Inf 674: Kleinberg's Grid

Nidhi Hegde Fabien Mathieu

February 1, 2016

The goal of this mini-project is quite straightforward: verify John Kleinberg's theorem on small-world navigability, from *The Small-World Phenomenon: An Algorithmic Perspective*.

G denotes a $n \times n$ flat grid, where each node has up to 4 regular neighbors (West, East, North, South). We use the Manhattan distance ($d(a,b) = |a_x - b_x| + |a_y - b_y|$). Any node a has a special "shortcut" picked up at random among the other nodes of G such that the probability that $b \neq a$ is chosen is proportional to $(d(a,b))^{-r}$ for some $r > 0$.

The routing algorithm works as follows: we pick up a start and a final nodes i.i.d. at random on the grid. From the starting node, at each step, one moves from the current node to (one of) the neighbor(s) that is the closest to the destination (for the Manhattan distance), until the arrival is reached. We are interested in the average routing time, i.e. the average number of steps required to reach destination.

Kleinberg's result: if $r = 2$, then the routing time is short ($O(\log^2(n))$), while if $r \neq 2$, the routing time is long ($O(n^\alpha)$ for some $\alpha(r) > 0$).

This result is often interpreted as follows: shortcuts can turn graphs of large diameter into navigable small-worlds, but this works only if the shortcuts follow a very precise distribution.

Your mission: try and validate this result through simulations.

For this assignment, I expect a report and a working program (preferably in Python 3.x, but other languages may be used if "Instructions for dummies" are provided along with some justification for this choice). You will NOT be judged on your raw programming skills. Discuss your methodology, why and how you designed your experiments (choice of the parameters, ...), keep the complexity of your simulator to a minimum, interpret your results, draw plots and/or figures... If you made a "good" mistake that helped you get a better grasp of the problem, use it instead of hiding that under the carpet.

Remarks and hints

- To compare with, you can remind the average path length between start and arrival in absence of shortcuts.
- One of the issue with Kleinberg's grid is that the shortcut distribution depends on the starting node. If that frightens you, you can approximate the grid by a torus, which will give you a unique shift distribution. I did try that. Sadly, what you will discover there will probably draw you back to the grid.
- You can draw shortcuts on the fly as you route instead of pre-computing them.
- 10-100 trials are good when you are in front of your computer, to get rough estimates for debugging and seeking directions. 10 000 trials are slow and better executed in background, but it gives precise results ("1%" error margin).
- After Snowzilla, it is probably best to use "polar" coordinates in New-York. Why? 4 *i* say so!
- *Tant que je perds, je joue.* ("As long as I loose, I play again")
- The Python corner:
  - `np.random.choice` can be handy. You can build your own cumsum random generator if you want to learn how it works, but I don't ask you to.
  - Random number generation is much cheaper (in time) in bulk, especially for custom distribution. More precisely, if you draw k values from a distribution of size n, the cost is $O(n + k\log(k))$. If you plan to use a given distribution a lot, it is best to draw k values in advance (I did use $k = 1000000$) and draw again anytime your stock is depleted.
  - Avoid vectors with more than 10^8 entries if you value your RAM
  - you can save / load your results with `np.load` and `np.save`
  - My own program, probably not optimal, is less than 2K in size. For $n = 10000$, it computes 10 000 trials for each `r in np.arange(.1, 3, .1)` in a few hours on my laptop. All the "tricks" I used are described above in explicit or cryptic ways.