Abstract

Many simple models of disease spread assume homogeneous populations (or population groups) with scalar interaction rates. The goal here is to develop an agent-based model of disease spread that models 1) variability between agents in interaction rates, and 2) the structure of the in-person contact network.

We start by specifying the model and observing its most basic properties, and then spending considerable time investigating its detailed behavior in many different scenarios. By the end, some of the conclusions will be that

- Unsurprisingly, there is a robust critical point as the mean interaction rate varies which determines whether a large fraction of the population is infected or a very small fraction. (This is essentially when the base of the exponent which approximates the number of infected people is close to 1).
- The structure of the contact network significantly affects disease spread (even after controlling for things like graph density).
- It is better to have many small meetings than a few large ones.
- The heterogeneity of interaction rates significantly affects disease spread. In other words, you need to model not just the average amount of interaction, but the distribution of interaction rates from agent to agent.

Code is included, but it is not necessary to understand the model or results.

Making the Agent-Based Model

Outline

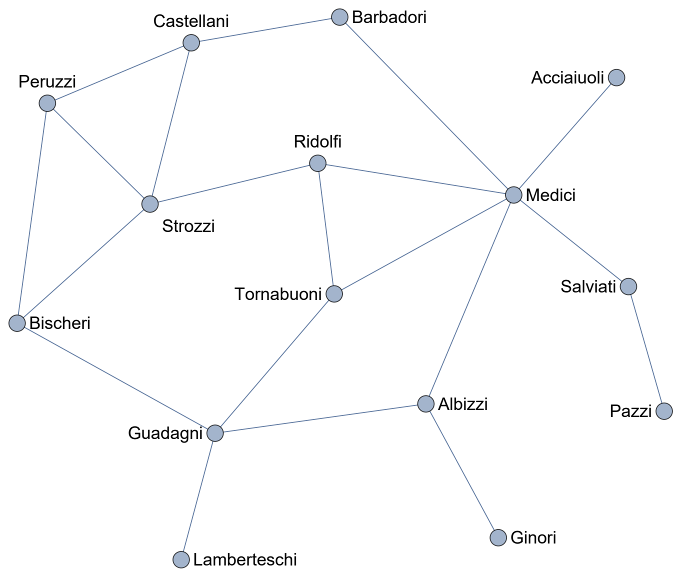
We will start with a graph representing the contact network of the population. Each vertex will represent an agent and edges will connect pairs of agents who might interact. Each agent will be marked as susceptible, infected, or recovered. Initially, a few agents will be marked as infected, while all others will be marked as susceptible. At each step, every agent will pick a random subset of its neighbors in the contact network, and it will “meet” (interact) with them. If a susceptible agent meets an infected one, it becomes infected as well. Meetings are considered symmetric, so it doesn’t matter which agent initiates the meeting. Finally, infected agents will become recovered after some period. We run this sequence repeatedly until there are no infected agents left, at which time the disease has run its course.

There are many parameters here to investigate. For example, we can look at how the structure of the contact network influences disease spread. We can also look at the importance of the distribution that determines the number of meetings each agent initiates, as well as the distribution of recovery times. We will start by implementing the general model, and then we will explore the parameter space.

Implementation

We start by getting a social network to work with. Later we will use something larger and more realistic, but for the purposes of illustration we will start with a very small network:

```
In[∘]:= g = ExampleData[{"NetworkGraph", "FlorentineFamilies"}]
Out[∘]=
```



(For efficiency, we will use associations to store the neighbors for each agent)

```
In[∘]:= agentNeighborhoods[g_] := Merge[Catenate[{#1 → #2, #2 → #1} & @@@ EdgeList[g]], Identity]

In[∘]:= neighbors = agentNeighborhoods[g]
Out[∘]=
<|Acciaiuoli → {Medici},
 Medici → {Acciaiuoli, Barbadori, Ridolfi, Tornabuoni, Albizzi, Salviati},
 Castellani → {Peruzzi, Strozzi, Barbadori}, Peruzzi → {Castellani, Strozzi, Bischeri},
 Strozzi → {Castellani, Peruzzi, Ridolfi, Bischeri}, Barbadori → {Castellani, Medici},
 Ridolfi → {Medici, Strozzi, Tornabuoni}, Tornabuoni → {Medici, Ridolfi, Guadagni},
 Albizzi → {Medici, Ginori, Guadagni}, Salviati → {Medici, Pazzi},
 Pazzi → {Salviati}, Bischeri → {Peruzzi, Strozzi, Guadagni},
 Guadagni → {Tornabuoni, Albizzi, Bischeri, Lamberteschi},
 Ginori → {Albizzi}, Lamberteschi → {Guadagni}|>
```

At each step of the simulation, each agent will pick some subset of its neighbors to meet with. We need to specify the distribution that determines how many neighbors each agent will meet with at each step. For now, we will choose something somewhat arbitrary (an exponential distribution), but later we will explore which properties of these distributions are important:

```
In[∘]:= meetingCounts = AssociationThread[Keys[neighbors], ExponentialDistribution[2]]
Out[∘]=
<|Acciaiuoli → ExponentialDistribution[2],
 Medici → ExponentialDistribution[2], Castellani → ExponentialDistribution[2],
 Peruzzi → ExponentialDistribution[2], Strozzi → ExponentialDistribution[2],
 Barbadori → ExponentialDistribution[2], Ridolfi → ExponentialDistribution[2],
 Tornabuoni → ExponentialDistribution[2], Albizzi → ExponentialDistribution[2],
 Salviati → ExponentialDistribution[2], Pazzi → ExponentialDistribution[2],
 Bischeri → ExponentialDistribution[2], Guadagni → ExponentialDistribution[2],
 Ginori → ExponentialDistribution[2], Lamberteschi → ExponentialDistribution[2]|>
```

Next, we need a distribution that will model how long agents remain infected before recovering. Again, we will pick something somewhat arbitrary for now (though many SIR models implicitly use an exponential distribution here), but we will try different distributions later:

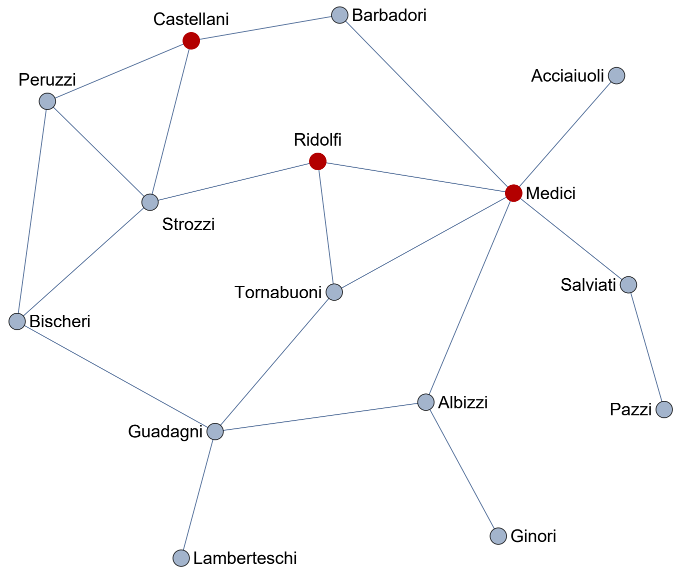
```
In[∘]:= recoveryTime = ExponentialDistribution[1 / 7];
```

Next, we will make an association that maps agents to their states. Each agent can be "S" (susceptible), "R" (recovered), or {"I", *n*} (infected for the next *n* steps). A few agents will start infected, while the rest will be susceptible:

```
In[∘]:= agentStates = Join[
          AssociationThread[Keys[neighbors], "S"],
          AssociationMap[
            {"I", Floor@RandomVariate[recoveryTime]} &, RandomSample[Keys[neighbors], 3]]
         ]
Out[∘]=
<|Acciaiuoli → S, Medici → {I, 6}, Castellani → {I, 5}, Peruzzi → S,
 Strozzi → S, Barbadori → S, Ridolfi → {I, 7}, Tornabuoni → S, Albizzi → S,
 Salviati → S, Pazzi → S, Bischeri → S, Guadagni → S, Ginori → S, Lamberteschi → S|>
```

Here the infected agents are highlighted in red:

```
In[∘]:= HighlightGraph[g, Keys@Select[agentStates, MatchQ[{"I", _}]]]
Out[∘]=
```

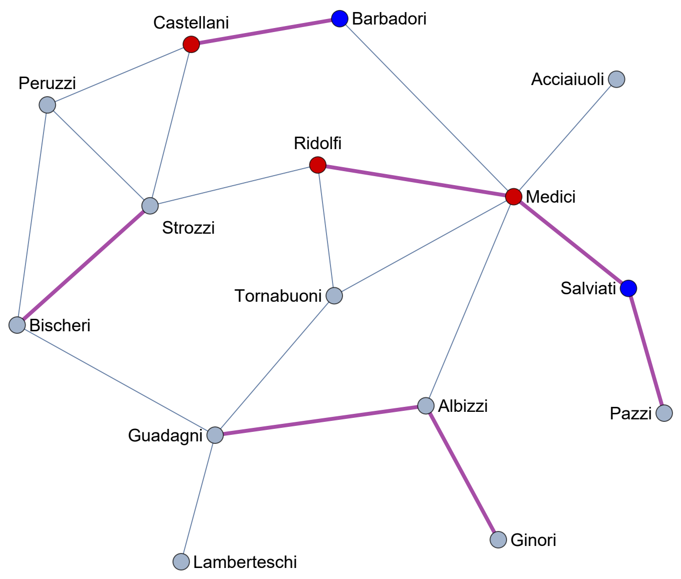


We then generate the list of meetings between agents for this step:

```
In[∘]:= meetings = DeleteDuplicatesBy[Flatten[
          Function[a,
            {a, #} & /@
             RandomSample[
               neighbors[a],
               Clip[Round[RandomVariate[meetingCounts[a]]], {0, Length[neighbors[a]]}]]
           ] /@ Keys[neighbors],
          1], Sort]
Out[∘]=
        {{Castellani, Barbadori}, {Ridolfi, Medici}, {Albizzi, Ginori},
         {Salviati, Medici}, {Pazzi, Salviati}, {Bischeri, Strozzi}, {Guadagni, Albizzi}}
```

And we can find all of the susceptible agents that met with an infected one (with infected agents in red, meetings in purple, and susceptible agents who are being exposed in blue):

```
In[◦]:= With[{infected = Keys@Select[agentStates, MatchQ[{"I", _}]]},
         HighlightGraph[g, {
           infected,
           Style[UndirectedEdge @@@ meetings, Purple, Thick],
           Style[
            Complement[Flatten[Select[meetings, IntersectingQ[#, infected] &]], infected], Blue]
          }]]
Out[◦]=
```



Putting this all together, we get the following implementation:

In[]:=
```
runStep[agentStates_, neighbors_, meetingCounts_, recoveryTime_] :=
Module[{meetings,agentMeetings},
    meetings =
        DeleteDuplicatesBy[Flatten[
            Function[a,
                {a,#}&/@
                    RandomSample[neighbors[a],
                        Clip[Round[RandomVariate[meetingCounts[a]]],{0,Length[neighbors[a]]}]]
            ]/@Keys[neighbors],
            1],
        Sort];
    agentMeetings = Merge[Catenate[{#1→#2,#2→#1}&@@@meetings],Identity];
    AssociationMap[
        Replace[agentStates[#],{
            "S":→If[
                MemberQ[Lookup[agentStates,agentMeetings[#]],{"I",_}],
                {"I",RandomVariate[recoveryTime]},
                "S"],
            {"I",n_}:→If[n>0,{"I",n-1},"R"],
            "R"→"R"
        }]&,
    Keys[agentStates]]
]
```

In[]:=
```
simulateSlow[neighbors_, meetingCounts_, recoveryTime_, initialInfected_, maxSteps_:Infinity] :=
Module[{agentStates, steps},
    agentStates = Join[
        AssociationMap["S"&,Keys[neighbors]],
        AssociationMap[{"I",RandomVariate[recoveryTime]}&,
            RandomSample[Keys[neighbors],initialInfected]]
    ];
    steps = NestWhileList[
        runStep[#,neighbors,meetingCounts,recoveryTime]&,
        agentStates,
        MemberQ[{"I",_}],
        1,
        maxSteps
    ];
    {Count[#,{"I",_}],Count[#,"S"],Count[#,"R"]}&/@steps
]
```

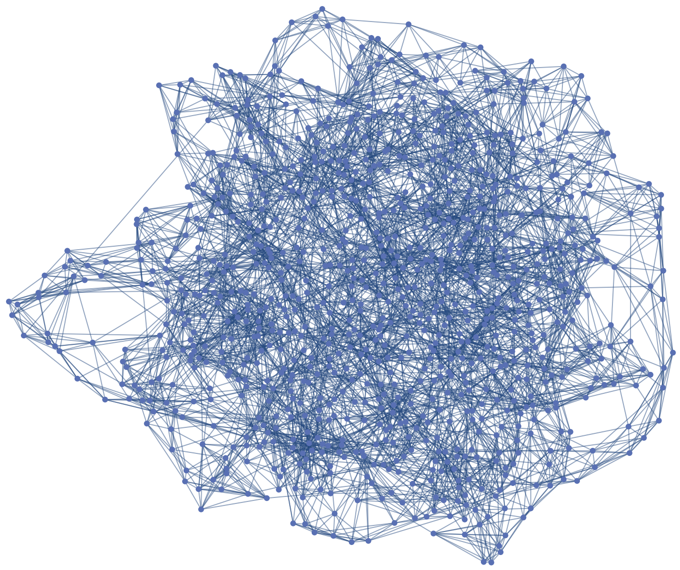
In the hidden cell below there is a slightly optimized version of this model which we will mostly be using moving forward:

Efficient implementation

Simple Example

Let’s run a simple example to test this model. First, we need a contact network to run on. We will look at different graph distributions later, but we can start by using the Watts-Strogatz graph distribution which is a decent approximation of a real social network:

```
In[]:= g = RandomGraph[WattsStrogatzGraphDistribution[1000, 0.1, 5]]
Out[]=
```



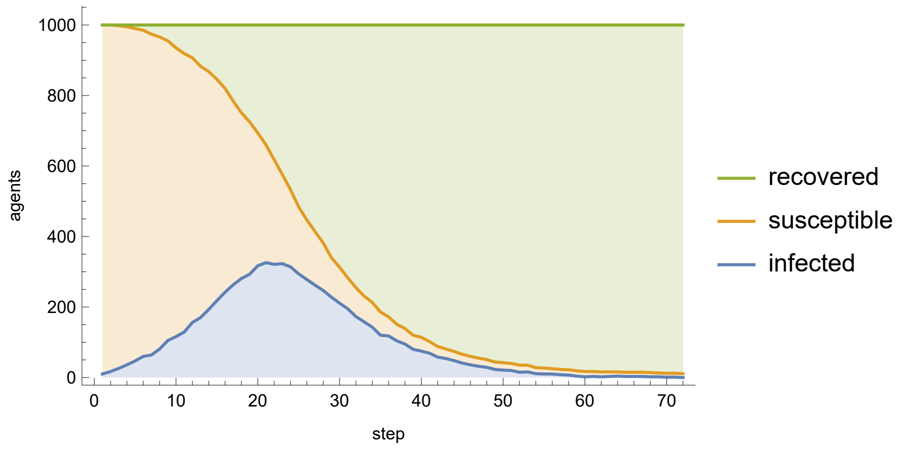
Next, we run a simulation on this network with the same exponential distribution with mean 0.4 modeling the number of outgoing meetings for each agent, the distribution of recovery times being exponential with mean 7, and 10 random agents initially infected:

```
In[]:= sim = simulate[
          agentNeighborhoods[g],
          AssociationMap[ExponentialDistribution[1 / 0.4] &, VertexList[g]],
          ExponentialDistribution[1 / 7],
          10,
          Infinity
        ];
```

The output is a list of triplets each containing the number of agents that are infected, susceptible, and recovered, for each step. We can use a standard visualization for SIR models to understand the output:

```
In[]:= sirPlot[sim_, opts___] := StackedListPlot[Transpose[sim],
         Frame → {{True, False}, {True, False}}, FrameLabel → {"step", "agents"}, opts]
```

In[∘]:=
```
sirPlot[sim, PlotLegends → {"infected", "susceptible", "recovered"}]
```
Out[∘]=



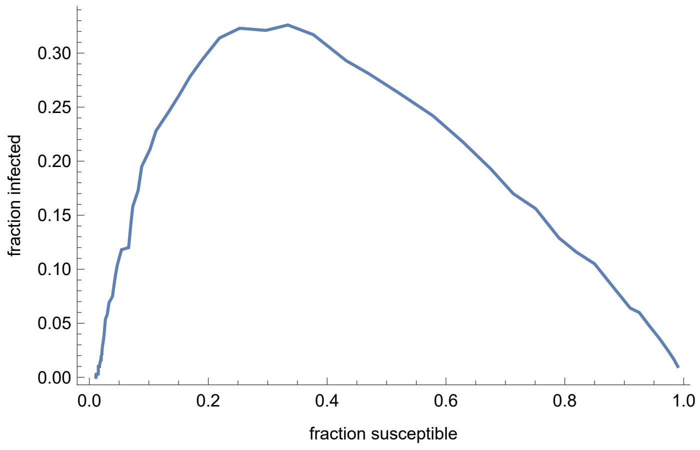
At the beginning, almost all agents are susceptible. Then, an exponentially increasing number of agents become infected, while previously infected agents recover. By the end, there are a handful of susceptible agents left who were never infected, while the rest were infected and recovered. This is essentially the output we would expect from a simple SIR model.

Phase Plots and Basic Properties

We could also visualize the evolution of this simulation as a phase plot with a curve through three-dimensional space, with axes for the number susceptible, infected, and recovered. However, because the number of agents remains fixed, we can convey the same information by just picking any two dimensions:

In[∘]:=
```
siMultiPlot[sims_, opts___] :=
 ListLinePlot[sims⟦All, All, {2, 1}⟧ / Total /@ sims⟦All, 1⟧,
  opts, Frame → {{True, False}, {True, False}},
  FrameLabel → {"fraction susceptible", "fraction infected"}]
siPlot[sim_, opts___] := siMultiPlot[{sim}, opts]
```

```
In[•]:= siPlot[sim]
Out[•]=
```



We started in the state on the bottom right (most susceptible and only a few infected), and ended on the bottom left (all recovered and so none susceptible or infected).

We can expand this into a vector field which will show us where the critical points and attractors are. However, we have to change the recoveryTime to be generated by a geometric distribution. The problem is that this phase plot collapses other important dimensions like how long the infected agents must wait before recovering. The only distribution where this information isn't needed is a geometric distribution, because a geometric distribution models the recovery time if there was a constant probability of recovering at each step, which doesn't require hidden state.

Using a geometric distribution for the recovery time, we can define a function that will take a position in SI space (susceptible-infected space) and return a displacement in SI space, from which we can generate the phase plot:

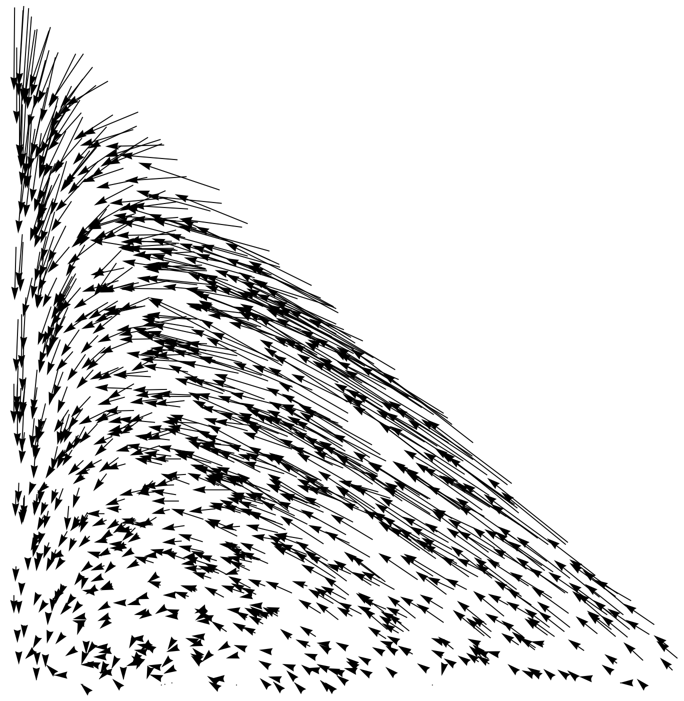
```
In[•]:= siVector[{s_, i_}, neighbors_, meetingCounts_, recoveryTime_] :=
         Module[{infected, susceptible, newStates},
          infected = RandomSample[Keys[neighbors], i];
          susceptible = RandomSample[Complement[Keys[neighbors], infected], s];
          newStates = runStep[
            Join[
             AssociationMap[{"I", RandomVariate[recoveryTime]} &, infected],
             AssociationThread[susceptible, "S"],
             AssociationThread[Complement[Keys[neighbors], infected, susceptible], "R"]
            ],
            neighbors,
            meetingCounts,
            recoveryTime
           ];
          {Count[newStates, "S"], Count[newStates, {"I", _}]} - {s, i}
         ]
```

```
In[•]:= siVectorField[neighbors_, meetingCounts_, recoveryTime_, n_] :=
         {#, siVector[#, neighbors, meetingCounts, recoveryTime]} & /@
          Round@
           RandomPoint[Triangle[{{0, 0}, {Length[neighbors], 0}, {0, Length[neighbors]}}], n]
```

We can use this to generate a simple phase plot:

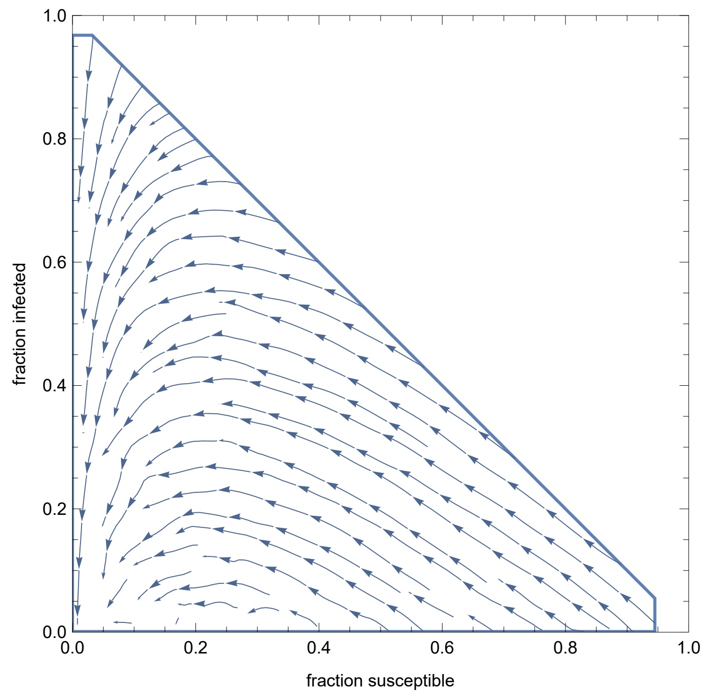
```
In[•]:= vectorField = siVectorField[
           agentNeighborhoods[g],
           AssociationMap[ExponentialDistribution[1 / 0.4] &, VertexList[g]],
           GeometricDistribution[1 / (1 + 7)],
           1000
          ];

In[•]:= Graphics[{Arrowheads[Small], Arrow[{#1, #1 + #2} & @@@ vectorField]}]
Out[•]=
```



```
In[ ]:= ListStreamPlot[vectorField / VertexCount[g], ⋯ ]
```

```
Out[ ]=
```



We can see that there is an attractor at {0, 0}, suggesting that with with these parameters everyone will become infected (because nobody susceptible will be left). Later we will see how the inputs affect this phase plot.

Which Parameters are Important?

There is a huge parameter space for this model, so we won't be able to go through everything. There is the structure of the underlying contact network. There is the distribution of recovery times. There are the distributions which determine the number of meetings that take place (as well as the distribution of those distributions). And so on.

For practical purposes, there are two outputs for any simulation that seem important: the total fraction and the peak fraction of agents who are ever infected (the latter of which is important for "flatten the curve" strategies).

We will experiment with the input parameters to try to see their relationship to these two outputs, and to simulate mitigation strategies.

Critical Points

We start by investigating the dynamics of the the total infected and peak infected variables. To do so, we can define functions for extracting these properties:

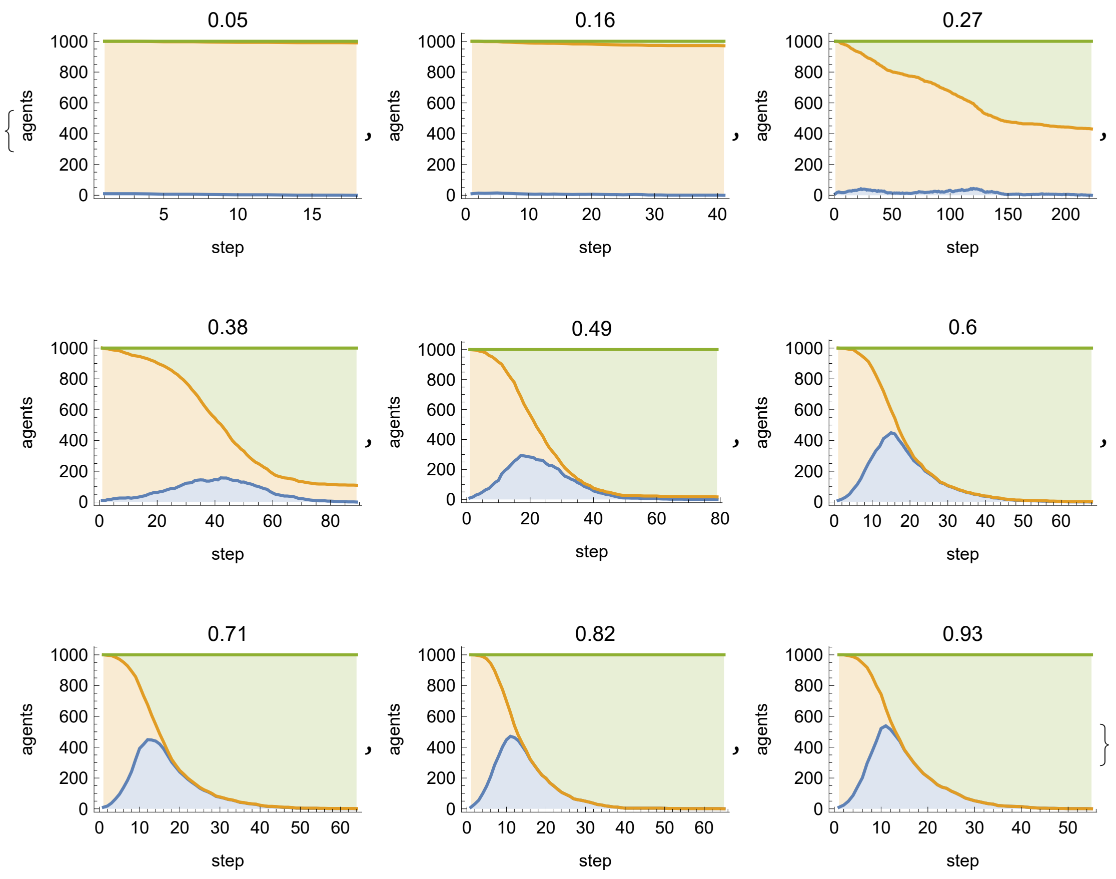
```
In[ ]:= peakInfected[sim_] := Max[sim⟦All, 1⟧] / Total[sim⟦1⟧]
```

```
In[∘]:= totalInfected[sim_] := (sim⟦-1, 1⟧ + sim⟦-1, 3⟧) / Total[sim⟦-1⟧]
```

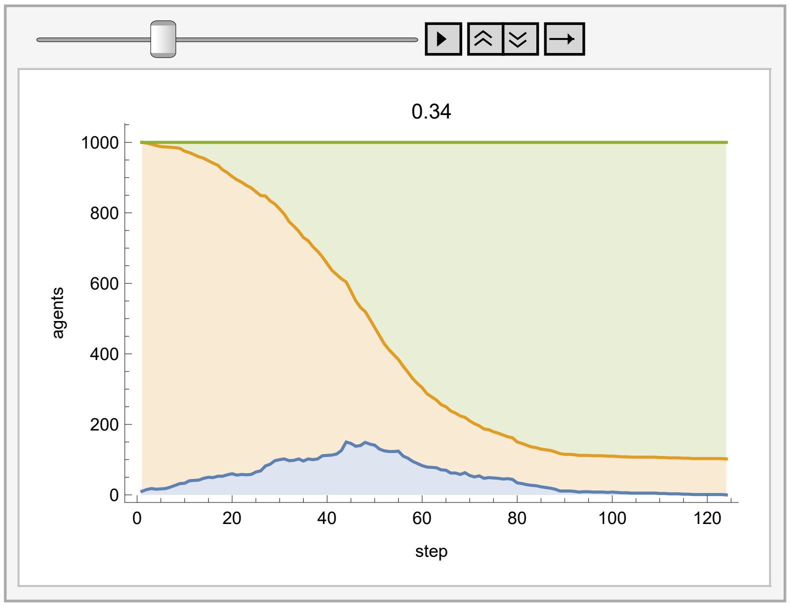
We might guess that one of the most important variables is the mean number of meetings per agent per step, which is essentially the mean interaction rate. We can control this by altering the distribution of meeting counts. For now, we will give every agent the same distribution and just change the mean, simulating a homogeneous population:

```
In[∘]:= sims = Association@Table[
            m → simulate[
              agentNeighborhoods[g],
              AssociationMap[ExponentialDistribution[1 / m] &, VertexList[g]],
              GeometricDistribution[1 / (1 + 7)],
              10,
              Infinity
             ],
            {m, 0.05, 1, 0.01}
           ];
```

```
In[∘]:= KeyValueMap[sirPlot[#2, PlotLabel → #1] &, sims⟦ ;; ;; 11⟧]
Out[∘]=
```

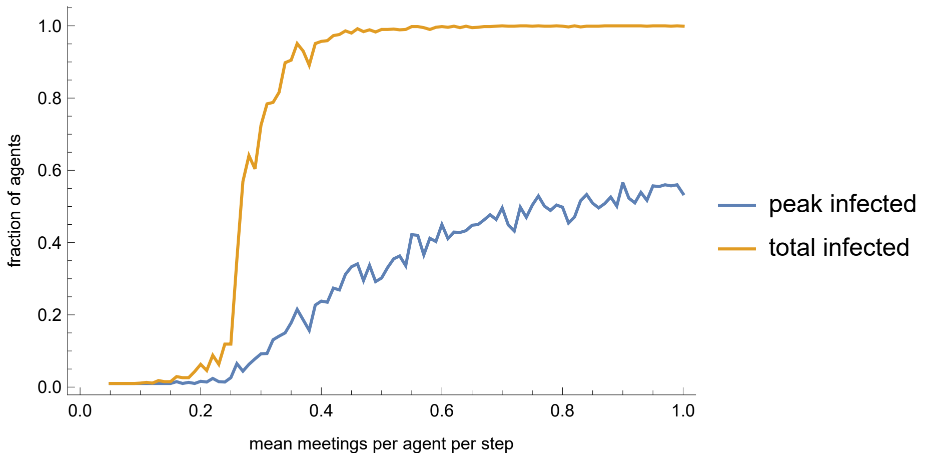


```
In[∘]:= ListAnimate[KeyValueMap[sirPlot[#2, PlotLabel → #1] &, sims], AnimationRunning → False]
Out[∘]=
```



We can see that as the number of meetings increases, it passes a critical point that determines whether almost everyone gets infected or almost nobody. We can use the peak infected and total infected functions to measure this more precisely:

```
In[∘]:= ListLinePlot[
         {KeyValueMap[List, peakInfected /@ sims], KeyValueMap[List, totalInfected /@ sims]}, ⋯ ]
Out[∘]=
```



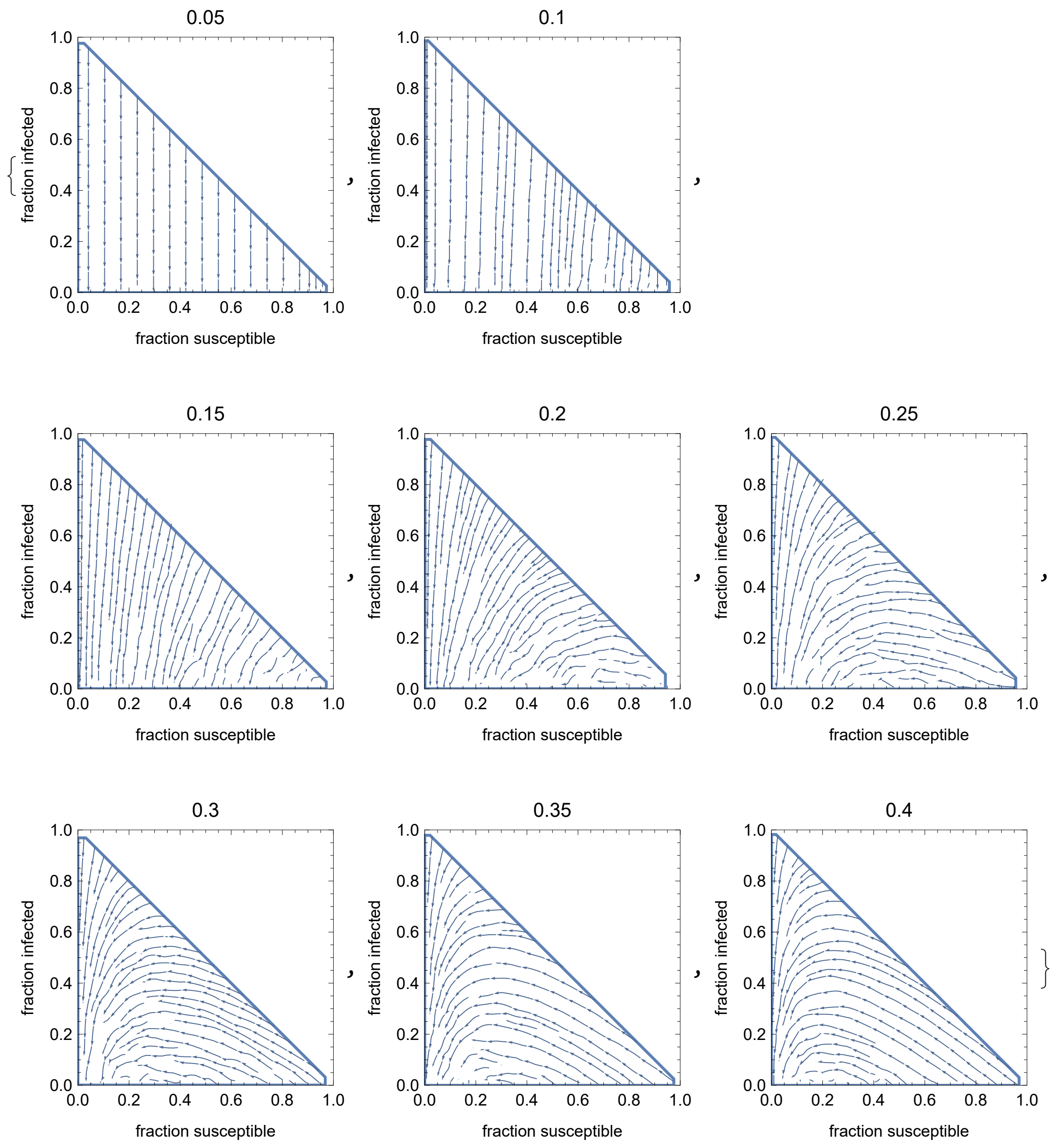
We can clearly see the critical point around 0.2. Interestingly, while the total infected jumps almost immediately to 100%, the peak infected is much more gradual. This suggests that incremental progress in reducing interaction can have an effect on the peak infected while the total infected requires more radical intervention (which is essentially the flatten the curve strategy).

We can also visualize this transition with phase plots:

```
In[•]:= vectorFields = Association@Table[
          m → siVectorField[
            agentNeighborhoods[g],
            AssociationMap[ExponentialDistribution[1 / m] &, VertexList[g]],
            GeometricDistribution[1 / (1 + 7)],
            1000
           ],
          {m, 0.05, 1, 0.01}
         ];
```

```
In[•]:= KeyValueMap[ListStreamPlot[#2 / VertexCount[g], PlotLabel → #1, ⋯] &,
         vectorFields⟦ ;; 40 ;; 5⟧]
```

```
Out[•]=
```



Here we see that when there is limited interaction, there is an attractor along the bottom edge (the infected = 0 edge), but as the number of meetings goes up that attractor is pushed to the bottom left corner (the infected = 0 and susceptible = 0 point). We can see that there is a small leftward push for any number of meetings, but that the downward push is so fast that the number of infected goes to 0 before there can be leftward drift. As the number of meetings goes up, each vector gets more “lift” which allows to to travel further to the left.

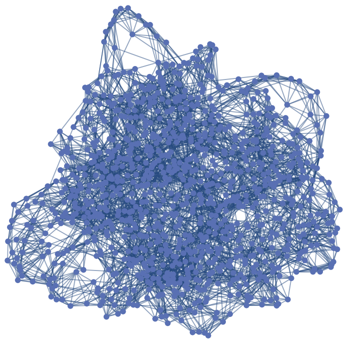
Contact Networks

So far we have been using the same Watts-Strogatz graph for our contact network. We could, however, run this model on any graph. Watts-Strogatz is often used to simulate social networks because it generates small-world networks, but there are other graph distributions that simulate various other aspects of social or contact networks. More generally, we do not know yet whether the structure of the contact network matters at all, so we will start by examining that question.

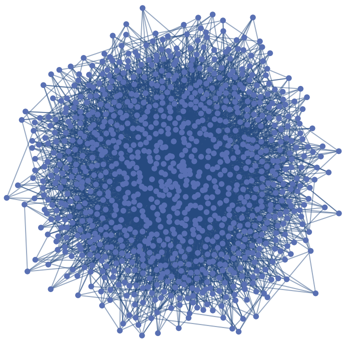
There are a few graphs that we should try:

```
In[ ]:= gs = <|
          "Watts-Strogatz" → RandomGraph[WattsStrogatzGraphDistribution[1000, 0.1, 5]],
          "Barabasi-Albert" → RandomGraph[BarabasiAlbertGraphDistribution[1000, 5]],
          "Grid" → GridGraph[{25, 40}],
          "DelaunayTriangulation" →
           IndexGraph[MeshConnectivityGraph[DelaunayMesh[RandomPoint[Disk[], 1000]]]],
          "Grid3D" → GridGraph[{10, 10, 10}],
          "Random" → RandomGraph[{1000, 5000}]
         |>
Out[ ]=
```

⟨| Watts-Strogatz →

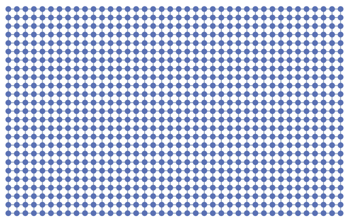


, Barabasi-Albert →

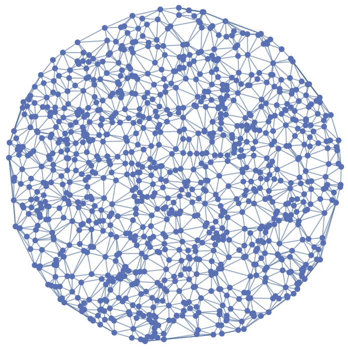


,

Grid →

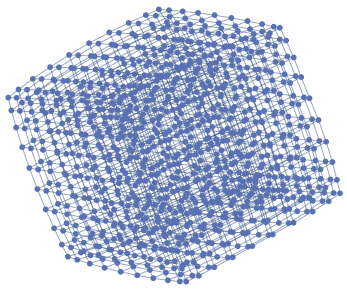


, DelaunayTriangulation →

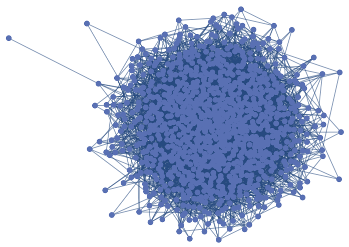


,

Grid3D →



, Random →



|⟩

We start with Watts-Strogatz for the reasons mentioned above. Next, we will look at the Barabasi-Albert graph distribution which is also often used to simulate social networks. The tradeoff between Watts-Strogatz and Barabasi-Albert is that Watts-Strogatz has realistic clustering and an unrealistic degree distribution, while Barabasi-Albert has unrealistic clustering and a realistic degree distribution. (Barabasi-Albert has a power law degree distribution which could be good for modeling how there are a few people who interact with lots of different people, like cashiers, while most people interact with a smaller consistent set.)

Next we use a grid graph because it is analogous to how many simple spatial SIR models work. It is also not a small-world network, which might yield different results. We can also use the Delaunay triangulation of random points to generate a less structured planar graph. Grid3D is interesting because it is structurally similar to the grid but it has a higher graph dimension.

Finally, we look at a random graph with approximately the same density as the small-world networks.

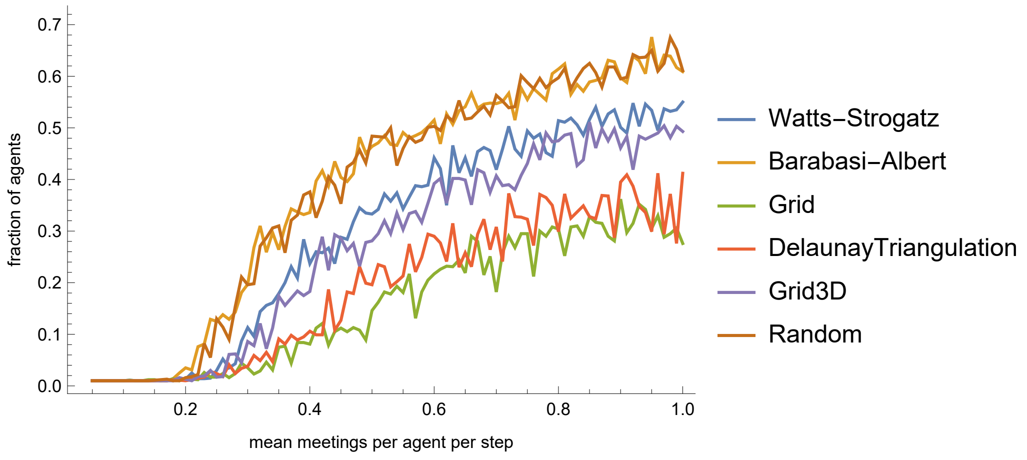
We can now look at how the results change as we change the underlying contact network:

```
In[∘]:= simulateWithGraph[g_] :=
         Association@Table[
           m → simulate[
             agentNeighborhoods[g],
             AssociationMap[ExponentialDistribution[1 / m] &, VertexList[g]],
             GeometricDistribution[1 / (1 + 7)],
             10,
             Infinity
            ],
           {m, 0.05, 1, 0.01}
          ]
```

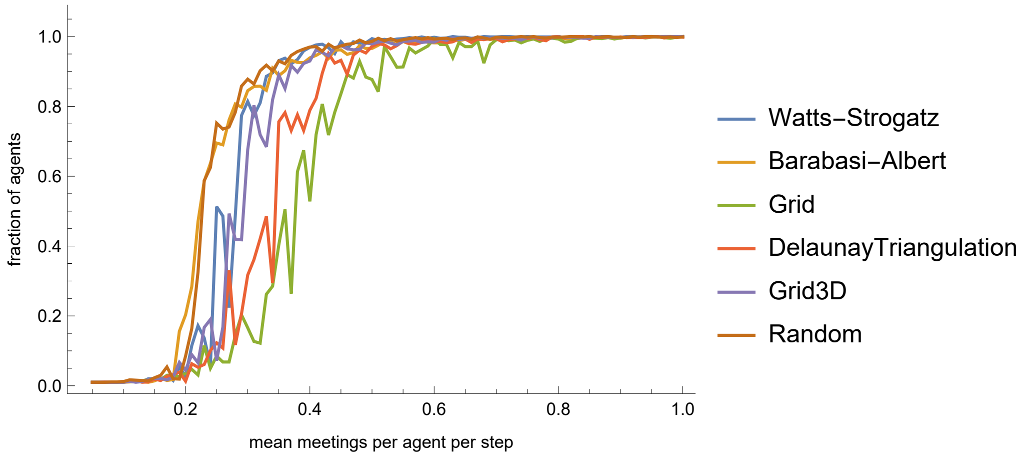
```
In[∘]:= plotPeakInfected[simss_, opts___] :=
         ListLinePlot[KeyValueMap[List, peakInfected /@ #] & /@ simss, ⋯, opts]
        plotTotalInfected[simss_, opts___] :=
         ListLinePlot[KeyValueMap[List, totalInfected /@ #] & /@ simss, ⋯, opts]
```

```
In[∘]:= graphSims = simulateWithGraph /@ gs;
```

```
In[∘]:= plotPeakInfected[graphSims]
Out[∘]=
```



```
In[•]:= plotTotalInfected[graphSims]
Out[•]=
```



The planar networks saw much slower growth. The random network and the Barabasi-Albert network were very similar and showed very fast growth. Watts-Strogatz gave an intermediate result.

This ordering seems like it coincides with the graph dimension (which models how many vertices will be in a ball of a given radius). The graph radius is related to the dimension for graphs like these, and we can see that the radius accuracy predicts how fast the disease spreads:

```
In[•]:= Sort[GraphRadius /@ gs]
Out[•]=
<|Barabasi-Albert → 3, Random → 4, Watts-Strogatz → 6,
 Grid3D → 15, DelaunayTriangulation → 16, Grid → 32|>
```

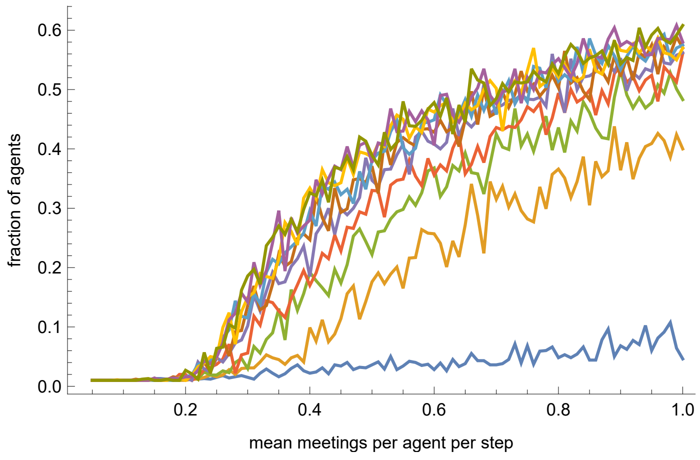
The main conclusion here is that even if the number of meetings is held the same, agents with a larger number of neighbors that they could potentially meet cause faster spread, demonstrating that the structure of the contact network is important independent of the amount of interaction.

(It should also be noted that this is not just because there are “wasted meetings” where the number of meetings is greater than the number of neighbors. Empirically, this is a rare occurrence even for a high mean number of meetings per agent per step.)

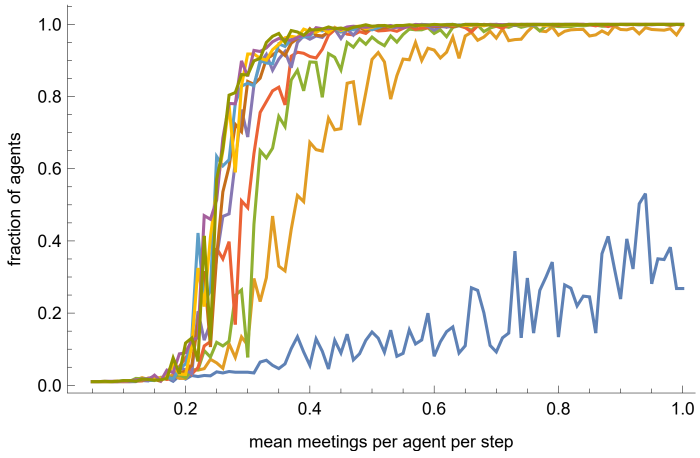
To test this further, we can try generating several graphs from the same distribution but with different densities:

```
In[•]:= graphDensitySims = AssociationMap[
    simulateWithGraph[RandomGraph[WattsStrogatzGraphDistribution[1000, 0.1, #]]] &,
    Range[1, 10]];
```

```
In[ ]:= plotPeakInfected[Values[graphDensitySims]]
Out[ ]=
```



```
In[ ]:= plotTotalInfected[Values[graphDensitySims]]
Out[ ]=
```



Density is related to dimension, but it is not exactly the same. The Barabasi-Albert and Watts-Strogatz graphs used above were the same density but different dimensionality. However, in this example with only Watts-Strogatz graphs we see that density is related to the speed of disease spread.

Joined Networks

While social networks are generally considered to be small-world, it is unclear if this is also true of in-person contact networks. People who are geographically close likely have more in-person contact than people who are not. Somebody who lives thousands of miles away might be a friend, but they are many steps away on the network of to person-to-person contact. It is not immediately clear what this contact network should look like, but we will attempt to make something that might approximate it.

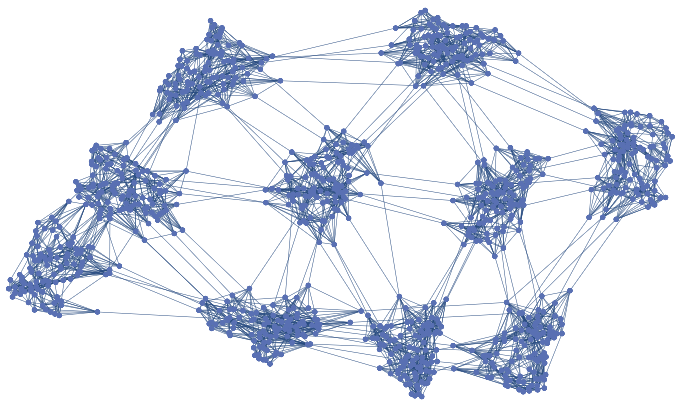
We start by making a “supergraph” made up of smaller clusters. Each cluster will be a small-world network, modeling a local community which could be like a town or city, while the network of clusters will be some planar graph representing the connectivity between these geographic entities. We will start by writing a function that takes a graph representing the large-scale structure along with a graph distribution and returns a version of the structure graph where every vertex is replaced by a cluster sampled from the graph distribution:

```
In[ ]:= generateSupergraph[structure_, graphDistribution_, edgeThickness_ : 1] :=
         Module[{clusters, g, previousCount = 0, vertices, edges, connectingEdges},
          clusters = Table[
            Module[{cluster},
             cluster = IndexGraph[RandomGraph[graphDistribution], previousCount];
             previousCount += VertexCount[cluster];
             {VertexList[cluster], EdgeList[cluster]}
            ], {i, VertexCount[structure]}];
          {vertices, edges} = Union @@@ Transpose[clusters];
          connectingEdges = UndirectedEdge @@ RandomChoice@*First /@ clusters⟦{#1, #2}⟧ & @@@
            Catenate[ConstantArray[EdgeList[structure], edgeThickness]];
          Graph[vertices, Join[edges, connectingEdges]]
         ]
```

For our purposes, we will use Watts-Strogatz for the clusters (communities) and the Delaunay triangulation of random points for the large scale structure (the contact between communities). We can vary how many edges connect adjacent clusters to change how interconnected the communities are:

```
In[ ]:= generateSupergraph[
         IndexGraph@MeshConnectivityGraph@DelaunayMesh@RandomPoint[Disk[], 10],
         WattsStrogatzGraphDistribution[100, 0.1, 5], 5]
```

Out[ ]=



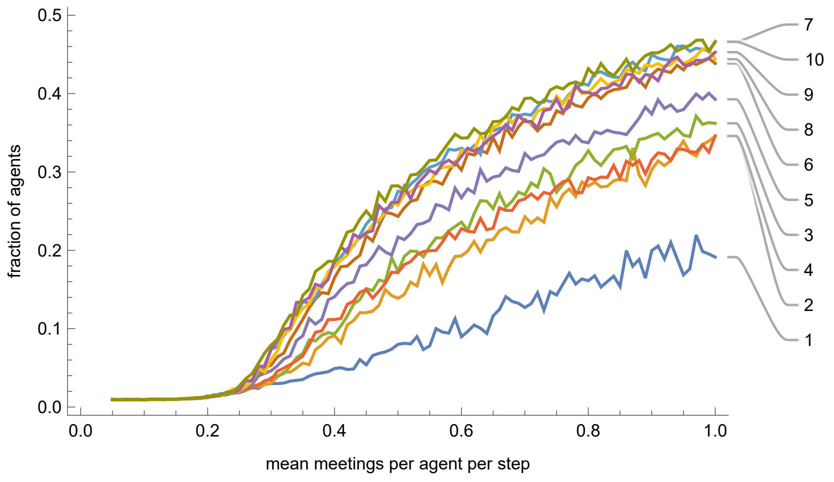
We can now run simulations on these graphs in the same way as before except we will force all the initially infected to start in one cluster (so it is forced to spread from cluster to cluster):

```
In[∘]:= superGraphSimsByConnectivity =
          Table[
           Module[{g},
            g = generateSupergraph[
              IndexGraph@MeshConnectivityGraph@DelaunayMesh@RandomPoint[Disk[], 10],
              WattsStrogatzGraphDistribution[100, 0.1, 5], n];
            Table[
             Association@Table[
               m → simulate[
                 agentNeighborhoods[g],
                 AssociationMap[ExponentialDistribution[1 / m] &, VertexList[g]],
                 GeometricDistribution[1 / (1 + 7)],
                 RandomInteger[{1, 100}, 10],
                 Infinity
                ],
               {m, 0.05, 1, 0.01}
              ],
             25]
           ], {n, 1, 10}];
```

The plot below shows the peak fraction of infected agents as a function of the mean interaction rate for 10 runs with different numbers of edges connecting adjacent clusters:

```
In[∘]:= ListLinePlot[MapIndexed[Callout,
          Merge[Map[peakInfected, #, {2}], Mean] & /@ superGraphSimsByConnectivity], ⋯ ]
```

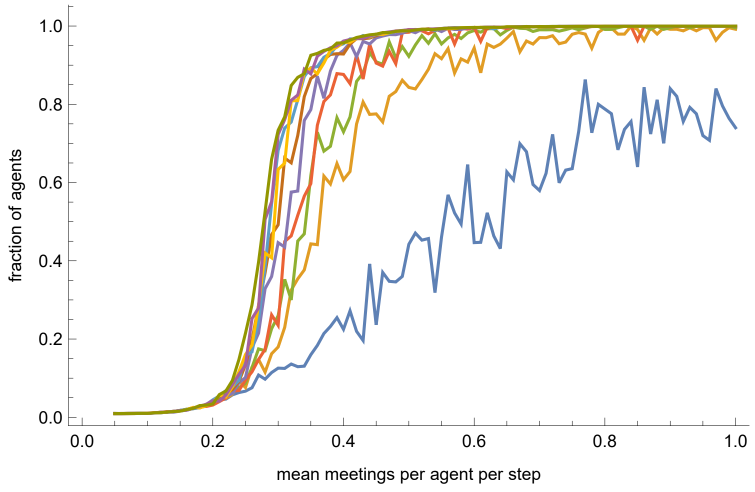
Out[∘]=



The next plot is the same except we calculate the total fraction infected instead of the peak fraction:

```
In[]:= ListLinePlot[
        Merge[Map[totalInfected, #, {2}], Mean] & /@ superGraphSimsByConnectivity, ⋯ ]
```

Out[]=



In both of these cases we see that when clusters are only weekly connected, the disease unsurprisingly spreads slower. In particular, we see that the total fraction infected does not transition as quickly from close to 0% to close to 100%, but instead ramps up slowly. We get this because while disease spreads quickly inside the small-world clusters, it does not spread as fast on the large-scale planar graph, meaning that not all clusters get infected. As the clusters become more interconnected, however, it limits to the curves we get with a giant small-world network. Interestingly, however, the curve we get when clusters are only weakly connected seems different from the curve we get on planar networks of this type, suggesting that there is modeling value by simulating interactions within the clusters.

In other words, reducing interaction between communities can be more effective than reducing interaction within communities.

Interaction Rates

Many interventions (like social distancing) involve changing the frequency or size of meetings, so we should look at how the distribution of interaction rates changes the the peak and total infection rates.

We'll start by defining a function for investigating this property:

```
In[]:= g = RandomGraph[WattsStrogatzGraphDistribution[1000, 0.1, 5]];
```

In[∘]:=
```
simulateWithMeetingCounts[meetingCounts_] :=
 Association@Table[
   m → simulate[
     agentNeighborhoods[g],
     AssociationMap[meetingCounts[m] &, VertexList[g]],
     GeometricDistribution[1 / (1 + 7)],
     10,
     Infinity
    ],
   {m, 0.05, 1, 0.01}
  ]
```

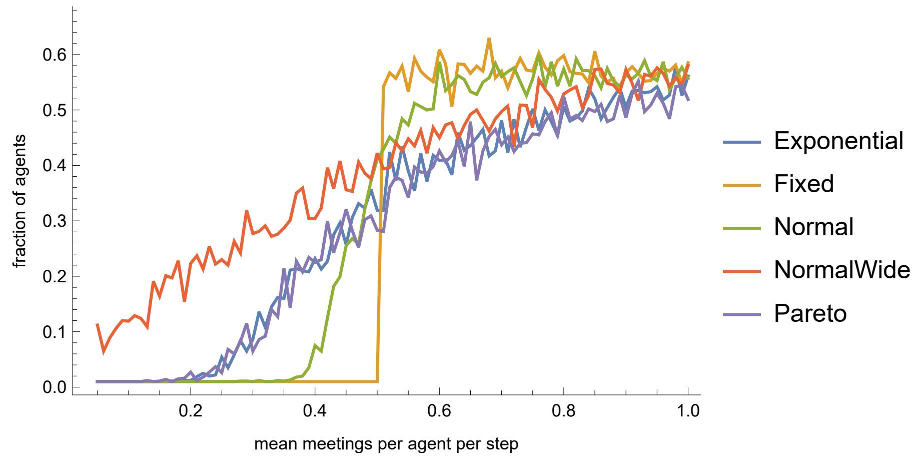
We have already looked at what happens when we change the mean meeting count (there is a critical point where the disease switches from dying out to infecting everyone). However, so far we have only used exponential distributions to model the number of meetings per agent. We begin by trying different distributions:

In[∘]:=
```
meetingCountDists = <|
   "Exponential" → (ExponentialDistribution[1 / #] &),
   "Fixed" → (UniformDistribution[{#, #}] &),
   "Normal" → (NormalDistribution[#, 0.1] &),
   "NormalWide" → (NormalDistribution[#, 0.5] &),
   "Pareto" → (ParetoDistribution[(b - 1) #, b, 0] /. b → 5 &)
   |>;
```

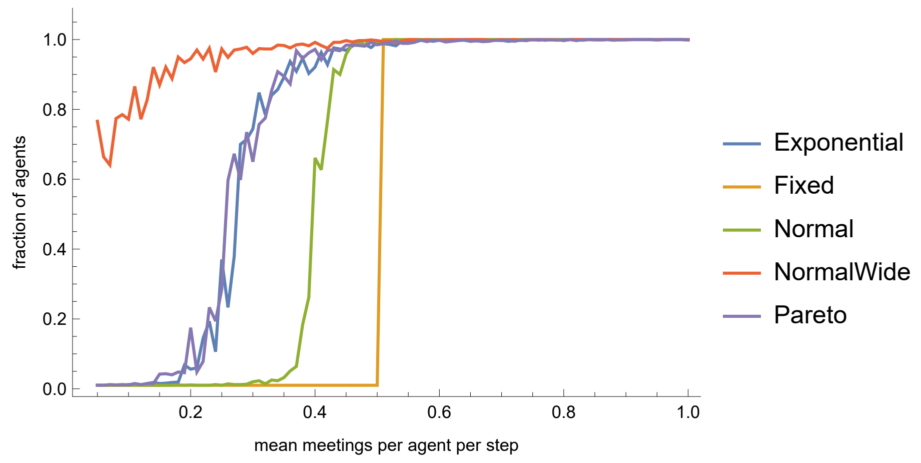
In[∘]:=
```
meetingCountSims = simulateWithMeetingCounts /@ meetingCountDists;
```

In[∘]:=
```
plotPeakInfected[meetingCountSims]
```

Out[∘]=



```
In[∘]:= plotTotalInfected[meetingCountSims]
Out[∘]=
```



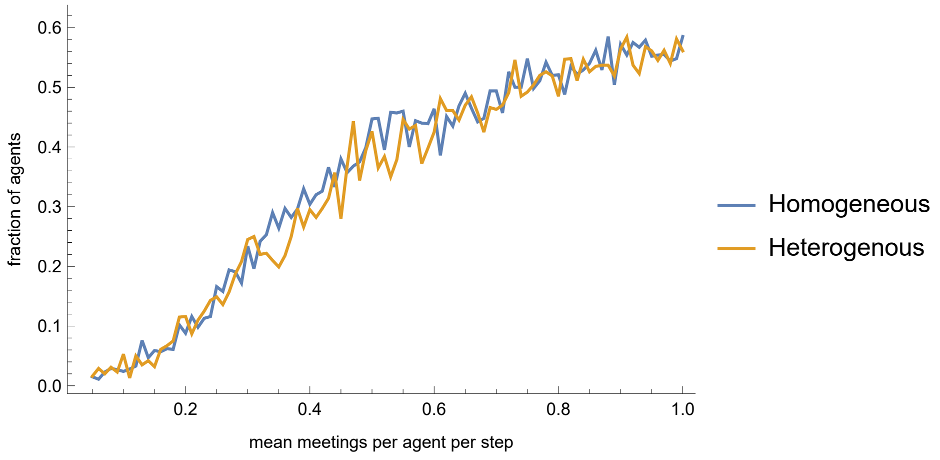
The Pareto distribution produced similar results to the exponential distribution. Otherwise, the higher the standard deviation, the softer the curve, and the earlier the critical point. In other words, it is on average worse to have some agents meeting lots of people and other meeting nobody than it is to have every agent meeting an average number of people. That is, it is better to have many small meetings than a few large ones.

So far every agent has been the same, with the same distribution determining how many meetings they make. However, we could also model a heterogeneous population by giving different agents different meetingCount distributions.

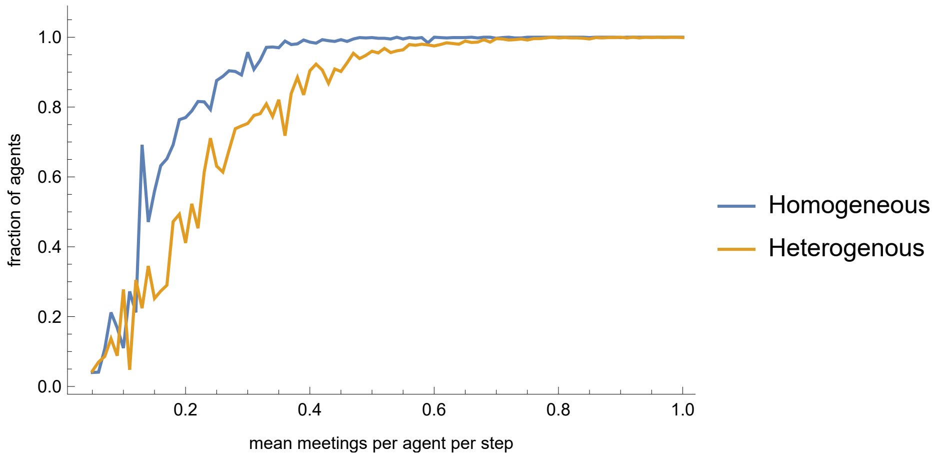
We will simulate homogeneous and heterogenous populations. In the homogeneous population, every agent has a normal distribution that determines the number of meetings they will make at each step. In contrast, every agent in the heterogenous population makes a constant number of meetings at each step, but that constant number is different between agents and modeled by the same normal distribution as in the homogeneous population. In other words, the mean and variance of the number of meetings per agent per step are the same, but in the homogeneous population we sample at each step and in the heterogenous one we only sample once:

```
In[∘]:= meetingCountSims2 = simulateWithMeetingCounts /@ <|
           "Homogeneous" → (NormalDistribution[#, 0.35] &),
           "Heterogenous" → (UniformDistribution[
                ConstantArray[RandomVariate[NormalDistribution[#, 0.35]], 2]] &)
           |>;
```

```
In[∘]:= plotPeakInfected[meetingCountSims2]
Out[∘]=
```



```
In[∘]:= plotTotalInfected[meetingCountSims2]
Out[∘]=
```



Interestingly, these produce slightly different results, with the heterogenous population generally seeing slower disease spread. In the heterogenous population there might be a few outliers who meet very few neighbors and who keep the disease from spreading quickly.

Recovery Times

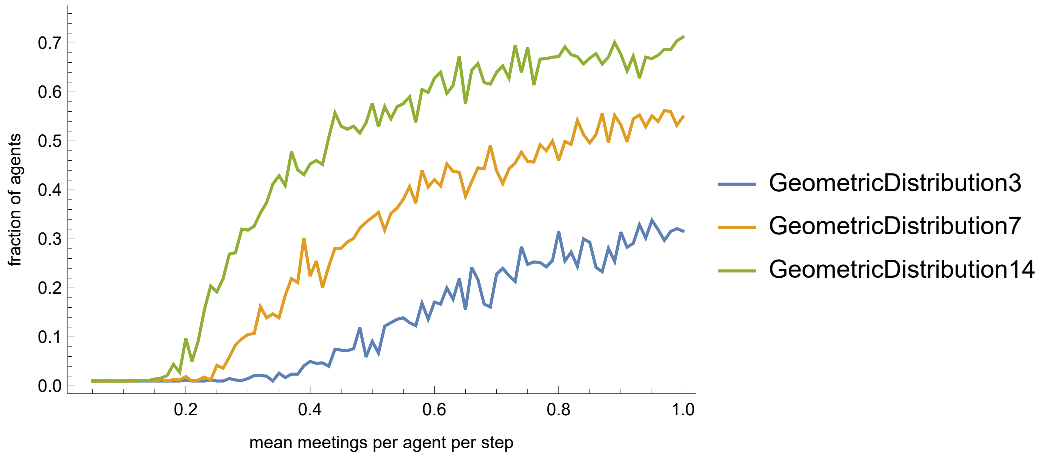
So far we have been using a geometric distribution for recovery times because it makes it possible to generate phase plots. However, actual recovery times are probably modeled by something else (for example, many SIR models assume an exponential distribution).

We can also modify the recovery time distribution to simulate quarantines because a recovered agent no longer transmits, and so a quarantined agent behaves like a recovered one in our model. However, this trick only works when looking at the total fraction of the population infected because the peak infection rate will be hidden because many “infected” people who are in quarantine will be marked as recovered.

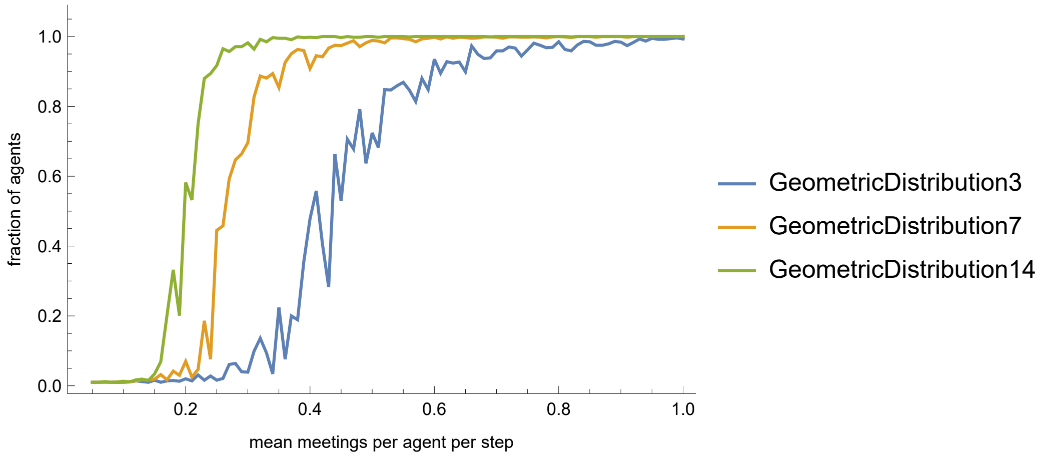
```
In[∘]:= simulateWithRecoveryTime[recoveryTime_] :=
         Association@Table[
           m → simulate[
             agentNeighborhoods[g],
             AssociationMap[ExponentialDistribution[1 / m] &, VertexList[g]],
             recoveryTime,
             10,
             Infinity
            ],
           {m, 0.05, 1, 0.01}
          ]
```

```
In[∘]:= geometricRecoveryTimeSims = simulateWithRecoveryTime /@ <|
            "GeometricDistribution3" → GeometricDistribution[1 / (1 + 3)],
            "GeometricDistribution7" → GeometricDistribution[1 / (1 + 7)],
            "GeometricDistribution14" → GeometricDistribution[1 / (1 + 14)]
            |>;
```

```
In[∘]:= plotPeakInfected[geometricRecoveryTimeSims]
Out[∘]=
```



```
In[•]:= plotTotalInfected[geometricRecoveryTimeSims]
Out[•]=
```

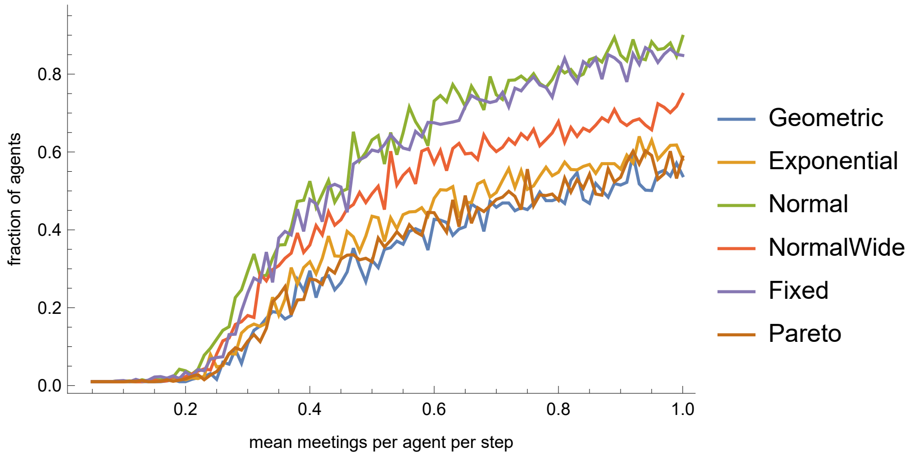


We can see that as the mean recovery time gets shorter, the peak fraction of agents infected decreases (which is not surprising because we are just reducing the mean time that each agent spends infected), but the critical point between when everybody gets infected and nobody does is shifted right. In other words, quarantines increase the amount of interaction required for the disease to spread.

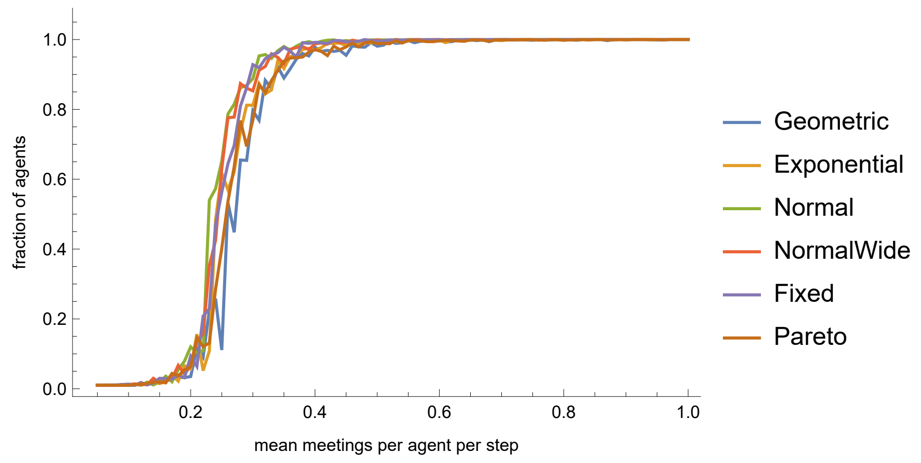
Next we will look at the effect of choosing different distributions. Here we consider a list of plausible distributions all with the same mean:

```
In[•]:= recoveryTimeSims = simulateWithRecoveryTime /@ <|
          "Geometric" → GeometricDistribution[1 / (1 + 7)],
          "Exponential" → ExponentialDistribution[1 / 7],
          "Normal" → NormalDistribution[7, 1],
          "NormalWide" → NormalDistribution[7, 5],
          "Fixed" → UniformDistribution[{7, 7}],
          "Pareto" → (ParetoDistribution[(b - 1) 7, b, 0] /. b → 5)
          |>;
```

```
In[•]:= plotPeakInfected[recoveryTimeSims]
Out[•]=
```



```
In[•]:= plotTotalInfected[recoveryTimeSims]
Out[•]=
```



Here we see that the total fraction infected is not significantly affected by the distribution of recovery times, which is not too surprising. The peak fraction infected, however, is affected by the distribution of recovery times, and that the fixed and normal distributions lead to a higher peak. Interestingly, the normal distribution with a higher standard deviation leads to a lower peak than the one with a lower standard deviation.

Conclusions

The main purpose of this project was to produce this model and investigate its properties. As with many agent-based models, its behavior is complicated and the relationship between the inputs and outputs is often hard to model. Below is a partial list of findings:

- There is a critical point in the amount of interaction which determines whether everybody gets sick or nobody does. This corresponds to the base of the exponent if we think of this in terms of exponential growth.
- The structure of the contact network matters. Even with the same number of vertices, edges, etc, different contact networks produce different results. Barabasi-Albert has very fast spread, Watts-Strogatz had intermediate spread, and the “supergraph” which was a planar graph made up of Watts-Strogatz clusters had slow spread when the clusters were only weakly connected.
- Reducing interaction between communities increases the uncertainty in the outcome, but flattens the curve and reduces the average total number of people infected.
- It is better to have many small meetings than a few large ones.
- Changing the recovery time (which is equivalent to having quarantines in this model) changes the position of the critical point between everybody getting infected and nobody getting infected.
- The heterogeneity of the agents is important, and the distribution determining how much individual agents interact changes the outcomes. That is, the mean rate of interaction is not enough to model disease spread, and the distribution of interaction rates is needed.