A Hybrid Equation- and Agent-Based Model for Efficient Computation of Epidemic Spread

Cooper Doyle, 310275156

November 20, 2019

1 Introduction

Agent-based models capture the intricacies of complex systems by simulating the interactions of a multitude of individual entities. Due to the large number of agents involved in modelling disease outbreaks in major cities or countries, these models usually come with a significant computational cost. In contrast, equation-based models use mean-field approximations to yield sets of differential equations representing the time-evolution of summary statistics of a system. These equations are often analytically tractable, or can be solved using computationally efficient finite-differencing schemes. While agent-based models can accurately represent a system in any state, the mean-field approximations of equation-based models are mostly effective where the central limit theorem applies, that is, where the number of relevant interactions is sufficient for statistical means to be meaningful. In the following, we use a hybrid model that switches from agent-based to equation-based once a threshold of "active" agents is reached. This serves to accelerate the computational time of complex models, and to evaluate the efficacy of mean-field approximations. In particular, we study the spread of an epidemic through a network of US cities linked by flight paths.

2 Agent-based Model

Agent-based models are computer simulations involving multiple entities (the agents) acting and interacting with one another based on their programmed behavior. Agents can be used to represent living cells, animals, individual humans, even entire organizations or abstract entities.

Agent-based models are useful in describing disease transmission involving human transport and local interaction. The joint behaviour of the agents can be very complex and tracking their behaviour requires a disciplined approach. Realistic models typically involve complex transport networks and millions of agents, thus requiring a large amount of computational power and time.

In this model, agents can move between nodes in a network of flight paths connecting airports. interactions are captured through a proximity-based probabilistic effect. At each iteration, agents within a certain distance of one another will have a probability of disease transmission if one of them is infected and the other is susceptible. Agents also have a small probability of migrating to adjacent nodes and, at each iteration, any Infected agent has a probability of removal.

The parameters of the agent-based model are the migration probability, the infection probability for agents in the same city (virality), and the removal probability (resilience).

3 Equation-based Model

These systems can also be represented using equation-based compartmental models that use a mean-field approximation to relate the rate of change of populations in the system to their density. The most famous of these models is the SIR model. It distinguishes between three categories of agent: Susceptible (S), Infected (I) and Removed (R). Susceptible individuals have a nonzero probability of being infected, while Infected individuals have a nonzero probability of becoming Removed and developing an immunity to the disease.

The compartmental SIR model can be reduced to a set of coupled ordinary differential equations like so:

$$\frac{dS}{dt} = -\alpha IS$$

$$\frac{dI}{dt} = \alpha IS - \beta I$$

$$\frac{dR}{dt} = \beta I$$

The constant α represents the rate of infection, and β represents a constant rate of removal. For β we use the removal probability from the agent-based model. The appropriate value of α is determined empirically as half of the transmission probability for individuals at the same node.

4 Threshold Switch

To determine the switching mechanism between equation-based and agent-based models, a switching threshold is implemented. When the proportion of Infected individuals in the agent-based model becomes larger than the threshold, the regime is switched to equation-based. To avoid artefacts from the equation-based model, if the population dips below the threshold during evaluation, the regime is switched back to agent-based, with infected agents selected randomly.

To tune the threshold, results of the hybrid model are compared to the agent-based model. In this way, the agent-based model is treated as the "gold-standard" since it captures the intricacies of human and social interaction. It's hoped that the hybrid model will produce similar results to a purely agent-based model with less computational work.

5 In Literature

Reference [1] attempts to create a highly complex agent-based transport model involving each individual agent selecting a list of plans from memory based on econometric scoring, and local infection interactions. The model is highly intricate and potentially extremely precise, however the computational demands of such a complex algorithm mean that its applicability is reduced. In ref. [1], the model is run on only 10% of the total population.

To try and correct for this, a hybrid model is implemented as described in [2]. Passing from an agent-based regime to an equation-based epidemic model when a threshold is reached allows for significantly faster computational time, and, ideally, little loss in accuracy.

Rather than implementing a complex econometric preference-based model for social transport as in [1], we adapt a probabilistic transport model to capture individual agents' flight routes.

6 Implementation

The agent-based model is implemented in Mesa, a framework for agent-based modelling in Python 3.7. The library used an object-oriented modular approach, in which the modelling, analysis and visualisation components are kept separate but intended to work together. The agent and model objects are defined within the Mesa modelling module, along with the proximity-based probabilistic disease transmission process and the transport of individual agents across the network of air routes. Extraction of macro-statistics from the model is accomplished using the Data Collector from the Mesa analysis module.

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Flight path data for the US is taken from the Python Cookbook Github and each individual airport is added as a node to a Networkx graph. Flight paths are represented by edges in the graph, connecting each airport. Naively, we omit transmission of disease during flights, and only allow for the possibility of infection in each city. A visualisation of the flight paths is shown in figure 1.

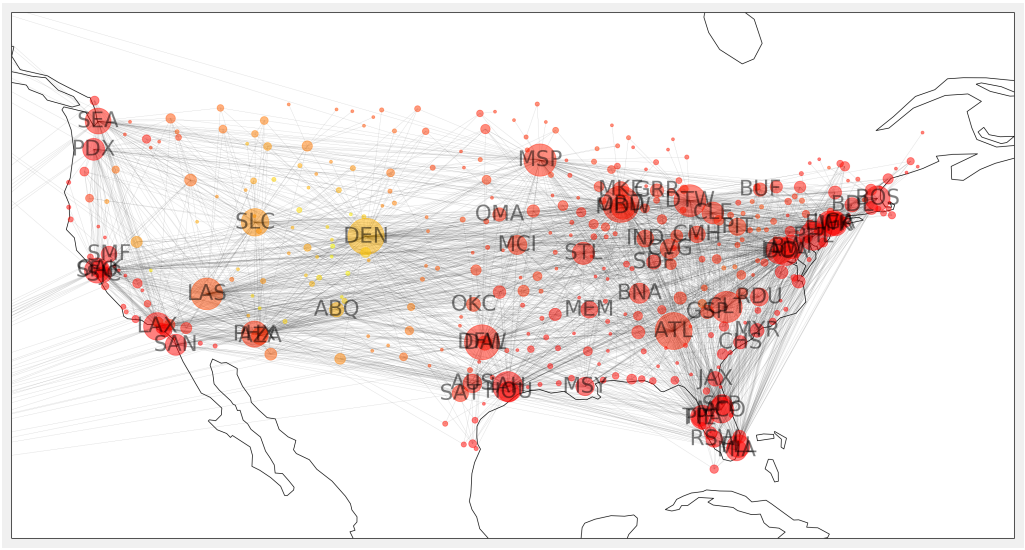

Figure 1: Graph of the NetworkX flight path network

The equation-based SIR model is implemented using Euler's finite differencing method. To minimise computational effort, we make use of the fact that $S + I + R = 1$. The update at each iteration is shown below:

$$S \to S - \alpha * I * S$$
$$R \to R + \beta * I$$
$$I \to 1 - R - S$$

7 Results

To compare the hybrid model to the pure agent-based (standard) model, we first naively set the switching threshold to an I value of 0.5 and vary the transmission probability between individuals within the same city, here referred to as "virality". The next step is to tune the switching threshold by establishing the highest threshold value at which the standard and hybrid models are in good agreement. This also allows us to evaluate the validity of the mean-field approximation used in deriving the SIR model and the finite-differencing scheme that forms the basis of the equation-based approach. In each case, the model is run through 100 iterations, representing time in arbitrary units. We also measure the run-time for each, to

establish the computational efficiency improvement offered by the hybrid model over the pure agent-based approach. In each case, the removal probability (resilience) at each iteration is set to 0.01.

7.1 Varying Transmission Probability

The results for varying transmission probabilities (virality) are shown in figure 2. We see that for a high switching threshold, the hybrid model is in very good agreement with the agent-based model. One interesting observation that is consistent for all transmission probabilities is that even when the models deviate above the switching threshold, the equation-based model is able to correctly predict the turning-point of the curve, and the two models gradually converge beyond that point which generally leads to impressive agreement once the density of infected individuals falls back below the switching threshold.

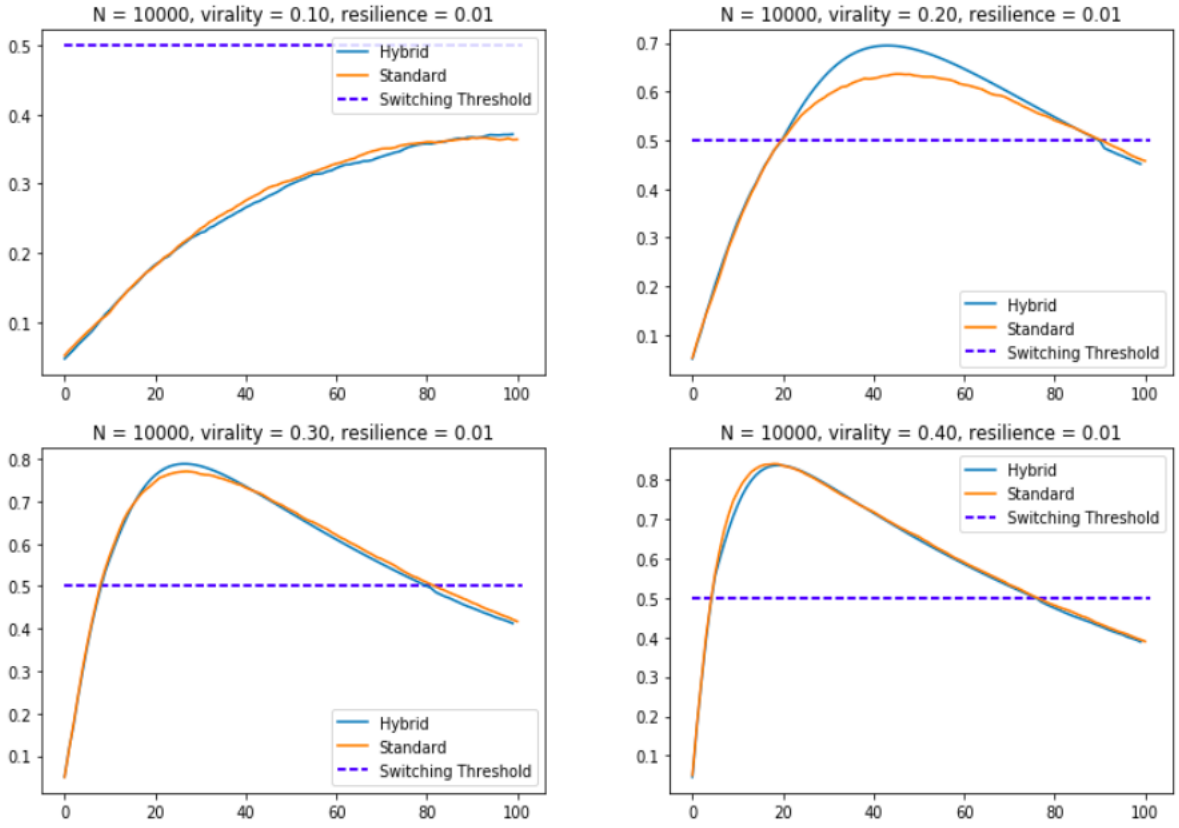

Figure 2: Plots of density of infected individuals (I) over time for various values of virality

Unfortunately the runtimes for the two models do not differ significantly in this case, however this is most likely due to the computational effort involved in implementing the switching mechanism. When the curve dips below the threshold, the agent-based model must be re-initialised with a new distribution of infected and removed entities, which is a computationally demanding process.

7.2 Varying Switching Threshold

In order to tune the switching threshold, the same curves are compared with various threshold levels to identify the highest value at which the models are in good agreement. The resulting plots of I over time are shown in figure 3. Unsurprisingly, larger thresholds tend to produce better models which have a greater

agreement with the pure agent-based model, but offer a reduced computational speedup.

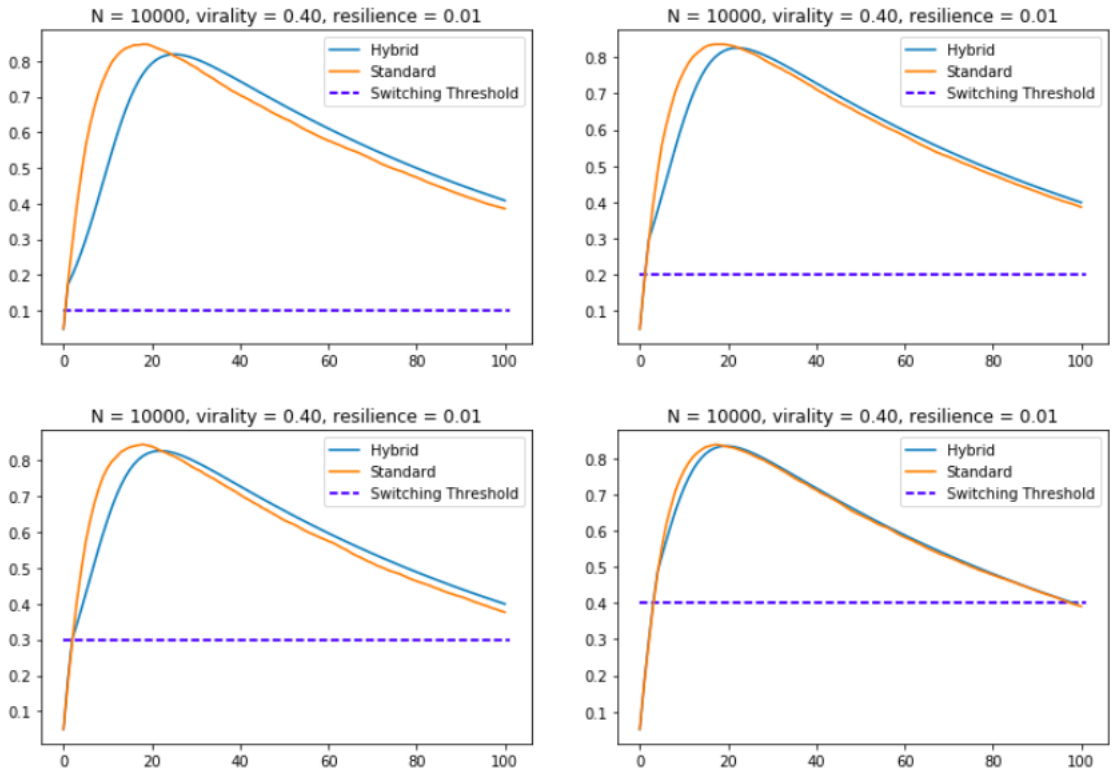

Figure 3: Plots of density of infected individuals (I) over time for various values of switching threshold

In these figures it's clear that for a threshold of 0.3 and below, the models begin to deviate significantly in their predictions. Below this threshold value, the equation-based model no longer reliably identifies the turning-point in the outbreak, and long-term predictions differ slightly. For this particular system, we can identify a density of infected individuals of 0.3 as the approximate point where the mean-field approximation no longer holds and the central limit theorem ceases to apply. In all cases, the hybrid model offers a run-time improvement of approximately 10x when compared to the standard model. This computational speedup is now visible since the models do not dip back below the switching threshold, and thus the issue of added computational time taken in re-initialising the agent-based model is no longer present.

The ideal threshold value is found to be 0.4, offering a runtime of $2.3s$ as opposed to $24s$ for the pure agent-based model. The predicted curves are near identical, as showed in the bottom-right plot of figure 3.

8 Conclusions

The hybrid model is able to replicate the results of a pure agent-based model with astounding accuracy when the switching threshold is carefully tuned. The results show that this results in a computational speed of 10x when compared to a pure agent-based modelling approach. Using the correct model parameters, the threshold can inform of the validity of mean-field approximations used in deriving the differential equations that form the basis of the SIR model for epidemic spread. In this example, we find that the central limit theorem begins to break down at a density of infected individuals of approximately 0.3, where the hybrid model and standard model begin to deviate significantly in their long-term behaviour as well as in their exhibited turning point.

These results are particularly useful for more complex transport models, involving more complex networks, which will typically require a large number of agents (in the thousands or millions). In these cases, a computational speedup of a factor of 10 is very significant.

An interesting next step would be to introduce weighted edges to represent the frequency of flight paths in order to better model inter-city transport.

References

[1] Hackl, Jurgen and Thibaut Dubernet. "Epidemic Spreading in Urban Areas Using Agent-Based Transportation Models." Future Internet 11 (2019): 92.

[2] Bobashev, Georgiy V., D. Michael Goedecke, Feng Yu and Joshua M. Epstein. "A Hybrid Epidemic Model: Combining The Advantages Of Agent-Based And Equation-Based Approaches." 2007 Winter Simulation Conference (2007): 1532-1537.

9 Code

```
import pandas as pd
import networkx as nx

names = ('airline,airline_id,'
         'source,source_id,'
         'dest,dest_id,'
         'codeshare,stops,equipment').split(',')

routes = pd.read_csv(
    'https://github.com/ipython-books/'
    'cookbook-2nd-data/blob/master/'
    'routes.dat?raw=true',
    names=names,
    header=None)

names = ('id,name,city,country,iata,icao,lat,lon,'
         'alt,timezone,dst,tz,type,source').split(',')

airports = pd.read_csv(
    'https://github.com/ipython-books/'
    'cookbook-2nd-data/blob/master/'
    'airports.dat?raw=true',
    header=None,
    names=names,
    index_col=4,
    na_values='\\N')

airports_us = airports[airports['country'] ==
                       'United_States']

routes_us = routes[
```

```
    routes['source'].isin(airports_us.index) &
    routes['dest'].isin(airports_us.index)]

edges = routes_us[['source', 'dest']].values

g = nx.from_edgelist(edges)

# Agent-based model

from mesa                  import Model, Agent
from mesa.space            import MultiGrid, NetworkGrid
from mesa.time             import RandomActivation
from mesa.datacollection   import DataCollector
from random                import random, randrange, randint, choice
import pandas as pd

def compute_infection_rates(model):
    agents_infected = [agent.infected for agent in model.schedule.agents]
    num_infected = 0
    N = model.num_agents
    for infected in agents_infected:
        if infected == True:
            num_infected += 1
    return num_infected/N

def compute_prev_infection_rates(model):
    agents_prev_infected = [agent.prev_infected for agent in model.schedule.agents]
    num_prev_infected = 0
    N = model.num_agents
    for prev_infected in agents_prev_infected:
        if prev_infected == True:
            num_prev_infected += 1
    return num_prev_infected/N


class Person(Agent):
       def __init__(self, unique_id, model, infection_prob, removed_prob = 0, virality = 0
        self.model = model
        self.infection_prob = infection_prob
        self.virality = virality
        self.resilience = resilience
        self.unique_id = unique_id

        self.prev_infected = decision(removed_prob)
        if ~self.prev_infected:
            self.infected = decision(self.infection_prob / (1 - removed_prob))
        else:
            self.infected = False

        self.year = 0

    def move(self):
        """Agent changes position in grid"""
        possible_steps = self.model.grid.get_neighbors(
```

```
            self.pos)
        new_position = choice(possible_steps)
        self.model.grid.move_agent(self, new_position)

    #Determines probability of infection being transmitted
    #from one agent to another
    def transmit(self):
        cellmates = self.model.grid.get_cell_list_contents([self.pos])
        #determies number of agents in the same space

        if len(cellmates) > 1: #if there's more than 1 agent in cell
            for other in cellmates:
                if other.infected == True and self.prev_infected == False:
                    self.infected = decision(self.virality)

                else:
                    pass


    def step(self):

        self.year += 1/52
        self.move()

        if self.infected and decision(self.resilience):
            self.infected = False
            self.prev_infected = True

        if self.infected == False:
            self.transmit()

        else:
            pass


class EpiModel(Model):
    def __init__(self, N, network, infection_prob, removed_prob = 0, virality = 0.5, resili
        self.running = True
        self.num_agents = N
        self.grid = NetworkGrid(network)
        self.schedule = RandomActivation(self)
        self.num_incarcerated = 0

        # Create agents
        for i in range(self.num_agents):

            a = Person(i, self, infection_prob, removed_prob, virality, resilience)
            self.schedule.add(a)
            # Add the agent to a random grid cell
            x = choice(list(g.nodes))
            self.grid.place_agent(a, x)

        self.datacollector = DataCollector(
           model_reporters = {"Infection_Rates": compute_infection_rates, "Removed_Rates":
```

```
            agent_reporters = {"Infected:": lambda a : a.infected})

        self.datacollector.collect(self)

    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)

def decision(probability):
    ''' Given a probability p, P(True) = p,
    P(False) = 1-p '''
    return random() < probability

## Hybrid Model
import time

def hybrid_model(N, infection_prob, virality, resilience, threshold, tot_iter):

    avg_cellmates = N / len(list(g.nodes))
    model = EpiModel(N, g, infection_prob, 0, virality, resilience)
    model_df = pd.DataFrame(columns=['Infection Rates', 'Removed Rates', 'SIR model'])

    alpha = virality/2
    beta = resilience

    n_iter = 0
    start = time.time()
    while n_iter < tot_iter:
        iteration_df = model.datacollector.get_model_vars_dataframe().iloc[-1,:]
        model_df = model_df.append(iteration_df, ignore_index=True)

        I = model_df['Infection Rates'].iloc[-1]
        R = model_df['Removed Rates'].iloc[-1]
        S = 1 - I - R

        while I > threshold and n_iter < tot_iter:
            S -= alpha*I*S
            R += beta*I
            I = 1 - R - S
            model_df = model_df.append({'Infection Rates': I,
                        'Removed Rates':R,
                        'SIR model':True,
                        }, ignore_index=True, sort=False)
            n_iter += 1

        switch = 0
        if model_df['SIR model'].iloc[-1]==True:
            switch_start = time.time()
            model = EpiModel(N, g, I, R, virality, resilience)
            switch_end = time.time()
            switch = switch_end - switch_start

        model.step()
        n_iter += 1
```

```
    stop = time.time()
    runtime = stop - start - switch
    return model_df, runtime

# Standard Model

def standard_model(N, infection_prob, virality, resilience, tot_iter):

    model = EpiModel(N, g, infection_prob, 0, virality, resilience)

    start = time.time()
    for i in range(tot_iter):
        model.step()

    model_df = model.datacollector.get_model_vars_dataframe()

    stop = time.time()
    runtime = stop - start
    return model_df, runtime

# Standard Model

def standard_model(N, infection_prob, virality, resilience, tot_iter):

    model = EpiModel(N, g, infection_prob, 0, virality, resilience)

    start = time.time()
    for i in range(tot_iter):
        model.step()

    model_df = model.datacollector.get_model_vars_dataframe()

    stop = time.time()
    runtime = stop - start
    return model_df, runtime

# Run models
hybrid, runtime_h = hybrid_model(N, infection_prob, virality, resilience, threshold, n_iter
standard, runtime_s = standard_model(N, infection_prob, virality, resilience, n_iter)

print(runtime_h, runtime_s)

# Run models
hybrid, runtime_h = hybrid_model(N, infection_prob, virality, resilience, threshold, n_iter
standard, runtime_s = standard_model(N, infection_prob, virality, resilience, n_iter)

print(runtime_h, runtime_s)
```