      = len(states)

# transition matrix
P = np.zeros((n, n))
for c in states:
    i = idx[c]
    for v in range(V):
        bad   = {c[(v-1) % V], c[(v+1) % V]}
        legal = [x for x in range(3) if x not in bad
            ]
        for x in legal:
            p = 1 / 4 / len(legal)
            d = list(c); d[v] = x
            P[i, idx[tuple(d)]] += p

pi = np.full(n, 1 / n)
T  = 150
tv = np.zeros(T + 1)
for c in states:
    mu = np.zeros(n); mu[idx[c]] = 1
    for t in range(T + 1):
        tv[t] = max(tv[t], 0.5 * np.abs(mu - pi).sum
            ())
        mu    = mu @ P
print(tv)                    # numeric check
```

Reflection

Biggest challenge? The biggest challenge was learning how to program in the first place. At the start of the summer, I knew basically zero programming and came from a very anti-CS background. To even get to this project, I had to self-teach the equivalent of an intro CS/data structures course. I was very much forced out of my comfort zone. The most rewarding part was being recognized for my work, and knowing that I am capable of forcing brutal study weeks when nobody is watching.

If you had more time, what would you add? If I had more time, I'd experiment aggressively with different simulation setups and methods. Right now I use a straightforward MCMC approach; given extra time I'd update rules, graph sizes, and maybe compare using more novel techniques.