Catalyst: Agents of Change

Shannon Gallagher

January 27, 2018

1 Outline of software

```
INPUT: Control file with agents file; disease parameters;
 disease model; time steps; parallel options;
 activity schedule; individual agents/instantiations to track
for every time step t
  for every activity at time t
    prob_infection = infect_agents(agents, activity,
                     time, disease)
  end
  agent_status = update_agents(prob_infection)
  update_activities()
  update_disease()
end
OUTPUT: status of agent at each time step, saved as multiple text files;
 number of agents in compartment at each time step;
 specific agents over time; activity info
```

1.1 Infect Agents

```
INPUT: agents, activity, time, disease
for every infected agent ii
  for every neighbor of agent ii, jj
    update_infection_prob(ii,jj, activity, disease, time)
  end
end
OUTPUT: updated agent probabilities
```

2 Description

2.1 General Outline

The benefit of this program is that the agent-based part (the infect_agents portion) is completely modular. The function update_agents() is a multinomial draw. Thus this portion can be used both for heterogeneous individual agents along with homogeneous agents (the compartment-model).

2.2 Infect Agents – the "AM" portion

This function is what differentiates the model from a CM. In this step, agents explicitly interact with one another and hence can change one another. This interaction has the potential to be very slow. Naively, if we were to loop over all pairs of agents and all environments then we would be looking at $O(N^2E)$ run-time where N is the number of agents and E is the number of environments.

Hence, it is critical to be speed up this step. It is worth noting that in the SI-framework, the only interactions that are worthwhile is when an agent is actively infecting another. All other compartment transitions are independent from one another, meaning they do not depend on the status of other agents. Hence we only need to loop over agents in "infectious" states. Moreover, we only need to look at the neighbors' of these infectious agents, specifically those in susceptible states.

We also assume that once an agent is infected by another, she will be unaffected by further infections. As such, once an agent is infected by another, we remove her from the susceptible pool.

We plan to reduce this step to $O(I)$ where I is the number of infectious individuals where, typically $I << N$. The first step is that we only need to loop over infectious agents, for a run time of $O(I)$. We then need to loop over all neighbors of infectious agents. However, we note that we can pre-compute potential neighbors and store them in a dictionary, so look up will be $O(1)$. Neighbors will be over all environments so we no longer have to loop over those. Thus we only have to check whether neighbors of the infectious are susceptible and update the infection probabilities based on the environment activities, pair of interacting agents, and disease parameters.

2.3 Working Prototype v0.0.1

I have implemented the code in the general outline as `catalyst v0.0.1`.

Specifically, I am using 10 agents with 3 environments. Agents with an environment assignment of 0 are considered null assignments (e.g. the agent does not attend school). The 3 compartments are Susceptible, Infectious, and Recovered. In this model, there is a 100% chance of an infectious agent infecting their neighbors. Once infectious, an agent has a 50% chance of recovery at each time step. All recovered agents remain in that state.

The results are displayed in Figure 1, where the first 9 agents are initially susceptible (blue) and the 10th agent is infectious (red). We can glimpse some sense of the neighbor structure in the next step ($t = 1$) as we see agents 6-9 are all infected by agent 10. Agent 10 recovers at time $t = 1$. Then agents 6-9 infect agents 2-5 at time $t = 2$. All agents 2-10 end up recovering by time $t = 6$. We see agent 1 escapes infection completely because it has no neighbors.

The **output** of my program is a csv with dimension $T \times N$ where entry t, n is the agent n's status at time t.

2.3.1 Modular parts

Although the above working prototype is simple, it is modular in many aspects.

1. **Compartments**. The user can identify arbitrarily many compartments. The only requirement is that some compartments must be of class "susceptible" and at least one compartment is an "infectious" agent.

2. **Base probabilities**. This is a 3D array of size $K \times K \times T$ where K is the number of compartments and T is the total number of time steps. Here entry i, j, t is the probability of an agent in state i transitioning to state j at time t to $t + 1$. These base probabilities, may be set, for instance, to the CM probabilities specified in my proposal. These serve as default transitions. And since $K << N$ in general, the memory and/or the computational time required to compute these probabilities is relatively small.

3. **Conditional agent probabilities**. Instead of storing probabilities of transition at each time step and for every possible state the agent is in, we only store a single time step at a time, conditioned on the agent's current compartment. This is a $N \times K$ array where N is the total number of agents and K is the total number of compartments. Then entry n, k is agent n's probability of transitioning to compartment k given it's current compartment. These conditional agent probabilities are by default, taken from the base probabilities. However, these can be updated through the "AM portion" of `catalyst` to allow for individual interactions to take effect.

4. **Environmental and disease properties**. Environment and disease properties are included (although currently without effect) so in the future environment closings and dynamic disease properties may be added into the program.

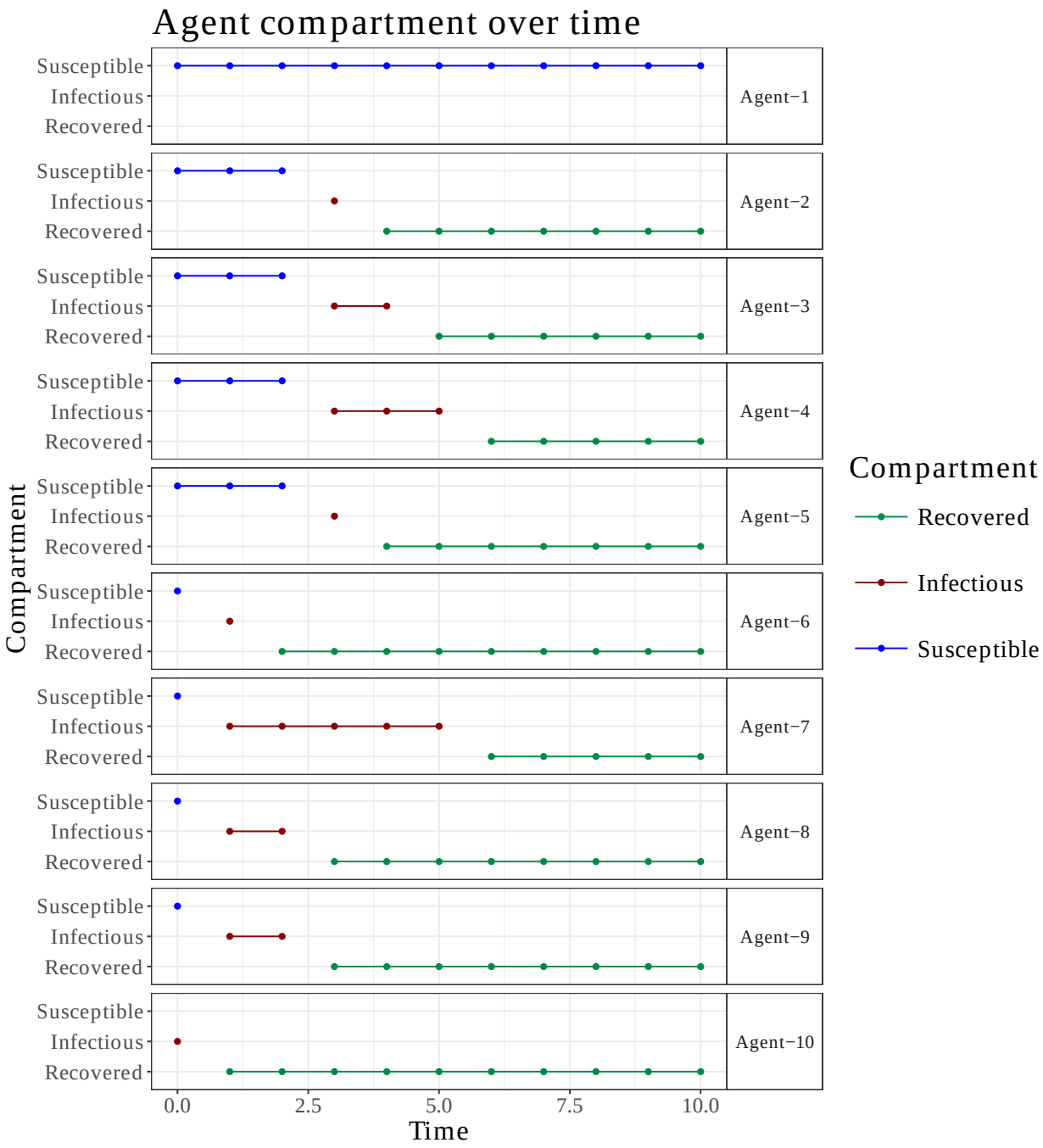

Figure 1: Prototype of catalyst with one infectious agent at time t=0.

3 Current implementation

- A working framework in C
- Incorporation of GNU Standard Library packages for ODE integration and random number generation
- A working SIR model
- Testing framework to initialize agents and environments

- A Dictionary of neighbors through GLib's GHashTable
- Working prototype

4 Next Steps

- Integrate ODE probabilities into model as base probabilities
- Fit ODE models
  - Fit a SIR ODE model after the the AM part is run
  - Fit a SEIR ODE model
  - Attempt to make framework to fit generic model
- Run for a number of repetitions
  - Should be able to share pre-computations like base probabilities and neighbor dictionary
- Option to summarize output instead of saving everything
- Way to track who infects whom (to estimate R0 down the line)

5 Long Term

- Input file standards
- Make the ODE solver more generic to support general systems. The main problem is how to incorporate this symbolically to be an easy input for the control file
- Estimate ODE parameters from an AM through least squares or mle.
- Convert SPEW agents to a viable format for catalyst
- Make nice interface for control file (shiny)
- Parallelization
- Visualize outputs

6 Schedule

6.1 DONE *<2018-01-01 Mon> - <2018-01-13 Sat>*

- Set up preliminaries.
- Learned how to use C libraries, especially GSL (GNU Scientific Library).
- Got a much clearer demonstration of how pointers work
- "Finalize" framework for software

6.2 DONE *<2018-01-15 Mon> - <2018-01-21 Sun>*

6.2.1 The "AM" part

- set up testing framework for cm-am
- Work and get used to GLib and included data structures including Arrays which can grow, singly-linked lists, and hashtables to use within C.
- Made a function to make a dictionary of neighbors.

6.3 DONE *<2018-01-22 Mon> - <2018-01-26 Fri>*

- Store both a single probability and multiple probabilities for agents, ... likely not stored at each step
- Visualize outputs in R

6.4 TODO *<2018-01-29 Mon> - <2018-02-03 Sat>*

- Use SIR base probabilities in current model
- Learn how to use multiple files, headers, and libraries
- Make file??
- Options to summarize output instead of printing out everything