CS 4/5740 Networks, Crowds, and Markets

Project #2: Schelling's Model of Segregation

Submission requirements:

- A .zip file containing your source code. You may use any language you would like.
- A PDF (**submitted separately** to the Canvas assignment) containing each item below that is listed as a **Deliverable.** For each item contained in your PDF, clearly mark which deliverable it is associated with. Plots should be clearly labeled and have descriptive captions.

5740 Students:

- You will modify the baseline model in a way of your choosing and report results; see Deliverable 4.

Tips:

- The technical purpose of this project is to practice thinking algorithmically about a dynamic process which occurs on networks. You will need to carefully plan your data structures and your algorithms to avoid having your runtimes blow up.
- The extra-technical purpose of this project is to start you on a path to curiosity: as you run these simulations and generate and these plots, you owe it to yourself to take the opportunity to *describe what is happening in your own words.* Any time you come up with a result of some kind, try to explain what the result means in non-technical language.
- Visualization: I provide a crude Python script (with usage examples) which you can use to generate animations of your simulations here; however, feel free to create your own visualizations.

Schelling Segregation. In class, we discussed Schelling's famous agents-based model of urban segregation. We analyzed this model at some depth when agents are in a row of houses, but this model works just as well (and is far more interesting) when agents live on a grid square grid. This webpage has a reasonably good overview.

Here are the basics of the grid-based model as you should implement it for this project. The model has 4 parameters:

- `N`: Grid size; your overall grid should be `N`×`N`. You may leave `N` as a parameter in your code, but all your deliverables should use the same value of `N`, which should be at least 30.
- `red_blue_split`: the fraction of total agents that are blue; between 0 and 1 inclusive.

- `t`: the satisfaction (contentedness) threshold; the fraction of an agent's neighbors that it wants to be its own type for it to be satisfied. That is, if `t=0.4` and I have 5 neighbors, I am satisfied (contented) if at least 2 of them are the same type as me. Between 0 and 1 inclusive.
- `pct_empty`: the percentage of grid squares that are empty. If `pct_empty=1`, this means the grid is completely empty; if `pct_empty=0.1`, this means the grid is only 10% empty. Between 0 and 1 inclusive.

How your simulation should work:

1. Initialize an `N`×`N` grid with agent types split according to `red_blue_split` and empty squares allocated according to `pct_empty`. The agents should be dispersed randomly.
2. Iterate through the society and check if each agent is satisfied. Satisfaction is defined as: at least a `t` fraction of your neighbors (occupying the 8 squares surrounding you) are the same type as you. Move each non-satisfied agent to an empty space (select the empty space at random from the available empty spaces).
3. Repeat step 2 until either:
   - Every agent is satisfied, or
   - A maximum iteration count (of your choosing) has been reached.
   - (or use this advanced stopping criterion: track the cross-type-fraction CTF of your society over time, and if it goes a long time without changing much, terminate).

Assignment:

1. Implement Schelling's model of segregation (on square grids) using code of your own design. I will check for copy-pasted code from the internet. You may use any system you like as long as the above idea is implemented faithfully.

   Deliverable 1: The code for your project (you may write your own visualizer or use the Python one that I have provided). In the PDF, provide several examples (for example, using small 3x3 or 4x4 grids) to verify that your "contentedness" and "move-to-empty" functions are operating as intended. Also note: step 2 in the algorithm above is ambiguous (i.e., I didn't tell you whether to iterate over the grid, over the agents, or in what order; I also didn't tell you whether to check satisfaction for all agents before moving any, or if you should check-move check-move one-by-one). Your PDF should explain precisely and explicitly how you decided to resolve this ambiguity! I.e., give a precise description of your satisfaction-checking and agent-moving mechanic.

2. How segregated is the map throughout a simulation run? Create a function which checks the cross-type-fraction (CTF) of a snapshot of the map. As demonstrated in class, this function will divide the number of different-type neighbors (summed over all agents) by the total number of neighbors (summed over all agents). You should be able to call your function at each step along a simulation to see how the CTF changes as agents move around.

   Deliverable 2: Create a series of plots which demonstrate how the CTF changes over the course of several simulation runs of the model. Each plot should have iteration number on the

horizontal axis. You should choose at least 3 sets of parameter values (i.e., at least 3 different combinations red/blue split, satisfaction threshold t, and % empty). For each of these 3 sets of parameter values, do 5-10 simulation runs and plot their CTF traces together. You should have one plot with 5-10 traces for each of the 3 sets of parameter values, for a total of 3 plots. Clearly mark what parameter values gave rise to each of the plots.

3. How does segregation depend on parameter values? Create a function which measures the average CTF *at the end of a simulation run* as a function of input parameters. Your function's call signature should be

   ```
   average_CTF(red_blue_split, t, pct_empty)
   ```

   When called, this function should perform 10 simulation runs with those parameters, record the CTF found at the end of each run, and then return the average of those 10 CTF values.

   Deliverable 3: Three plots created using your `average_CTF` function:

   (a) Plot 1 should display average CTF as a function of `red_blue_split` with the other parameters held constant,

   (b) Plot 2 should display average CTF as a function of `t` with the other parameters held constant, and

   (c) Plot 3 should display average CTF as a function of `pct_empty` with the other parameters held constant.

   Each plot should have at least 10 points on the horizontal axis so that you can see the general shape of the curve. Note that each time you call `average_CTF`, it performs 10 simulation runs, so if you have 10 points on the horizontal axis of each of these plots, this deliverable requires a total of 300 simulation runs. Plan your time accordingly, and if your code is very slow, you might want to consider saving the results of each simulation to disk after each run just in case your code hits a bug before it performs all 300 runs.

4. For the graduate section: Your goal is to come up with a significant modification to Schelling's model and see how it changes. If you find interesting visual phenomena, you might want to Here are some ideas to get you started:

   - Modify neighborhood size: what changes if agents care about the nearest 24 agents rather than just the nearest 8?
   - Modify grid structure: Do the same phenomena emerge if this were played on a hex or triangular grid instead of a square grid? What if the grid were 3D? (note: 3D societies would be harder to visualize, but everything else about the math would work out completely fine.)
   - Modify movement rule: experiment with a different movement rule. What if agents only move to a spot if they would be contented there instead of selecting their move-to spot at random (note that you'll need some kind of fall-back strategy if no acceptable spot exists)? What if agents move to the closest empty spot?
   - Add obstacles: Some cities have "obstacles" like rivers (see St. Louis) and mountains (see Phoenix). How does the presence of obstacles affect emergent segregation in the model?

- Differentiate `t` on the types: what would happen if red agents didn't care about being in the minority, but blue agents cared a lot? Try simulations where `t` is different for red and for blue.

Deliverable 4 (CS 5740 only): Implement your modification and repeat the experiments from Deliverables 2 and 3 using your modification. Discuss how your results with your modified model deviate from the results given by the original model. **Creativity here will be rewarded!**