Thesis for the degree
Master of Science

עבודת גמר (תזה) לתואר
מוסמך למדעים

Submitted to the Scientific Council of the
Weizmann Institute of Science
Rehovot, Israel

מוגשת למועצה המדעית של
מכון ויצמן למדע
רחובות, ישראל

By
Adi Shindler

מאת
עדי שינדלר

מידול התפשטות מחלות
Disease Spread Modeling

Advisors:
Prof. Uriel Feige
Prof. Moni Naor

מנחים:
פרופ. אוריאל פייגה
פרופ. מוני נאור

Month and Year
09/09/2021

חודש ושנה עבריים
ג' תשרי התשפ"ב

Abstract

The global outbreak of COVID-19 highlighted the need for robust tools for simulating disease spread in a population. While various tools and frameworks already exist, they often include a very limited subset of features, and make simplifying assumptions, both of which can limit the ability to detect and capture more subtle phenomena. In this work, we present a system which can be used to test intricate hypotheses. Towards this goal, the system includes many parameters and a high number of features, including uncommon ones such as agent (dis)obedience and multiple diseases (or strains). In addition, several "cost functions" that depend on the various intervention methods exist in order to evaluate the effectiveness of these methods with respect to their cost. Finally, the program provides functionality to view and modify (the intervention methods used in) simulation in real (simulation) time, further enhancing the ability to evaluate the effectiveness of various methods.

1 Introduction

1.1 Overview

Mathematical models of infectious diseases generally aim to provide a way to predict the spread of a disease or the effectiveness of different intervention methods. Many such models exist, varying greatly in their assumptions, aims, trade-offs, and complexity. In this work, we do not deal with the task of predicting the future course of a specific outbreak, and instead focus on comparing and evaluating the effectiveness of different mitigation methods and strategies when combating infectious diseases. We provide a theoretical model as well as a working program which runs simulations (with respect to this model) in order to determine the effects of various mitigation tools, taking into account both their effectiveness and their cost.

1.1.1 The Model

We wish to model the spread of a disease over a population of individuals, in which (we assume) people infect one another directly. Normally, this is done via a *social graph/network* i.e., a graph between *people*, or *agents* (we will use these terms interchangeably). An edge between two agents models the interaction between them. Our model is different, in that it employs a bipartite graph of *agents* and *activities* instead of the usual *agents*-only graph. The rationale behind this modeling choice is as follows. Rather than modeling the direct interaction (and thus infection) between people, we model their interaction as something that happened at a certain *location*, or *activity*, such as a classroom or a bar. For a more in-depth explanation of this modeling choice and its assumptions, see section 3.1.

Expanding on the above, our model considers a family of weighted bipartite graphs between *People* and *Activities* for its simulations. The weight on an edge between a person p and an activity a can be thought of as the time (number of hours) p spends at activity a (within a day, say). That is, whereas conventionally a *people-to-people* network is used (or no network is used at all), we simulate on a *people-to-activities* network. An infected person who visits an activity may infect other persons who visit the activity at the same time. In our people-to-activities network, this is modelled by the infected person infecting the activity (with some probability, and for a random duration that may correspond to the random length of the visit of the infected person), and then the activity might infect other visitors (with some probability). Our model is agent-based and heterogeneous — it allows every node to have its own set of neighbors (and accompanying weights) and various other parameters (such as obedience to lockdown measures). That is, we are not merely interested in simulating and obtaining "global information" such as how many people become sick, but rather all of the various complexities that arise throughout the simulation on the agent level.

1.1.2 The Simulation

Simulation occurs in continuous-time (namely, we do not skip ahead to the next day, but rather events can happen at arbitrary times, allowing for more complex interactions) using a modified version of the Gillespie algorithm ([6] [7] [5]) to epidemics on networks. Our basic simulation procedure is based on an implementation of this algorithm given in [11] (see also [9]).

In addition to this "basic" simulation procedure, our system provides many additional features which can be enabled. We briefly describe a subset of these features, and leave a more in-depth explanation for section 3.2.2.

Multiple Diseases. Enabling an arbitrary number of diseases to be present in the system at once (and affect one another) is a key and non-trivial feature of our system. In light of current events, it seems natural to consider a variant where several diseases (or strains of a single virus) spread over the network, where each virus has its own set of parameters (such as how deadly it is or how quickly it spreads). Additionally, one can consider direct dependencies between the two diseases: perhaps being infected with one strain of the virus prevents one from becoming infected with another strain in the future (acting as a "natural vaccine" of sorts).

Vaccination Protocols. People can be vaccinated against diseases that are present in the network. This can be done at different (and usually periodically recurring, e.g. daily) time-points. Additionally, a *vaccination policy* can be used in deciding who gets priority. Three such policies are given: age-based (elderly-first), degree-based (the higher the weighted degree of the node in the graph, the higher their priority), or uniformly at random. Each vaccine is associated with its effectiveness in reducing susceptibility for every disease and strain in the system, not simply against a single strain of a disease.

Lockdown Measures give the ability to control and modify interaction levels between people and activities at any point in time as a function of current events, such as the percentage of ICU beds that are in use.

Agent Behavior - Lockdown Measures (dis)obedience allow agents to (individually) disregard lockdown measures and behave "normally". This can be used, for example, to model people becoming tired of too many lockdowns, or if we wish to model certain demographics being inherently more or less obedient.

Contact Tracing is a key tool when attempting to fight the spread of a disease. A contact of an individual is simply someone they have interacted with. Contact Tracing is done by isolating, and possibly testing, contacts of people who have tested positive for the disease. The idea behind it is to catch and isolate recently exposed contacts, especially ones that still may be unaware of their infection (as they may not show symptoms yet). We will use the terms "isolate" and "quarantine" interchangeably to denote a situation in which a person should isolate.

Disease Testing. People can be tested for each disease, and can take action based on the test result. For example, an individual who tests positively should quarantine. Testing can occur due to one (or a combination of) the following: contact tracing, showing symptoms, or as a part of a *Population-wide testing policy*. Such a policy is responsible for testing a random subset of the population (of a certain size) every so often (for example, it can be used to test 1% of the population on a weekly basis).

Hospitalizations. Critically infected individuals may require special equipment (such as ICU beds). If too many people become critically ill at once and no special equipment is available, their rate of death increases significantly.

Open/Closed System. Allows spontaneously infecting randomly selected susceptible individuals at arbitrary times, in order to model "the real world" in a more accurate way (as otherwise harsh lockdown measures may seem sufficient to eradicate any disease). Additionally, allows introducing new variants and diseases mid-way through a simulation.

Various Parameters and Rates. In addition to these features, our system has numerous parameters that can be modified. For example, one can control various parameters regarding the network and its topology, disease related parameters (such as infectiousness, symptoms and recovery rates), and many more. A full list of the parameters can be seen in the program's parameters file, whereas the GUI contains a limited subset of all parameters.

Pausers. A `Pauser` allows the advanced user to compute arbitrary functions (which they can implement themselves) that will be evaluated throughout the simulation (at times of their choosing). This gives a simple way to further extend the model to various needs.

1.1.3 The Program and GUI

Our program[1] consists of two sub-programs. A *backend* which is in charge of performing the simulations and can be used by itself, and a graphical user interface (GUI) of the backend. The GUI serves two purposes. The first purpose is to provide a user-friendly way to use the backend for those who prefer simplicity at the expense of precise control over the large number of available parameters. The second purpose is to provide live, real-time plots of the simulation, while additionally allowing the user the ability to pause the simulation, modify some of the parameters, and continue the simulation with the new parameters in place. Equipped with some "cost-functions" which are shown in real-time, this allows one to try to optimize their strategy (minimize the number of dead while keeping costs below a certain threshold, for example).

1.1.4 Related Work

Based on earlier works by Ross [18, 17, 19], the SIR model (**S**usceptible, **I**nfectious, **R**ecovered/**R**emoved) was introduced by Kermack and McKendrick [12] as a part of a sequence of three papers [12, 13, 14] describing basic compartmental models for the purpose of modeling transmissions of diseases. The SIR model, while fairly basic, is arguably the most influential epidemiological model, of which many variations exist: the *SIS* (**S**usceptible, **I**nfectious, **S**usceptible) model introduces the notion of re-infections, whereas the *SEIR* model adds the additional *Exposed* compartment for individuals who are infected but not yet infecting. Compartmental models suffer from several limitations due to their simplicity, and different models attempting to overcome these shortcomings exist. Such approaches include introducing non-homogeneity, agent-based modeling, behavioral considerations, and others.

Several other packages for disease simulation exist. For example, *EoN* [11] is a Python package providing tools to simulate the spread of a disease in SIS and SIR networks models, serving as the starting point for our implementation. Similarly, *SEIR+* [20] is a Python package implementing generalized SEIR models to study the effects of various intervention methods, such as testing, contact tracing, social distancing, and isolation in the single-disease case (while not accounting for the associated costs). *DISimS* [4] is an interactive modeling environment, which, similarly to us, allows the user the ability to pause and continue simulations. They consider a somewhat different set of intervention methods from the ones we do. Additionally, they do not put any emphasis on multiple diseases or economic considerations.

[1] The code of which, as well as a Windows executable of the GUI, can be found here: https://github.com/adi-shindler/disease_modeling

EpiSimdemics [2] simulate the spread of a disease in large and realistic social networks, where the underlying network is a bipartite graph between people and activities (as in our model). Their focus is on scalability and detailed, realistic social networks, rather than possible intervention methods.

In light of the recent (and currently ongoing) COVID19 pandemic, models involving vaccinations and multiple diseases (or strains) may be of particular interest, especially since these features are missing from the vast majority of simulation tools available today. Similarly to our model, [16] provide a compartmental model allowing for several strains and vaccinations. Their model operates under the assumption that once an individual is vaccinated or recovered from one strain, they are immune to all other strains of the disease. We do not operate under such an assumption and in fact generalize to complex relationships between different strains of diseases and different diseases altogether in our model (see *Multiple Diseases* in section 3.2.2).

Another point that was made by recent events is the high importance of taking into consideration, and further investigating, economic costs of combating a pandemic. Many works were done in this area, such as [10, 1, 21]. Similarly, metrics other than economic ones may be considered. For example, the effects of an epidemic on the psychological well-being [1] of the population, both caused either directly (i.e., grief) as well as due to the effects of intervention methods (loss of income, extreme isolation, etc).

In their work, Kempe, Kleinberg, and Tardos [8] considered simple models similar to the one we discuss in section 3.1.1. They showed, among other things, that in certain models of spread of disease, the (expected) total number of infected persons is a submodular function of the set of initially infected individuals. Such results need not hold once society reacts to the disease, e.g., via testing, lockdown measures, contact tracing, and so are less directly applicable in our case.

1.1.5 Roadmap

The rest of the document is structured as follows. Section 2 reviews some of the necessary background in disease modeling. Section 3 describes our theoretical graph model and our simulation procedure in practice, as well as the various features that exist within the system. Afterwards, section 4 reviews and demonstrates the way the program (both backend and GUI) should be used while providing several examples. Finally, the appendix provides a more in-depth overview of many implementation details.

1.1.6 Acknowledgments

I would like to thank Prof. Uriel Feige and Prof. Moni Naor for their insightful guidance and for always being eager to help. I found their deep knowledge and comments invaluable, and truly enjoyed our conversations, whether research-related or just casual chats.

I would also like to thank Jonathan Simon for his significant support and encouragement throughout the process.

2 Preliminaries

In this section, we review some of the methods used to model the spread of a disease. We begin by noting that a trade-off certainly exists when considering a model's complexity; on the one hand, it is obvious that a model that is too simple might not be of great use, since it will fail to capture many of the phenomena that take place in the real world. On the other hand, the more complex a model becomes, the less simple it is to *explain* (gain insight from): it becomes harder to pinpoint the exact relationships of causes and effects within the model, due to the sheer number of moving parts and complex relations. While such complex models are more difficult (if not outright impossible) to analyze or solve analytically, their practical value can be fairly high.

2.1 Compartmental Models

In a compartmental model, each individual is associated with a state (such as **S**usceptible, **I**nfectious, and **R**ecovered), which may change as a function of time. We are interested in the number of individuals of each state (the size of the corresponding compartment) at each time.

One of the most well known compartmental models is the deterministic *SIR* [12] model. It consists of the three compartments mentioned above. Each compartment defines a function indicating the number of people within it at any given time. That is, $S(t)$, $I(t)$, and $R(t)$ represent the number of susceptible, infectious, and recovered individuals at time t, respectively. In addition, the model consists of various transition rates between the possible states. For example, letting γ denote the recovery rate, we have $\frac{dR}{dt} = \gamma I$ (a γ fraction of infected people recover). These models can be formulated using ordinary differential equations (where the variables are the sizes of the various compartments).

In contrast, in the *stochastic SIR* model, the number of individuals within each compartment at any given time is a random variable. Suppose that individuals all interact with one another (a complete-graph as the network), infection between an infected and a susceptible individual happens at rate α, and an infected individual recovers at rate γ (that is, the time until infection/recovery distributes according to an exponential distribution with the corresponding rate). Then, the epidemic can be treated as a continuous-time Markov-chain, where the transitions are given by

$$(S, I, R) \xrightarrow{\alpha SI} (S-1, I+1, R)$$

and

$$(S, I, R) \xrightarrow{\gamma I} (S, I-1, R+1)$$

2.2 Network-based Models

The simplicity of the aforementioned models comes with several drawbacks. For example, it is clear that treating all individuals in the same way (having identical infection/recovery rates), or assuming they all interact with one another, can be fairly limiting assumptions. For example, the homogeneous mixing assumption falls short due to the observed fact that a relatively large fraction of infections are done by "superspreaders". Introducing heterogeneity by allowing an arbitrary graph to serve as the network can be a step in the right direction. In such a graph, the vertices (generally) represent individuals and the edges (and their weights) represent a connection between two individuals. For example, in an unweighted graph with infection rate α, a susceptible node v with k infected neighbors becomes infected according to a Poisson process with rate αk. Similarly, in a weighted network where the weight of an edge corresponds to the infection-rate between two individuals, v becomes infected with rate $\sum\limits_{u \in \Gamma(v)} w(u, v)$, since the sum of independent Poisson processes

is itself a Poisson process whose rate is the sum of rates of these processes.

An efficient way to simulate disease spread over a network is the Gillespie algorithm ([6] [7] [5]), which generates a statistically correct trajectory of a stochastic process. We now give a high level overview of the algorithm in the context of disease modeling.

The algorithm begins by calculating the time for the next event to occur, choosing/calculating the actual event to occur at this time, re-calculating all the rates within the system that could be affected by this event happening, and repeating this process until no more events are due (or sufficient time has passed). In slightly more detail, whenever an event occurs, every node that is affected by this event modifies its state and recalculates its *rate*, a scalar which determines the time until the next event relating to this node occurs. The algorithm then determines the next time at which an event will occur (using the rates of all nodes in the system as a combined rate for an exponential random variable), jumps to that time, and the process repeats. A more in-depth explanation of our simulation process appears in section 3.2.

3 Our Model

3.1 The Graph

As previously explained, we simulate the disease spread over a bipartite Activity-People graph, where nodes in our graphs are either agents (people) or activities (locations). The weight of an edge between a person and an activity is used as a basic measure of their likelihood to interact (although in the simulations it is further combined with other things such as the lockdown measures at the time of (potential) interaction).

Let us revisit the choice of graph for our model, as discussed previously in section 1.1.1. We remind the reader that we use this bipartite graph to model *agent-to-agent* interactions by giving a special meaning to the location at which each possible interaction occurs. In doing so, as the graph is bipartite, all interactions between agents go through an activity. Therefore, rather than having agents infect one another, for example, the infection process occurs through shared activities. That is, agents infect activities, which in turn remain infected for some brief period of time, during which they may infect agents which are connected to them in the graph, and so on. With small enough granularity (i.e., with a sufficiently large number of activities), no limitations arise. However, for computational reasons, we often limit the number of activities in our graphs. We expand upon the problems that may arise due to this in section 3.2 and in the contact-tracing paragraph of section 3.2.2.

3.1.1 An Alternative Graph

A natural question is why not simplify the simulation process by considering a new "agents-only" graph. More specifically, construct a new (agents only) graph, where for each two agents, their new edge-weight is a function accumulating the weights of their previous shared neighborhood, and operate on this graph instead (for instance, sum of product of weights, representing an approximation for shared visiting times). This has several benefits, including potential speedup during the simulations. However, this model loses some information and therefore carries certain drawbacks. To illustrate this, consider the example given in figure 1.

This graph consists of a single activity and four agents, two of which are infected. The infected agents visit the activity with a very low probability (say, 1/100) whereas the healthy agents visit with a very high one (say, probability 1). Assuming that both healthy agents get infected on the same day, what can we say about who infected whom? In the new people-to-people graph, both events (one sick agent infected both healthy ones, and each healthy agent got infected by a different sick agent) happen with probability 1/2. In contrast, in a people-activities graph, it is almost certainly not the case that both sick agents visited the activity that day (due to their low probabilities), and so it's almost certainly the case that a single sick agent infected both healthy ones. As illustrated in the example above, there exist correlations between people getting infected via the same activity in the people-activities graph, which may be lost if we consider a people-to-people graph instead.

One can also compare the "infectiousness" (some measure of the spread of the disease) of the two models. Consider, for example, the bipartite graph and the "equivalent" people-to-people graph shown in figure 2. For simplicity's sake, we use the basic discrete-time model in which nodes remain infected (and asymptomatic) for a single day, after which they recover. In the bipartite graph, with probability roughly ε, the activity becomes infected, which results in $P2$ and $P3$ becoming infected themselves on the second day.

In contrast, in the people-to-people graph, each one of P_2 and P_3 get infected with probability roughly ε. Since the edge weight between P_2 and P_3 is essentially 1, infecting one of them will result in both getting infected. Therefore, with probability roughly 2ε, both P_1 and P_2 will be infected in this model by the third

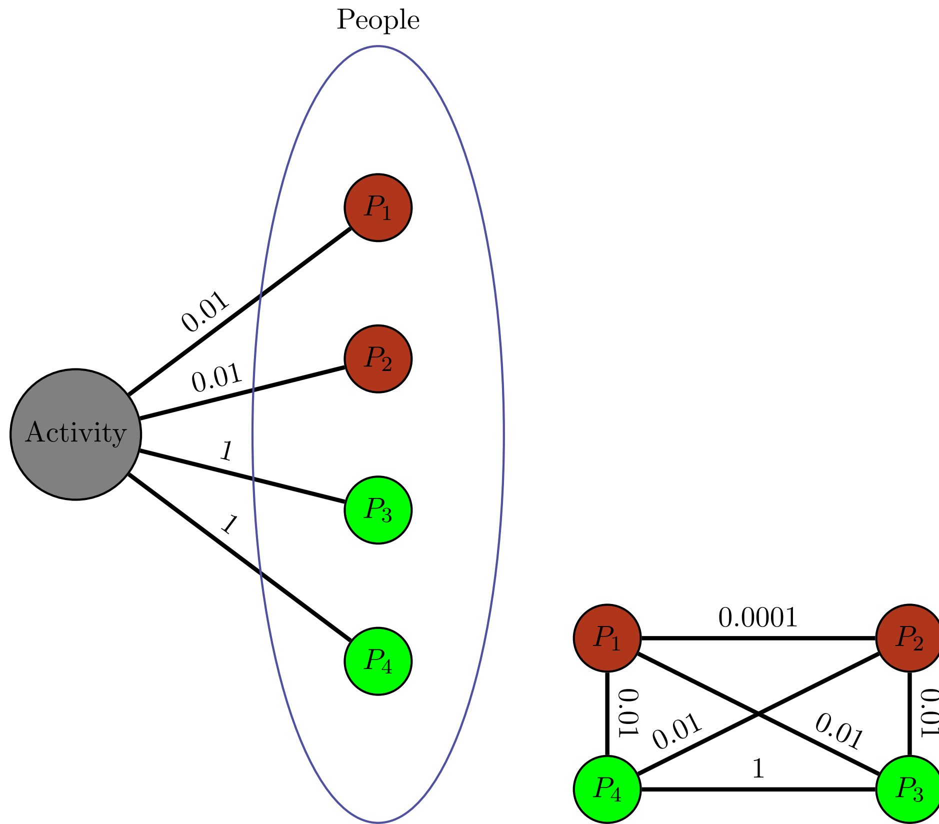

Figure 1: An example of a bipartite graph (and the "equivalent" people-to-people graph) demonstrating the loss of information that could result when using the people-to-people graph instead of the bipartite one. The states of the various nodes on the first day is shown. That is, on the first day, P_1 and P_2 are infectious (and asymptomatic), whereas P_3 and P_4 are susceptible.

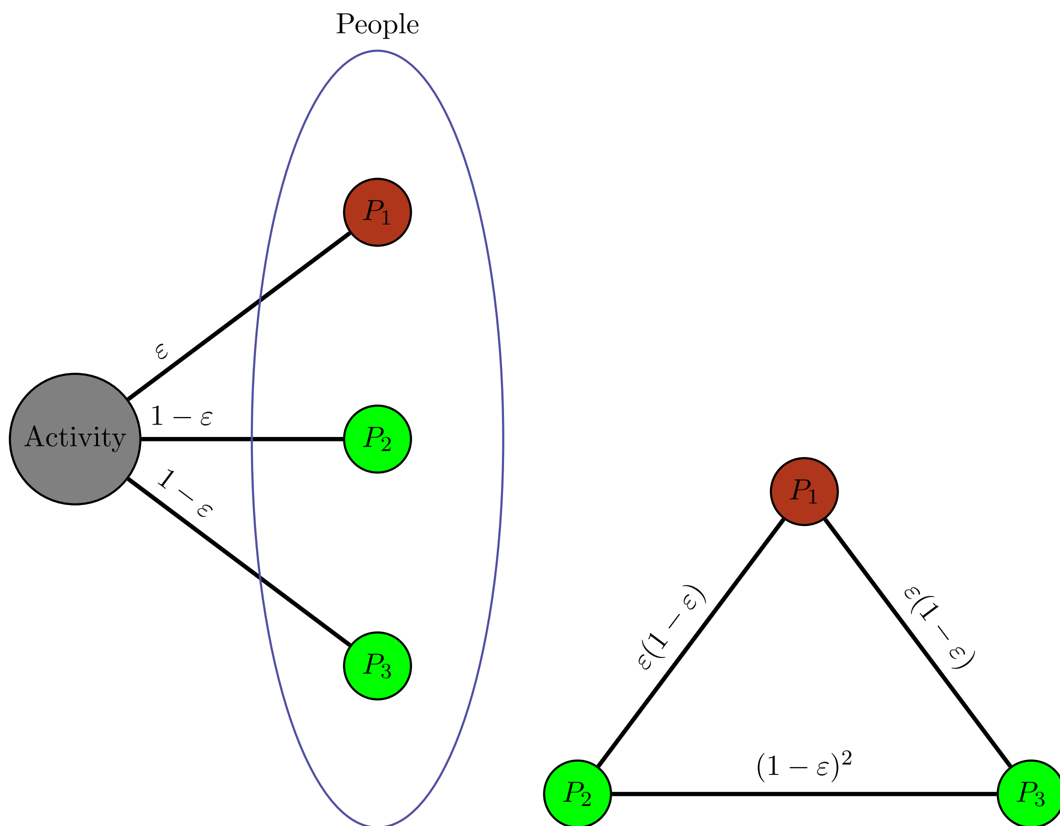

Figure 2: An example demonstrating a scenario in which the bipartite graph (left) is less infectious than the "equivalent" people-to-people graph (right). The statuses of the various nodes on the first day are shown.

day. That is, if we consider the spread of the disease after enough time has passed, in the bipartite model everyone eventually becomes infected with probability ε, while in the people-to-people model this happens with probability 2ε. The above example illustrates a scenario in which the people-to-people model is much more infectious than the bipartite model.

An interesting question is whether one can give an example demonstrating the opposite, i.e., one in which the bipartite model is more infectious, and if not, whether one of the models "dominates" the other in a certain way. For instance, is it the case that for every bipartite graph and the "equivalent" people-to-people graph, under certain assumptions, the expected number of infected people (in total) is larger in the people-to-people graph when compared to the bipartite graph? We remark that even though this may be an interesting theoretical question, it does not seem that a positive answer will be of much relevance to our work, because we allow society to react in various ways to the disease (e.g., via testing, lockdown measures, contact tracing), and results are unlikely to extend to this more complex setting.

3.1.2 Graph Generation

We now describe the way such graphs are constructed in our graph-generation module. We emphasize that any bipartite Activity-People graph can be used in our simulations, not just ones that are the product of our graph generation process. Each node is associated with various attributes. For example, each activity has an "activity kind" (household, gym, etc.), whereas people may be assigned ages and "social types". From these (and other assigned attributes), other attributes could be inferred; an activity's kind affects its "danger-level" – how easy it is to become infected in such a location, whereas age affects the distribution of activities an agent is connected to (of what kinds and how many), or how likely it is to become critically ill if infected with a disease. In addition, every node has a *geographical location* on a discrete 2-dimensional grid. This physical location is taken into account when deciding on the connectivity of the graph – nodes that are geographically closer are more likely to be connected via an edge than those that are farther apart. While every activity is initialized with a (uniformly random) location on this grid, a person's location is that of their (uniquely) assigned household.

Connectivity in the graph is determined as follows. We first decide, for each person and each activity kind, how many activities of this kind the person should (attempt to) connect to. This decision is based on various factors of both the person and the `Activity Kind`. For example, A person `Social Type`'s and age influence the amount (and kinds) of activities a person partakes in; young kids tend to be more connected to schools, whereas older people are more likely to visit bars. Once we know, for each person, how many activities they want to be involved in, we turn to deciding on *which specific* activities (of each kind) to connect the person to. This decision is done probabilistically, with probability (polynomially) inversely related to the ℓ_1 (Manhattan) distance between the two nodes. A `KD-Tree` data-structure can be (and is) used to speed up the processes in order to skip nodes outside a large enough radius, for which the probabilities become sufficiently small. Finally, an activity may reject a person due to its `Max Capacity` attribute.

3.2 The Simulation Procedure

3.2.1 The Basic Variant

We now examine more carefully the way (the bare bones variant of) the simulation procedure is implemented (for a more in-depth explanation of this algorithm (for the usual non-bipartite network), see [9]).

Broadly speaking, people visit activities, possibly infecting (or getting infected by) these activities in the process. Once an activity is infected, it remains infected for a brief period of time, during which it may infect people visiting it. During the simulation, each node is associated with a (time-dependent) *state*, such

as 'Susceptible", "Infected", and "Recovered", indicating the current state of the node with respect to the spreading disease(s). This state changes as events involving this node (or its neighbors) occur. Initially all nodes are susceptible, except a small number of nodes that are exposed to the virus (or viruses, if there are several). These nodes may be chosen based on different policies: their age, in-graph degree, or uniformly at random from the agent population (which is the default policy). Additional policies may be implemented (which, while fairly straightforward, requires some basic programming knowledge). For instance, one might wish to infect people initially based on a more complex decision rule (perhaps their geographical location).

Each node is then (and repeatedly throughout the simulation) assigned an exponential random variable "rate" (which is its rate of changing state). This rate is calculated by calculating separate rates for every possible (relevant) state transition, and summing them to a single rate. For example, an agent would calculate its rate to become infected (by summing up the weights on edges connected to infected neighbors), its rate to recover (if the agent is sick), and so forth. It would then sum the various rates into a single one, which corresponds to *some* event happening to it. The system then sums all of these (combined) rates to obtain the delay until the next time-step of the simulation. The delay is sampled from an exponential distribution whose rate is the sum of rates of all nodes. The event to occur (after the delay time had been skipped) is chosen by sampling a node with probability proportional to its rate, and then sampling which event happened to this node with probability proportional to its individual rates. Once the chosen node's state is modified accordingly, the rates of nodes that could be affected by this change must be updated (these nodes are nearly always simply the node's neighbours in the graph). Their new rates replace the old ones, and the process repeats until no new events are due (or a predefined amount of time, say two years, has passed). We highlight the fact that this simulation procedure operates under the assumption that inter-event times can be reasonably modeled using exponential random variables, although extensions that attempt to overcome this assumption exist [3].

Let us delve deeper into the way nodes' states are represented. We remind the reader that during the simulation every node has a state, where typically compartments (such as "susceptible", "infected", and "recovered") are used to represent these states. In contrast, our system is much more complex, and using simple, "manual" compartments can become unsustainable very quickly. For example, if one wishes to allow two diseases to spread on the network, should one really have to endure a possibly quadratic increase in the number of (manually constructed) compartments? This increase quickly grows as the number of diseases and features within the system grows, and so things can become quite cumbersome very quickly. Therefore, in order to allow greater flexibility (at the cost of more computational resources), we use what can be described as an attribute-based model, where each state is associated with various attributes rather than fixed compartments. For example, a person's state may consist of attributes corresponding to its current (and possibly previous) state of infectiousness, whether they have/had symptoms, current hospitalization status, disease test results, and various others. These attributes can be stored and later on extracted in order to calculate various statistics on the population at large, specific demographics of interest, and individuals in particular. The benefit of such a model is that we do not store and keep track of all possible combinations of attributes, but only those which arise in our simulations, which is a much smaller subset. Similarly, an activity's state is either "closed" (after becoming alerted to the presence of an infected individual) for a brief period of time, or susceptible/infected (unknowingly) for each disease. A more in-depth description of these states is given in section 5.3.1.

Several limitations exist when performing simulations in such a manner. One of which is that, due to there being no concept of explicit "interaction" (since interactions are hidden behind the various rates in our system), events simply happen because they had "sufficient reason to". To illustrate, consider a node becoming infected. It did not do so because it interacted with a particular infecting node, but rather because enough of its infected neighbors were likely enough to interact with it (i.e., the sum of their edge weights

was "high"), and so its "becoming-infected" rate was high enough to become infected by the stochastic process. As we do not directly simulate explicit interactions, some modeling issues may arise. Consider, for example, a "Concert" activity in real-life. Such an activity occurs very infrequently, but when it does happen, many people attend it at once. Now consider a "Concert" activity in the bipartite-graph. It is connected to various agents by weighted edges, where the weight on an edge takes into account various factors (such as the typical "visiting likelihood" for an activity of type concert, which is very low). As such, in expectation, people may infect/become infected by this activity in a "correct manner", even though some kind of "temporal correlations" may be lost. A possible workaround for issues of this nature (which we do not implement) would be to enable/disable edges on a day-to-day basis. For example, all edges connected to a Concert activity are "disabled" by default. Every day, with certain probability, the concert activity (and all its connected edges) becomes "active". If we increase the edge-weights to account for this change, these temporal correlations may be obtained.

3.2.2 Additional Features

In this section, we provide some more in-depth explanations for some of the features that were mentioned in the introductory section.

Multiple Diseases. A key feature of our system is that it allows the modeling of multiple viruses rather than a single one (a small number of agents are infected with each virus in the beginning of the simulation). Each virus has its own set of parameters such as its infectiousness, how likely symptoms are to develop, its deadliness, recovery rates, and more. This feature can be used in order to model the way two (seemingly unrelated, at least directly) diseases such as COVID19 and influenza influence one another (more COVID cases would result in more social distancing, and fewer influenza cases).

One could consider other, more complex, scenarios. Suppose one wishes to model a scenario in which becoming infected with one disease can affect the likelihood of catching another. For example, we may wish to model a scenario in which there are two strains of a disease, where each one acts as a "natural vaccine" of the other strain. If one of those strains were to be much more deadly, and the other much more infectious, perhaps it is not in our best interest to social distance as much? To model and analyze the arising complexities, a "disease matrix" parameter, which specifies the way viruses (and vaccines) interact with one another, can be supplied (see figure 3). The (i, j)th cell of this matrix indicates the susceptibility (a non-negative, possibly non-integer, number) of the node to disease j given that it was previously infected by disease i. For example, the all-zero matrix corresponds to the scenario in which every disease is a "natural vaccine" of every other disease, whereas the all-one matrix allows individual to catch (and possibly re-catch) diseases regardless of their previous infections. Evidently, this matrix can be used to model complex disease dependencies and relationships, and it has the additional benefit of allowing the modeling of diseases that may be caught repeatedly. We highlight the fact that any non-negative value can be used in this matrix, including non-integers and values greater than 1 (which would imply greater susceptibility). Additionally, since vaccines are internally implemented as diseases in our system, this matrix is in control of their pair-wise interactions as well. In reality, we row-stack to this matrix an additional matrix corresponding to the vaccines-disease relationship, which we actually refer to as the "disease matrix". See figure 6 for an actual representation of the combined "disease matrix" in our GUI.

Lockdown Measures. One can think of the spread of a disease as a game between the disease and society. So far we have mostly focused on the way the disease interacts and spreads over our network, and now we turn our focus to modeling the way society combats the disease. The two primary methods are vaccinations and social distancing, typically implemented (and enforced) via regulations and restrictions.

$$\begin{array}{cccc} \textbf{Susceptibility} & \texttt{C.19} & \texttt{Flu} & \textbf{given...} \\ \textbf{to...} & \left[\begin{array}{c} 0 \\ 1 \end{array}\right. & \left.\begin{array}{c} 1 \\ 1 \end{array}\right] & \begin{array}{c} \texttt{C.19} \\ \texttt{Flu} \end{array} \end{array}$$

(a) An example (excluding vaccines) of the disease matrix for COVID19 and Influenza. In this example, the two diseases are "independent" of one another (as can be seen by non-diagonal entries being 1). Becoming infected with COVID19 prevents re-infection, whereas it is possible for an individual to become infected with Influenza multiple times (since the diagonal entries are 0 and 1, respectively).

$$\begin{array}{cc} \texttt{C.19 A} & \texttt{C.19 B} \end{array} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{array}{c} \texttt{C.19 A} \\ \texttt{C.19 B} \end{array}$$

(b) An example (excluding vaccines) of the disease matrix for two strains of COVID19. In this example, the two diseases affect one another — each disease is a "natural vaccine" of the other (and itself). We emphasize that the two strains may have different parameters (such as infectiousness), which are not defined in this matrix.

$$\begin{array}{ccc} \texttt{C.19 A} & \texttt{C.19 B} & \texttt{Flu} \end{array} \\ \begin{bmatrix} 0 & 0.25 & 1 \\ 0.25 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{array}{c} \texttt{C.19 A} \\ \texttt{C.19 B} \\ \texttt{Flu} \end{array}$$

(c) A more involved example. The Flu can be re-caught and is independent of the two COVID19 strains, while the two strains cannot be re-caught. Catching one variant of COVID19 reduces the susceptibility to the other variant by a factor of 4.

Figure 3: Examples of several disease matrices.

The most basic restriction policy is a static lockdown. Such a lockdown limits the interaction within the system in a consistent manner, as it is simply "always on". While such a lockdown may produce appealing results in a closed system, these results may not hold in a more realistic setting (see the "Open/Closed System" feature described below). Additionally, such a lockdown may take a heavy toll on the population (with respect to their "happiness"), and carries significant monetary costs. As such, a more dynamic lockdown seems like a reasonable improvement. One typical lockdown is the "On/Off" lockdown policy [10] [15], which relaxes lockdown measures when very few people are infected, and increases their strictness when too many people become infected. More specifically, it uses a single threshold value τ. If there are more than τ infected individuals, lockdown measures will be in place, and if there are fewer than τ infected individuals, no lockdown measures will be in place. While such a policy may certainly be an improvement over a basic static lockdown, it can be improved further. One such improvement is to enable multiple thresholds, each is associated with a different set of restrictions that will take place (if this threshold is crossed). Our program supports all of the previous lockdown measures, as well as allowing a policy to take into account the *activity kind* when considering the effects of a lockdown on interaction-levels with respect to this activity.

We now give a more in-depth explanation for how lockdown measures work in our system. The system holds a lockdown-level variable indicating the current level of restrictions (as a number between 0 and 4, say). For each one of those levels, we would like to reduce interactions within the network by a certain factor. This is done by (essentially) multiplying every edge-weight in the network by a scalar, which can be computed separately for each type of activity. For example, a lockdown policy might be more reluctant to close a delivery-based business than a wedding venue. To reiterate, each pair of restriction-level and activity-kind is associated with a value that will be used when considering interactions between an agent and an activity of this kind during a restriction of such level. Now, the only question that remains is how to decide on the current restriction level. This is done using thresholds: consider, for example, the percentage of ICU beds that are in use at a certain time point. We can use this value to

% of ICU beds in use	Associated Lockdown Level
0 – 20%	0
20 – 40%	1
40 – 60%	2
60 – 80%	3
> 80%	4

	0	1	2	3	4
HOUSEHOLD	1	1.1	1.3	1.5	2
SUPERMARKET	1	1	1	1	1
GYM	1	0.95	0.75	0.25	0
CONCERT	1	0.8	0.5	0	0
CLASSROOM	1	0.9	0.75	0.5	0.1
RESTAURANT	1	0.8	0.75	0.25	0
...	...	...	...	...	...

Figure 4: A (partial) example of a complex lockdown. Each threshold (in the number of critically ill people, or ICU beds in use, for example) is associated with a lockdown level. For instance, in this case, if 70% of ICU beds are taken, then the lockdown level is 3 at that specific time. Each activity kind is then associated with a list of multipliers for the various lockdown levels that may occur. In the above scenario, interactions within supermarkets are unaffected, concerts are completely closed, and interactions within restaurants are cut by a factor of 4.

determine our restriction level: for example, the ith level ($0 \leq i \leq 4$) will be in place if $i/5$ of beds are in use at that time (and this is the highest i satisfying this condition, see figure 4 for an illustration). Our system provides the functionality to base the thresholds value on either the percentage of ICU beds in use or the number of critically ill individuals at the current time. It can be further extended to include other policies, including ones which are unknown to decision-makers at real time (such as the number of *actually* infected individuals). We note that such "hidden-information" (at real time) policies do not suffer from a delay (unlike the traditional "tests-based" / "ICU-beds-based" policies, where a delay of days and possibly weeks exists). Therefore, comparing the two kind of policies could shed some insight into whether such knowledge is helpful in combating the spread of a disease.

We highlight the fact that the current restriction level is not being re-calculated at every given moment, but rather with some frequency (on a weekly or bi-weekly basis, for example). This carries two advantages. First, it prevents the lockdown measures from being extremely volatile. Second, it more closely simulates the real-world scenario of a (weekly) cabinet meeting deciding on future lockdown measures.

Other ways to implement restrictions exist. For instance, in a discrete-time simulation (which moves from one day to the next), it is possible to choose (in advance for each day) which interactions (edges) will be a part of the network for the day. Therefore, rather than multiply the edge-weight, it is possible to multiply the probability of choosing the edge instead, leaving the weight untouched, as was done in [10]. We note that a reasonable improvement of our policy would be to allow different entry and exit thresholds (for each level). While such a policy seems like a good idea, it is not always used in the real world, and we leave the implementation of such a policy for future work.

To summarize, there is a trade-off between simplicity and flexibility. Our lockdown-measures system operates over two variables: lockdown level and activity kind. It is possible, therefore, to simplify it by essentially ignoring one of them. One possibility is to consider only *On-Off* lockdowns; those are lockdowns which only have two possible restriction levels (and so are either "on" or "off"). Alternatively, one could completely ignore the type of activity in question, and simply assign a single interaction multiplier for each lockdown level. Therefore, our complex lockdown system encompasses many other methods of modeling lockdowns.

We briefly mention that the number of days that were lost due to lockdown measures is given in the "Business Days Lost" function (see figure 12 and section 4.1.1.1).

Agent Behavior - Lockdown Measures (dis)obedience. Each agent may choose to ignore (either completely or to some extent) the lockdown measures at any given time, as a function of his perspective

of the simulation up until that point and its own attributes. These behavioral changes can be broken down into two separate components, which are then combined to a single value which is used to negate the effects of the lockdown on this agent's interaction levels.

First, each agent is associated with a "baseline obedience" value (between 0 and 1). This allows us to model specific agents, or entire demographics, being inherently less obedient than others. While the exact distribution of the agents' obedience can be specified by the advanced user (and by default has all agents being completely obedient), we additionally provide simple-to-use functionality to modify the baseline obedience value for a (uniformly random) subset of the agents population (and the size of this subset is a parameter in our system).

The second part is the "History Dependent Lockdown Obedience" feature. If enabled, it allows agents to (individually) disregard lockdown measures as a function of previous events, such as too many extreme lockdown measures in recent times.

Vaccination Protocols. Each disease is associated with a vaccine that may be administered to the population within the system. A vaccinated individual will not become infected with the disease in the future. A "Vaccination Program" consists of the frequency (how often to administer the vaccines), the number of vaccine to administer (per disease), and the manner (policy, in terms of priority) in which vaccinations occur. Three such policies are given:

- Uniform: vaccines are distributed to people uniformly at random, no priority is given to anyone in particular.
- Age-based: vaccines are given to older people first. This policy can make sense from a death-preventing perspective: since older people are more likely to become critically ill, vaccinating them first could result in potentially fewer deaths.
- Degree-based: priority is given to more "active" people: the higher their (weighted) degree in the network, the higher their priority. This policy's goal is to prevent the disease from spreading (as much) in the first place, by eliminating as much interaction as possible (potential "superspreaders"). While this policy uses knowledge that is not truly available in real life, several heuristics (such as using people's phones to collect GPS/Bluetooth data in order to determine their "popularity") which attempt to estimate this information exist.

As previously mentioned, the advanced user may implement additional policies to further extend the model to their own needs.

It seems plausible to wish to have more complex "vaccination systems" in place. For instance, consider a scenario in which different vaccines for a disease exist (by different companies), an instance where some vaccines are useful against newer variants and older vaccines are not, or one in which getting vaccinated with a disease carries limited advantages (such reducing the probability of getting infected by some value, but not to absolute zero). As vaccines are internally implemented using diseases in our system, the *disease-relationship matrix* (see "Multiple Diseases" in section 3.2.2 and figures 3 and 6) allows us to extend our model to account for such scenarios.

Finally, similarly to contact-tracing methods, each vaccine dose is associated with a certain cost contributing to the "Monetary Cost" function.

Disease Testing. Testing an individual may occur for the following two reasons. First, they will be tested if they were chosen as a part of the population-wide testing feature (which chooses a specific-size subset of the population uniformly at random). The other reason for testing is on an individual-level,

due to either contact-tracing, symptom development, or both. The various rates at which testing occurs are, of course, modifiable.

A disease-test result may be "True Positive", "True Negative", "False Positive", or "False Negative". The default settings of our system are that only the first two are given (that is, by default, no errors are made when testing individuals for diseases), but the user may change these settings.

As with contact-tracing measures and vaccinations, each test performed is associated with a certain cost contributing to the "Monetary Cost" function.

Contact Tracing. Contact tracing is done by isolating (and possibly testing) contacts of people who have tested positive for the disease. When a node tests positive for a disease, a subset of its neighbors should be contact traced. Specifically, once a node has been traced, it quarantines and is scheduled to be tested for the disease. If this node tests positive the process repeats. If it tested negative, however, the decision depends on what actually happened. If it was previously sick but recovered, it can (perhaps with a small time delay) leave its self-isolation. Otherwise, it should self-isolate for a while longer (these decision rules are similar to the ones used in practice for COVID-19). Several predefined contact tracing policies are available (none, "mild", "medium", and "intense"). These affect both the rates related to the tracing process (such as the expected time until a person is traced) and the choice of the set of neighbors to be traced (what fraction of neighbors are expected to be traced, for example). A "custom" policy can be defined manually using the various parameters that are available within the system.

Each contact that is traced, as well as the contact tracing policy as a whole, costs some monetary amount which contributes to the "Monetary Cost" function (see figure 12 and section 4.1.1.1).

The choice of which neighbors to trace is an interesting one. Naturally, we would like our procedure to be as similar to the one used in practice in possible. In certain models, such as discrete ones, interaction between nodes can be modeled directly, and so one could hope to model contact tracing functionality rather well. However, in our model, this might be harder than one would initially think. Although the principles behind our model are based on individual interactions, the actual interactions are hidden (i.e., are not modeled explicitly but through the rates of nodes becoming infected etc), and so we run into some issues when trying to trace contacts (past "actual" interactions) between nodes. So how does one trace contacts in such a model? Several alternatives exist. One possibility is to simply trace back a (random) subset of a node's neighbors, for example half of its neighbors. This carries the advantage of being extremely simple, but at the cost of not accurately modeling the real concept: correlations between groups of contacts are completely lost, for example. We note an alternative method which attempts to remedy some of these issues exists (although we do not implement it in our system). When a (subset of) neighbors of an infected node are traced, they can be partitioned to two groups - those who were actually infected by this node (and so interaction certainly happened), and those who were not. We note that we can more accurately trace the first group by assigning an "infecting-parent" to a node at the moment of infection (at random with probability proportional to the edge weight), and make sure to include this parent (as well as any node whose parent is the current node) in any contact tracing procedure involving this node. This way, we can get more "reasonable" traces which more accurately correspond to what had happened in the simulation. Nodes of the second group, however, are not (and it is unclear to us how it could be, without modifying the model in order to simulate interactions) dealt with in this manner. While certainly imperfect, one could argue that these nodes carry much less importance in tracing, and that by randomly selecting a subset of the neighbors to complement the group of certain contacts, we simply replace one group of "innocent nodes" with another, which does not make a great difference.

Hospitalizations. Once infected and symptomatic, an individual may become critically ill. This pos-

sible transition occurs with a rate which depends on both the disease (some diseases are more dangerous than others) as well as as the individual's attributes (such as age).

If an individual is in a critical state, their likelihood of dying increases substantially. A critically-ill individual requires specialized equipment, which we generally refer to as "hospital/ICU beds". If a bed is available, it will be used until the individual either recovers or dies. If no bed is available, a new one will be sought periodically (every hour, say) until one becomes available or the individual dies (a critically-ill individual with no hospital bed has a very quick death rate). Obviously, the various rates associated with each of these states and transitions can be modified.

The number of hospital beds is determined before the simulations begin, and each bed contributes to the "Monetary Cost" function. A reasonable improvement would be to allow one to "buy" additional beds during simulations as well.

Open/Closed System. Often times simulating very harsh measures can lead us to the conclusion that simply limiting most interaction for a small time period is enough to eradicate any disease. However, in reality this is usually not the case – communities are rarely "closed systems"; new people can fly into the country, and infections can appear seemingly out of nowhere. To overcome and model this issue, the *Open System* feature allows spontaneously infecting randomly selected susceptible individuals at arbitrary times. This feature therefore also provides the ability to delay introduction of viruses into the system, thereby allowing one to simulate "mutations" occurring at specific time-points.

Pausers. A "Pauser" is an object that can be used to control and affect the simulation arbitrarily. Each pauser is associated with a function which is evaluated at every "new day" of the simulation. Such a function checks for certain conditions (for example, whether a week has passed since the last time the pauser was used, or some other information regarding the state of simulation), and if these conditions are met, modifies the simulation in whichever way it wants to. Many of the features available within the simulation (such as lockdown measures and vaccinations) are internally implemented using pausers. As such, an advanced user (with some additional familiarity with the Simulator object) may find the Pausers feature extremely useful in extending the model even further, to their own needs. For a more in-depth explanation, see section 5.3.0.1 in the appendix.

4 The Program

4.1 High Level Overview

Generally speaking, simulations are divided into batches, where each batch consists of several simulations that have the same parameters but use different randomness. This makes the process of evaluating the effects of parameter modification easy.

To modify any of the parameters, one can simplify edit the given `default_parameters.py` file. One can store several parameter files and give the one they wish to run the backend on by running
`python -m backend.main -p parameter_file_path` instead of the usual `python -m backend.main`.

After running the code, a folder named "results" is created. It contains two folders, "pickles" and "experiments". "pickles" contains serialized versions of agent-activities graphs (constructed under the process described in 3.1.2). which can be later re-used saving computation in the (re-)construction process, if one wishes to run simulations on these graphs in the future. The "Experiments" folder contains the actual results of simulations that were performed. Each experiment is a folder with the corresponding date, time, and number of people as its name. Within it are the parameters used in the simulations, extra logging information stored, the difference in parameters between batches, and various plots and figures comparing

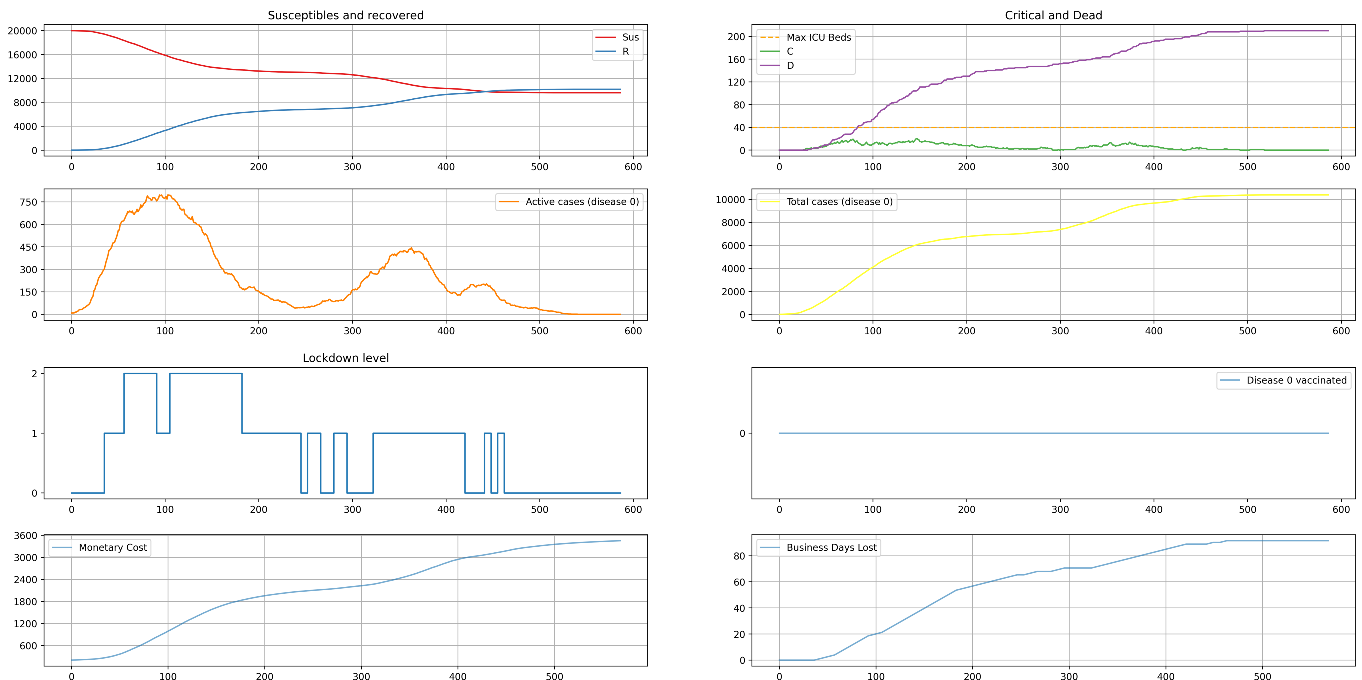

Figure 5: An example of the output generated for one of the simulations within a batch. 20,000 people and 20,000 activities were used, with "Complex" lockdown measures, "medium" contact tracing, and no other "advanced" features.

key data across the batches. Additionally, it contains batch-specific sub-directories, each of which contains many other plots (one for each simulation within the batch, see figure 5), activities distribution for the batch , pickled versions of the simulation results objects, and others.

A table highlighting the key parameters and the different parameters across the different batches (and some of the statistics generated for each batch) is given in the console, experiment folder, and the GUI. If a very high number of parameters differ, and the table is too large to be rendered correctly, `batches_table.csv` contains a comma-separated version of the table.

4.1.1 The GUI

The GUI provides a simple way to modify most parameters. When using the GUI, the first step, shown in figure 6, is to choose some of the general parameters that will be used in all simulations, such as the number of people and diseases that are present in the graph, and the number of simulations (per batch) to run.

Next, one needs to decide on how many batches of simulations to create, and the parameters of each batch. We remind the reader that all simulations within a batch use identical parameters, and only differ in their random seed. These parameter modifications are done in the "Batch Parameters" tab, which is shown in figure 7.

Once "Run Simulations" is pressed, the backend will perform the specified simulations, and relevant information can be seen in the "Live Simulation" and "Results" tabs.

The "Results" tab will initially show a progress bar showing the number of simulations that have finished and an estimated overall progress which relies on the number of days that simulations take on average. Once

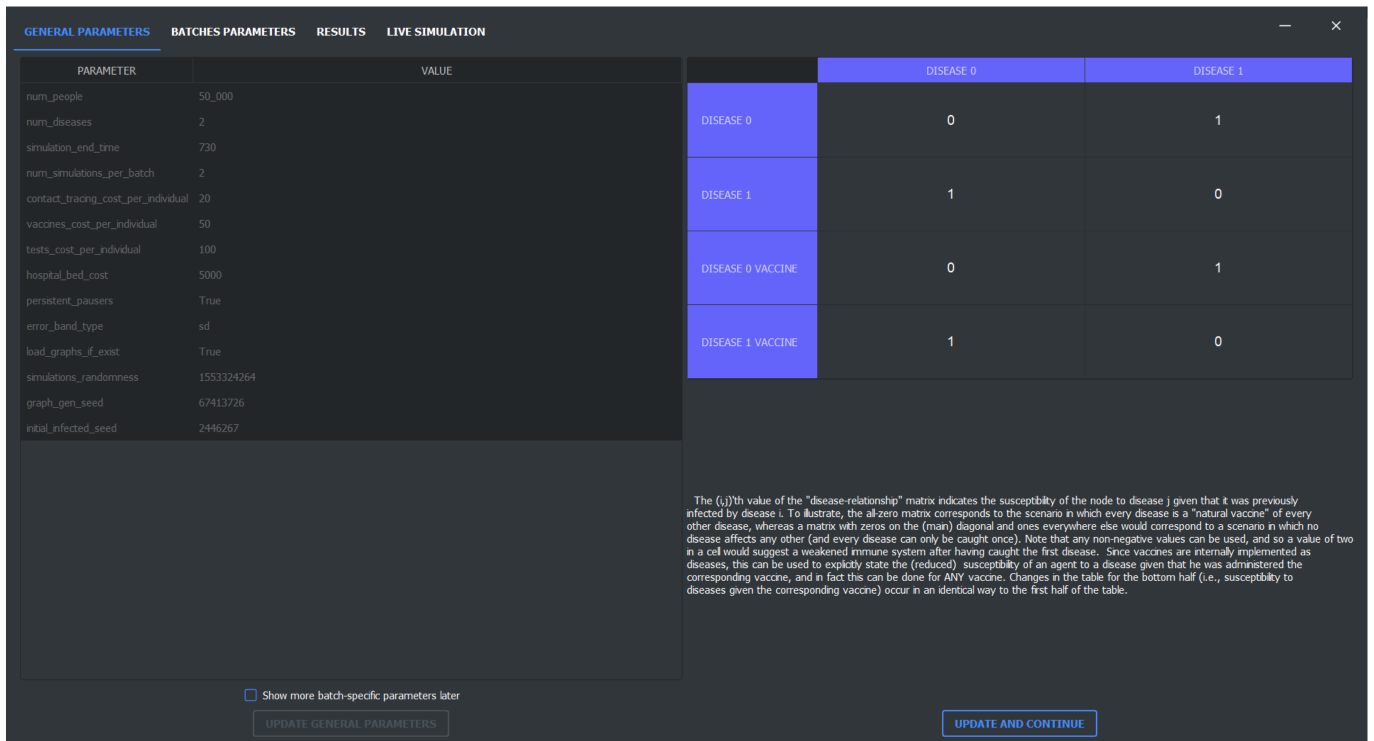

Figure 6: The “General Parameters” tab, which includes general parameters on the left and the “disease matrix” (as explained in section 3.2.2) on the right.

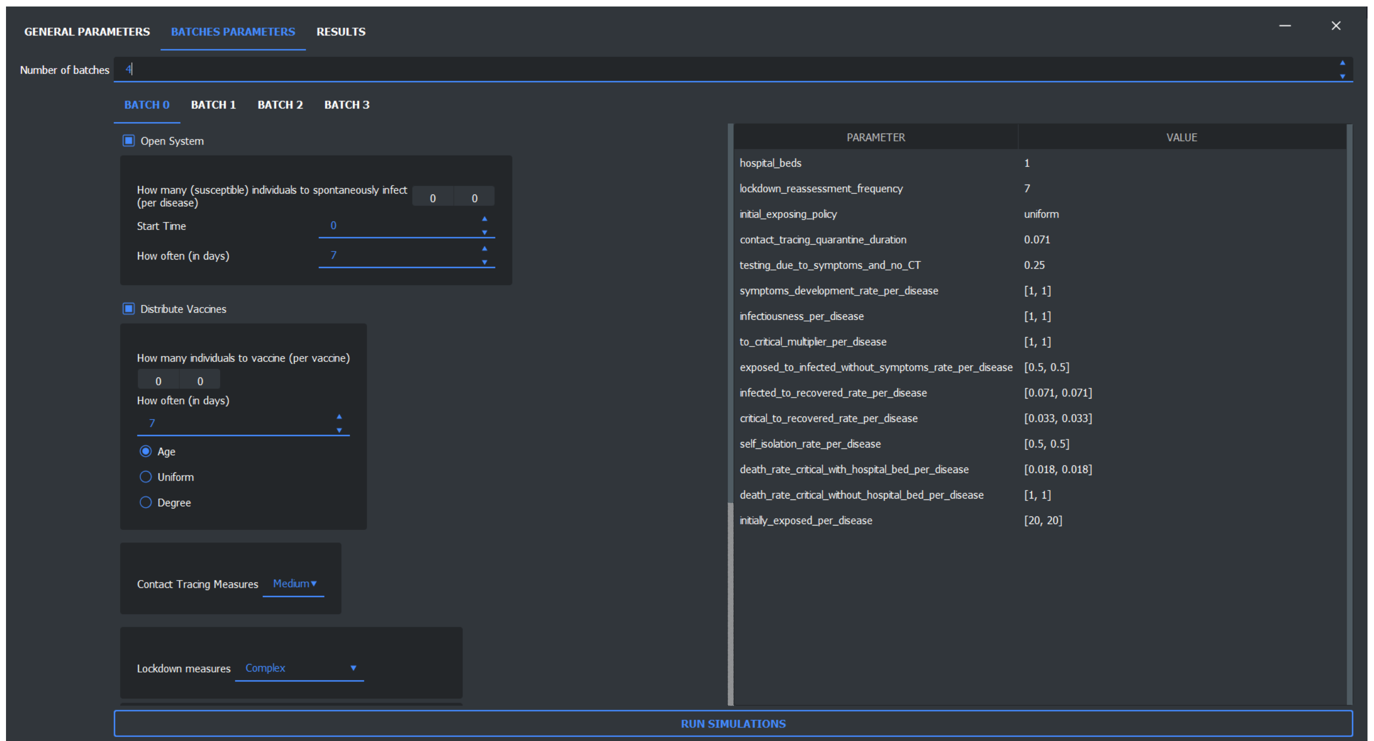

Figure 7: The “Batches Parameters” tab. Initially, one chooses how many batches they wish to have, and then updates the various parameters of each batch. On the left are the “main” parameters such as whether to apply contact tracing and lockdown measures, whereas on the right are the numeric parameters such as the various rates and constants that will be used in simulations of this batch. We note that while each parameter is partnered with a tooltip explaining its meaning, some of the parameters could be a bit more difficult to understand when compared to others, and so reasonable default values were supplied.

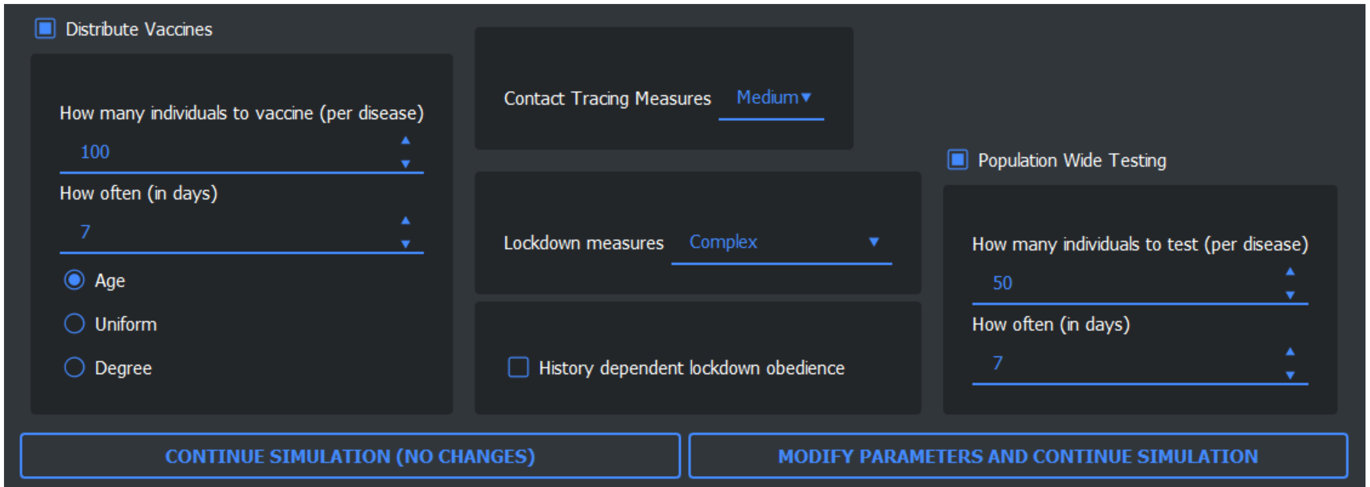

Figure 8: The various parameters and methods that may be changed in real (simulation) time, available in the "Live Simulation" tab. Simulations can be paused (and parameters can be changed) constantly. Note that many predefined policies are available to use (and are not shown in this figure).

all simulations are done, this tab will show a limited subset of the results, such as various plots comparing between batches. More detailed results can be found in the experiment folder, which can be opened through this tab as well.

The "Live Simulation" tab is the one responsible for the GUI's second mode: it provides the interface to view, control, and modify the (first) simulation in real-time (as can be seen in figures 9 and 10)). In addition to viewing the current state of the simulation, one can pause the simulation, modify the current parameters (specifically: vaccination protocols and their policies, contact tracing measures, lockdown measures, agents' obedience, and population-wide testing of agents, see figure 8), and continue the simulation with the new parameters in place, seeing the effects caused by the modifications in real-time.

4.1.1.1 Cost Functions

In addition to the usual plots (SIR-like), the live simulations tab provides two cost functions that are updated in real (simulation) time: *Business Days Lost* and *Monetary Cost* (figure 12). These plots are a function of the various parameters, measures, and decisions that were made to combat the disease(s), both those chosen initially and the modifications made during the simulation's run-time. This further allows the user to evaluate the effectiveness of their policy modifications, not only in terms of their effectiveness against the disease(s), but also their (absolute and relative) cost. In particular, the *Business Days Lost* function shows how many days were "lost" due to lockdown measures (excluding within-household interactions), whereas the *Monetary Cost* function considers costs associated with the number of vaccinations and tests that were performed, the number of people that were contact traced, and the cost of the contact tracing policy as a whole. The various costs of the different intervention methods can be modified in the parameters tab.

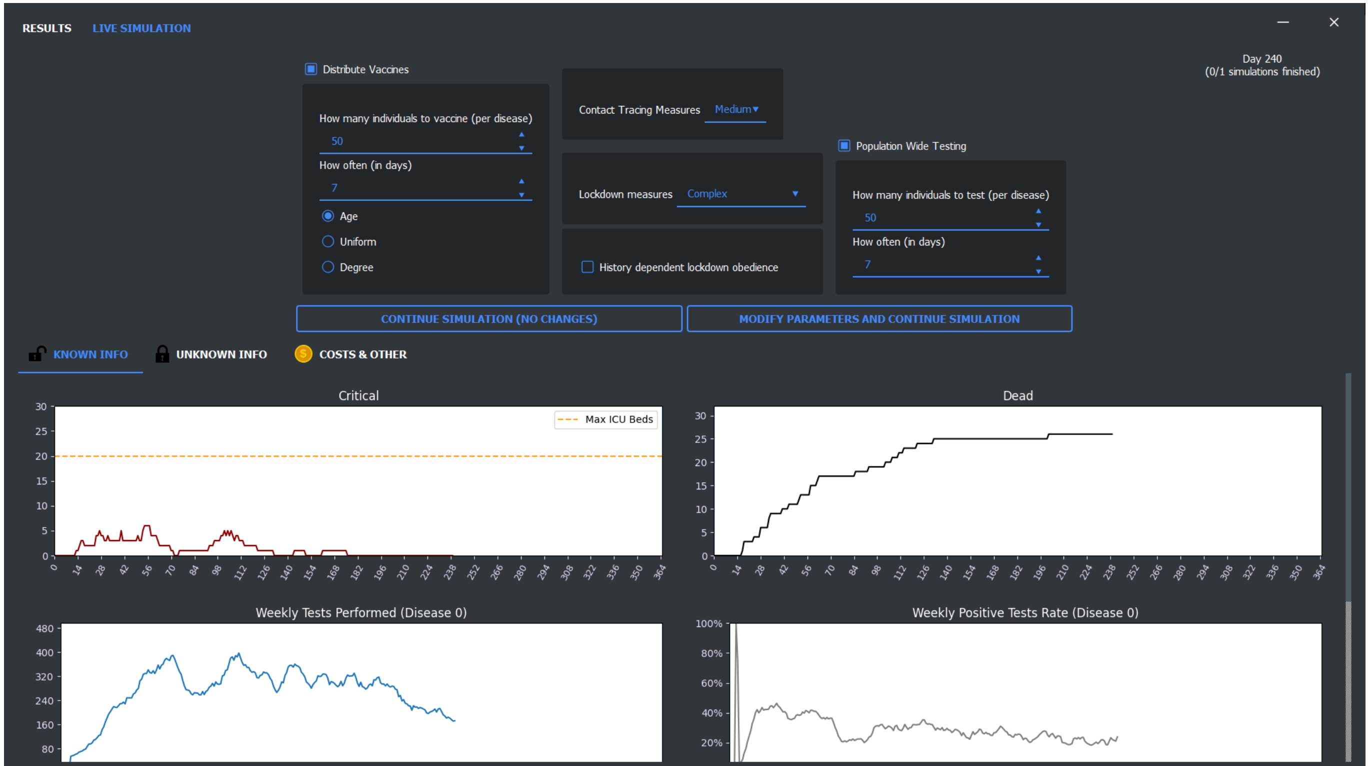

Figure 9: The “Known Info” section of the “Live Simulation” tab. This section contains live plots of information which is available to decision-makers in real-time. This includes data such as the number of dead and critically ill individuals, disease-testing and vaccination numbers, etc.

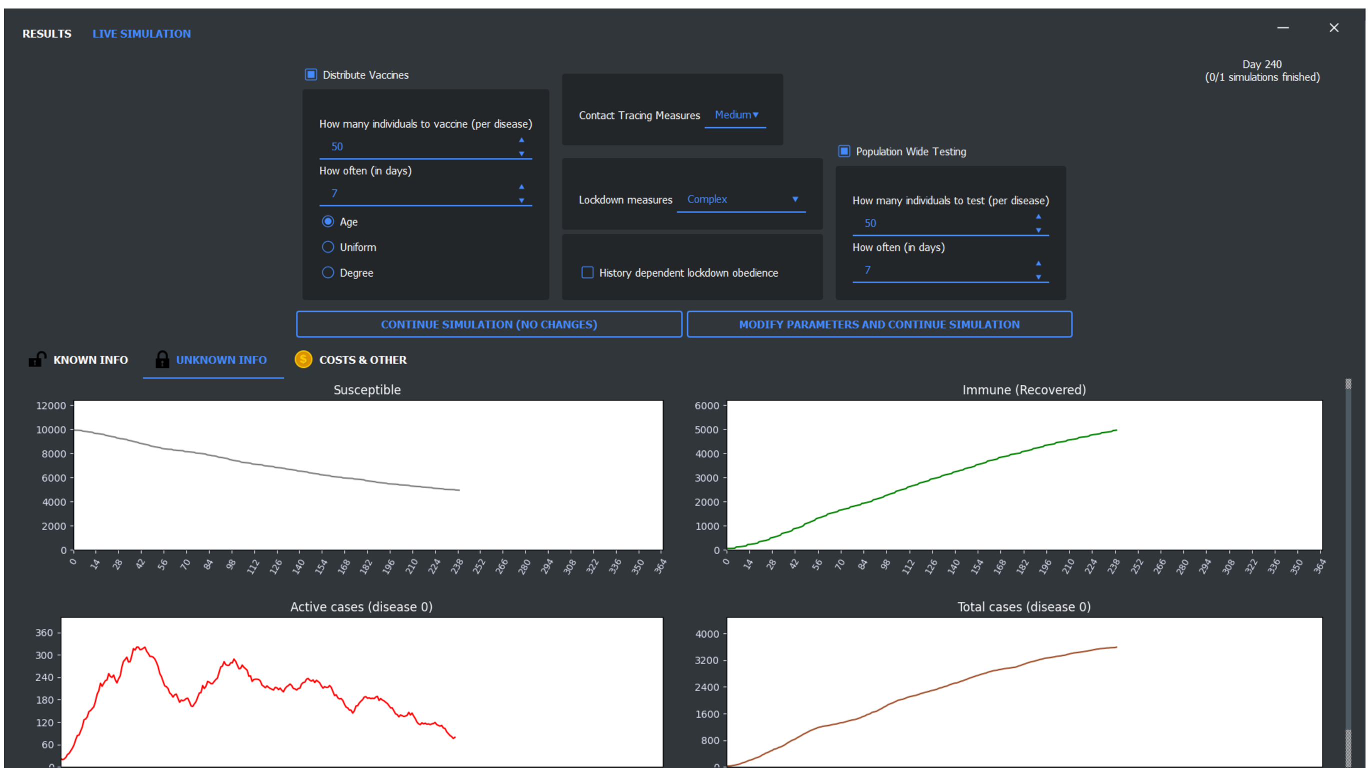

Figure 10: The “Unknown Info” section of the “Live Simulation” tab. This section contains live plots of information which is not necessarily available to decision-makers in real-time. This includes data such as the number of susceptible and insusceptible (immune/recovered) individuals at any given time, as well as per-disease information such as the number of (currently) active and total cases. One could investigate whether having this information accessible to them makes a difference in their now better-informed decision-making process when deciding on policy modifications.

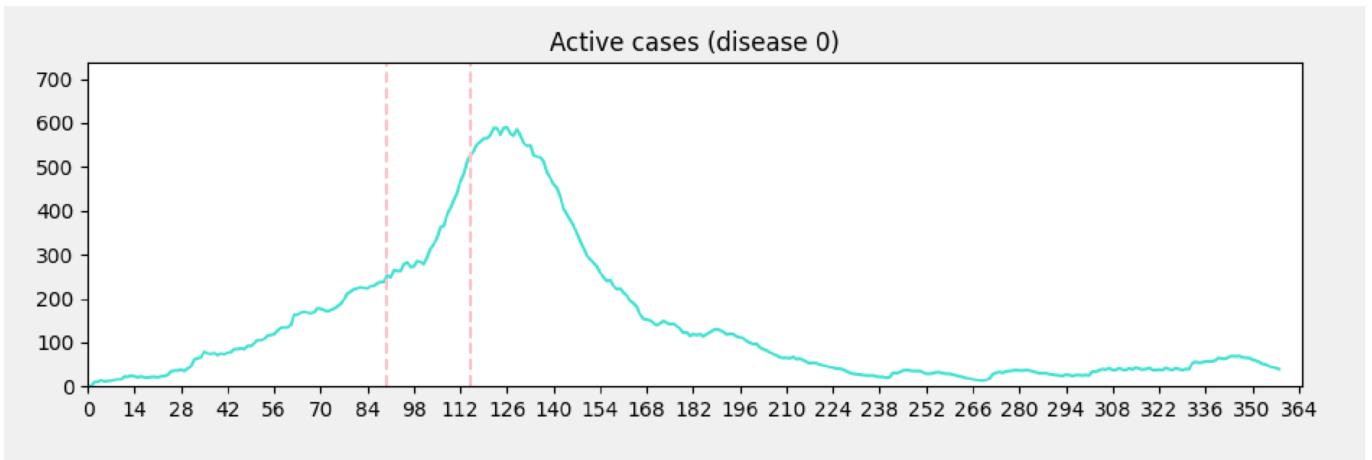

Figure 11: An example of how live modifications to the parameters can affect the number of active cases in the simulation. The first dashed line indicates the time at which some of the mitigation efforts were relaxed (and the number of new cases quickly increased), whereas the second one corresponds to the time point at which interventions were modified to be more strict (causing cases to trend down).

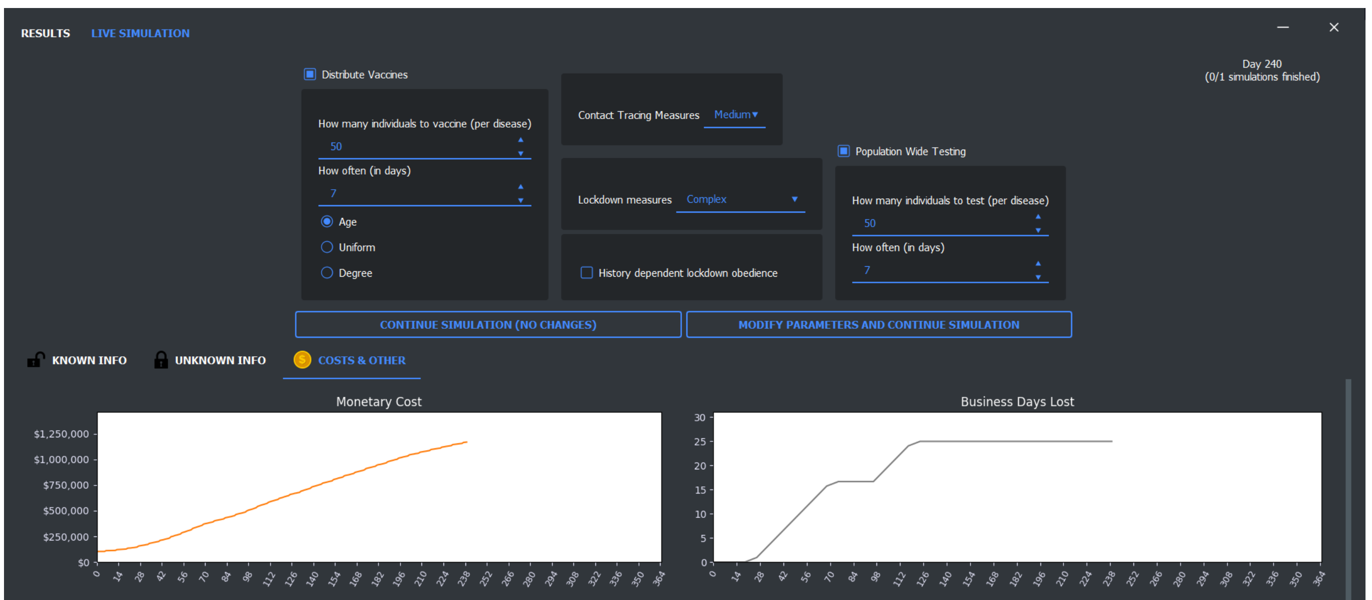

Figure 12: The "Live Simulation" tab also contains a section relating to cost functions and lockdown levels. In particular, the two cost functions are "Monetary Cost" and "Business Days Lost". "Monetary Cost" combines monetary costs associated with contact tracing, disease-testing, and vaccination doses. "Business Days Lost" shows the number of days of interactions (except those via households) that were lost due to lockdown measures. Similar plots are stored in the experiment directory on completion of the simulations (per simulation as well as on a per-batch basis.

Batch #	vaccination_program	contact_tracing	open_systems	lockdown_measures	population_wide_testing_policy	Total cases (disease 0)	Total Deaths
0	(age-based, 7, [1000])	medium	[]	Complex(6 levels, method=beds)	None	41.42%	0.78%
1	(uniform-based, 7, [1000])	medium	[]	Complex(6 levels, method=beds)	None	31.9%	0.75%
2	(degree-based, 7, [1000])	medium	[]	Complex(6 levels, method=beds)	None	22.54%	0.55%

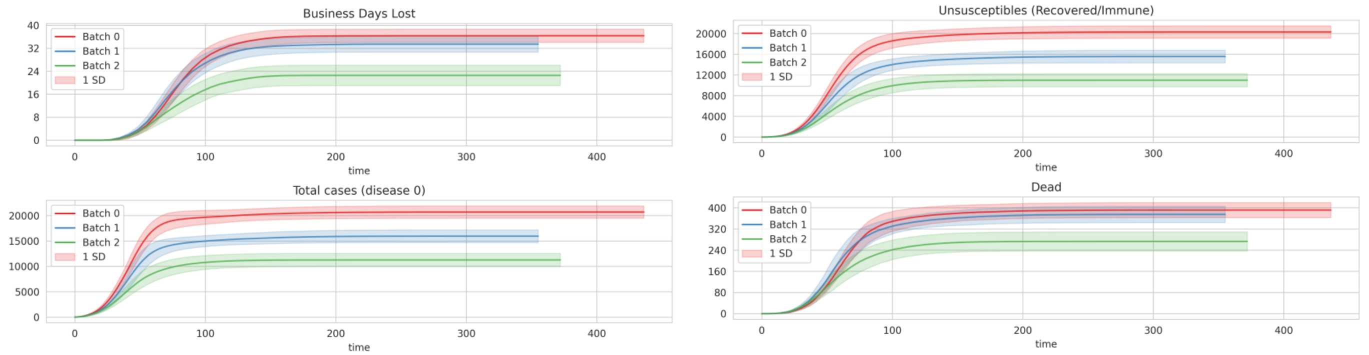

Figure 13: An example of one of the many figures generated by the program. This specific experiment compares between the three vaccination policies (each simulation consisted of 50,000 people and activities, with 30 simulations per batch, with a very infectious disease). 1000 Weekly vaccinations with different policies were used in each batch. In this experiment, comparable numbers of dead people were found between “age-based” (red) and “uniform” (blue) policies. This is because “age-based” specifically targets people who are in greater danger of dying, and so performs relatively well in this regard. On the other hand, it does not contain the disease to the same extent as the “uniform” policy, and so this comes at the expense of the rest of the population. The “degree-based” (green) policy seems to be the overall best policy, both in terms of number of infected and dead. This policy, however, is much harder to implement in real-life.

We now give several examples of simulations performed by our system. While we do provide default values for all parameters, we do not wish to claim that these values perfectly reflect any real-world viruses and parameters, and that the selection of these parameters is the responsibility of the user, based on their assumptions and models. Figure 13 demonstrates the effects of different vaccination policies (with respect to the number of infected and number of dead individuals). Figure 14 demonstrates differences that occur when subsets of various sizes (of otherwise an entirely obedient agent population) are disobedient (with respect to lockdown measures) when combating a very infectious disease. Finally, figure 15 shows what happens when a second variant enters the system after several months.

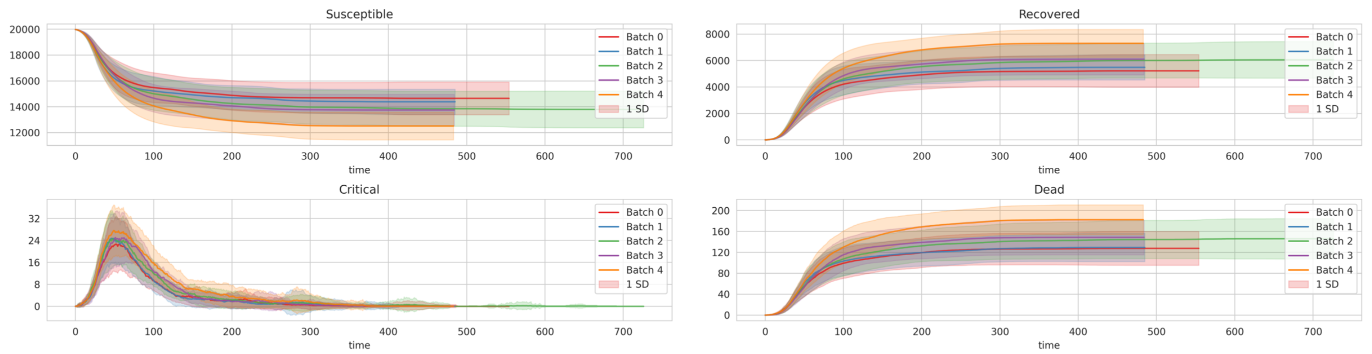

Figure 14: In this experiment, a 5-batch comparison for agent obedience was made. The batches differed in the fraction of (completely) disobedient agents (from the entire population), where the values used were 0, 0.01, 0.1, 0.15, and 0.2, respectively. As shown, the effect suddenly becomes more noticeable when roughly 20% of the agents are disobedient. Each batch consisted of 30 simulations over a network of 20, 000 people and 20, 000 activities.

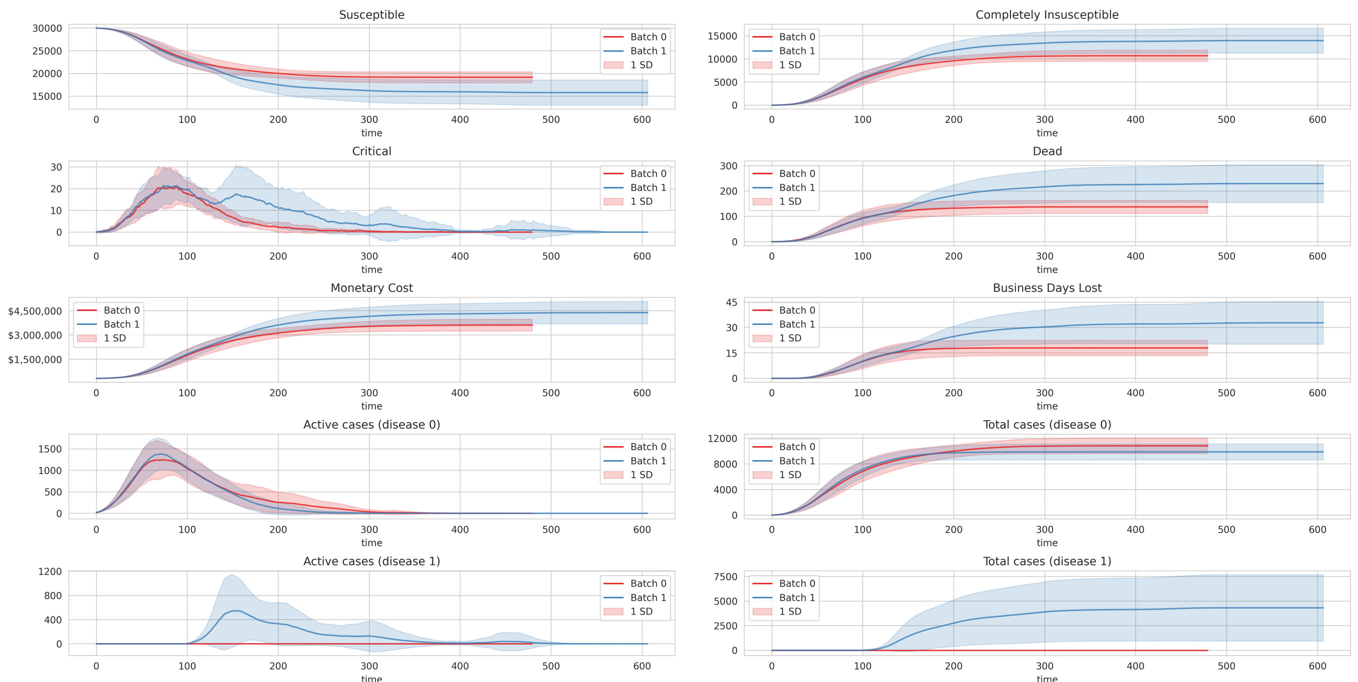

Figure 15: Multiple-strains comparison between 2 batches. Each batch consisted of 30 simulations over a network of 30, 000 people and 30, 000 activities. The first batch had only one variant of COVID19 spreading in its simulations. The second batch scheduled the introduction of a second, more infectious, variant of COVID19 at time 100 (by infecting 5 random individuals with it). No vaccination protocols were used in either batch, resulting in more devastating effects. It is worth mentioning we observed much more simulation-data spread in multiple-disease simulations, when compared to the more concentrated results for single-disease ones.

References

[1] Abdulkadir Atalan. "Is the lockdown important to prevent the COVID-19 pandemic? Effects on psychology, environment and economy-perspective". In: *Annals of Medicine and Surgery* 56 (2020), pp. 38–42. ISSN: 2049-0801.

[2] Christopher L. Barrett et al. "EpiSimdemics: An efficient algorithm for simulating the spread of infectious disease over large realistic social networks". In: *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. 2008, pp. 1–12.

[3] Marian Boguñá et al. "Simulating non-Markovian stochastic processes". In: *Physical Review E* 90.4 (2014). ISSN: 1550-2376.

[4] Suruchi Deodhar et al. "An Interactive, Web-Based High Performance Modeling Environment for Computational Epidemiology". In: *ACM transactions on management information systems* 5 (July 2014).

[5] J. L. Doob. "Markoff Chains–Denumerable Case". In: *Transactions of the American Mathematical Society* 58.3 (1945), pp. 455–473. ISSN: 00029947.

[6] Daniel T Gillespie. "A general method for numerically simulating the stochastic time evolution of coupled chemical reactions". In: *Journal of Computational Physics* 22.4 (1976), pp. 403–434. ISSN: 0021-9991.

[7] Daniel T. Gillespie. "Exact stochastic simulation of coupled chemical reactions". In: *The Journal of Physical Chemistry* 81.25 (1977), pp. 2340–2361.

[8] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the Spread of Influence through a Social Network". In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* 137-146 (July 2003).

[9] Istvan Z Kiss, Joel C Miller, and Peter Simon. *(Book) Mathematics of epidemics on networks: from exact to approximate models*. Springer, 2017.

[10] Yoav Kolumbus and Noam Nisan. *On the Effectiveness of Tracking and Testing in SEIR Models*. 2020. arXiv: 2007.06291 [q-bio.PE].

[11] Joel C. Miller and Tony Ting. "EoN (Epidemics on Networks): a fast, flexible Python package for simulation, analytic approximation, and analysis of epidemics on networks". In: *Journal of Open Source Software* 4.44 (2019), p. 1731.

[12] Kermack William Ogilvy and McKendrick A. G. "Contributions to the mathematical theory of epidemics". In: *Proc. R. Soc. Lond.* A 115 (1927), pp. 700–721.

[13] Kermack William Ogilvy and McKendrick A. G. "Contributions to the mathematical theory of epidemics. II. —The problem of endemicity". In: *Proc. R. Soc. Lond.* A 138 (1932), pp. 55–83.

[14] Kermack William Ogilvy and McKendrick A. G. "Contributions to the mathematical theory of epidemics. III.—Further studies of the problem of endemicity". In: *Proc. R. Soc. Lond.* A 141 (1933), pp. 94–122.

[15] Tomas Pueyo. https://medium.com/@tomaspueyo/coronavirus-the-hammer-and-the-dance-be9337092b56.

[16] A.M. Ramos et al. "Modeling the impact of SARS-CoV-2 variants and vaccines on the spread of COVID-19". In: *Communications in Nonlinear Science and Numerical Simulation* 102 (2021), p. 105937. ISSN: 1007-5704.

[17] Ross Ronald. “An application of the theory of probabilities to the study of a priori pathometry. Part—I”. In: *Proc. R. Soc. Lond.* A 92 (1916), pp. 204–230.

[18] Ross Ronald. *The prevention of malaria*. John Murray, 1911.

[19] Ross Ronald and Hudson Hilda P. “An application of the theory of probabilities to the study of a priori pathometry. Part—II”. In: *Proc. R. Soc. Lond.* A 93 (1917), pp. 212–225.

[20] *SEIR+*. https://github.com/ryansmcgee/seirsplus.

[21] Petrônio C.L. Silva et al. “COVID-ABS: An agent-based model of COVID-19 epidemic to simulate health and economic effects of social distancing interventions”. In: *Chaos, Solitons & Fractals* 139 (2020), p. 110088. ISSN: 0960-0779.

5 Appendix - System Design, Objects, and Implementation Detail

In this section we describe some of the objects that are used throughout the codebase, as well as their various attributes and methods.

5.1 Graph-related objects

Nodes

A node in the graph is either an *Activity* or a *Person*. Every node has the following (individual) attributes:

`Incoming Infection Multiplier` Used when calculating the node's chance of getting infected

`Outgoing Infection Multiplier` Used when calculating the node's chance of infecting others

`Location` A coordinate on a discrete two-dimensional grid, where the node is located. The distance between two nodes will dictate their probability of being connected in the graph. While every activity is initialized with a (uniformly random) location, a person's location is that of its assigned household.

Activity

An *activity* (sometimes referred to as a *location*) object also has the following attributes (in addition to those given previously in the Node section):

`kind` An `Activity_Kind` enum, such as `HOUSEHOLD`, `SCHOOL`, `GYM`, etc. Every such kind is associated with various attributes (such as the activity's infectiousness level and typical visiting frequency) which are described in the `activities_factory` parameter.

`max_capacity` the maximal number of people this activity can be connected to in the graph.

In addition, the `gen_activities()` function returns the list of activity objects to be used in the graph, based on the given parameters.

Person

A *person* (or *agent*) object, has the following additional attributes:

`age` Used to determine which activities this person will be connected to in the graph (a teen is more likely to attend school compared to an adult), as well as in the computation of the person's `Criticalness Multiplier` attribute

`criticalness_multiplier` Multiplies the probability of the person becoming critically ill based on the desired factors (age, obesity level, etc). The calculation can be modified in the corresponding method.

`social_type` Enum valued, used (in addition to `age`) when deciding how many activities of different kinds to connect to. Can be later used to affect the person's disobedience.

`behavioral_state` A class responsible to handle the person's behavioral aspects (such as their baseline disobedience level and their disobedience level as a function of the current state of the simulation).

In addition, it has the following methods:

`disobedience()` A wrapper for the `Behavioral State` object's identical method.

`__calc_criticalness_multiplier()` Calculates the `criticalness_multiplier` attribute.

Finally, the `gen_people()` function returns the list of people objects to be used in the graph. Uses various parameters such as wanted age distribution for the population, etc. An implementation we found reasonable enough is supplied, but it can, as usual, be readily changed.

Behavioral State This class is used when considering agents' ability to make choices (such as ignoring, to whatever extent, lockdown measures).

5.2 Graph

Any NetworkX Activities-People (where these two objects are as defined above) bipartite graph can be used in the simulations (be sure to define it using our *BipartiteGraph* class). The only additional (trivial) requirement for a graph to be compatible with our system are that it is augmented with `.people` and `.locations` fields that point to sequences of the graph's people and activities, respectively. Additionally, the weight attribute on each edge should be called 'weight'.

5.3 Simulation-related objects

Simulator

The simulator object is in charge of running the entire simulation. It takes many arguments, but those should not be given to it directly in most cases (and so we won't expand on them here). Instead, it is recommended to use the `run_simulation()` function of the main file.

We note some of its key attributes:

`cumulative_monetary_costs` A list which holds for each day the cumulative *monetary costs* of the simulation thus far. It is updated (daily) in the `update_cumulative_monetary_cost()` method.

`cumulative_days_lost` A list which holds for each day the cumulative (total) number of *"business days lost"* thus far (due to lockdown measures). The calculation of what constitutes the unit of a "business day" is done in the `update_days_lost()` method. Specifically, it considers the weighted reduction in interactions in the graph between agents and activities (excluding households). For example, suppose that the current lockdown halves interactions between agents and gyms, and that originally these interactions accounted for some ε fraction of all interactions in the graph. Then, as these interactions were reduced by half, gym-lockdowns account for $1/2\varepsilon$ of a day lost (of the current day). Summing up across all non-household activities, we obtain a measure of "days lost" which is weighted by both the relative interaction through this activity kind as well as the lockdown's effect for this specific kind. Observe that this measure is maximized (and equals 1) if the lockdown shuts down all non-household interactions for that day, and is minimized (being 0) if no lockdown is in place.

`simulation_results` Holds a `SimulationResults` objects containing (a possibly summarized version of) the results of the simulation.

`lockdown_restrictions_timeline` holds pairs of the form (t, ℓ) where t is a time in the simulation and ℓ is the lockdown level at that time. At any given time, it's last entry is used to determine current lockdown measures. This list is updated whenever the lockdown-restrictions pauser is called.

`max_lockdown_level_timeline` holds pairs of the form (t, ℓ) where t is a time in the simulation and ℓ is the maximum lockdown level at that time. This list is updated whenever the lockdown-restrictions pauser

is called. It is used, for example, to determine how severe a the current lockdown level is (compared to the maximum available one) when considering agent behavior.

`nodes_to_CT` is a list of nodes that should be contact traced at any given time

The following three basic simulator's methods are based on [11]:

`calculate_rate()` Calculates the rate of a node (to change state).

`calculate_new_state()` Calculates the new state of a node once this node was chosen to change state.

`calculate_influence_set()` Calculates which nodes should re-calculate their rates given that a certain node changed state.

These three functions control the "state-machine" of the various simulation-states of the nodes. Consider for example a node that becomes infected. `calculate_new_state()` is then called to calculate its new state: the `previously_infected` and `currently_infected` attributes of the relevant `PersonDiseaseState` should be now set to True. Next, the collection of nodes which might need to recalculate their rate (for example, its neighbors are more likely to become infected now) is calculated using `calculate_influence_set()`. Finally, each of these uses `calculate_rate()` to find their new rate, and the process repeats (by first deciding which event occurs by randomly sampling a node and a corresponding state transition).

Some other methods are:

`calc_visiting_likelihood()` Calculate a 'current' weight between the given person and activity, which depends on current events such as the the restriction level and the person's current behavioral state.

`calculate_contacts_to_CT()` Decide on which contacts should be contact traced, given that the relevant person is a known positive case of a certain disease.

`current_restrictions_level()` Return the current lockdown measures level of the system.

SimulationResults

A dataclass holding most relevant information produced by the simulation. Is usually returned instead of the entire Simulator object for memory-efficiency reasons (which can be modified by the `return_simulator` argument of the corresponding `SimulationArguments` object). If even less data is required, it can be further summarized using the `last_timestamp_only()` method which "throws away history" up until the very last time-step.

Its attributes are:

`times` is an array containing the times in which a simulation-event happened

`status_count_data` is a dictionary mapping each `SimulationState` to an array holding how many nodes had this state at the various simulation times.

`node_history` holds, for each node, its entire history of simulation statuses. This information is only stored if `return_full_data` is `True`.

`lockdown_restrictions_timeline` is a list of pairs, where each pair is a simulation-time and the corresponding lockdown level at that time.

`cumulative_monetary_costs` is a list of cumulative daily monetary costs, as defined for "Simulator" above.

`cumulative_days_lost` is a list of cumulative (daily) "business days lost", as defined above.

Intervention Methods

We briefly describe the various classes relating to intervention methods.

`VaccinationProgram` A class holding all relevant information regarding a vaccination program to be used in the vaccination pauser (see the `vaccinate_individuals` pauser below). This includes `how_many_per_vaccine` (a list containing, for each vaccine, how many individuals to vaccinate for this vaccine), `frequency` (how often, in days) to perform this vaccination program, and `policy`. `policy` should be either `uniform` (no priority is given when choosing which individuals to vaccinate), `age` (which gives priority older people), and `degree` (which gives priority to agents with higher weighted degree in the graph).

`PopulationWideTestingPolicy` A class holding all relevant information regarding a population testing policy to be used in the relevant pauser. Specifically, it simply contains `frequency` (how often to perform these tests) and `how_many_per_disease` (a list containing for each disease how many tests to perform whenever the relevant pauser is activated).

`OpenSystem` A class holding all relevant information regarding an open system protocol to be user in the relevant pauser. This includes `how_many_per_disease` (a list containing, for each disease, how many individuals to spontaneously infect for this disease), `times` (at what times, in days, to perform this infection process), and `policy`. `policy` should be either `uniform` (no priority is given when choosing which individuals to vaccinate), `age` (which gives priority older people), and `degree` (which gives priority to agents with higher weighted degree in the graph). We briefly mention that instead of `times`, `frequency` and `start` can be supplied.

`OnOffLockdown` Contains a single `threshold` and a single `effect`. After `threshold` is passed (percentage of critically ill individuals among entire population by default, though this behavior can be modified to check against available ICU beds etc), lockdown will become "On" and all edge-weights (except household-related ones) will essentially be multiplied by `effect`. Once total number of cases drops below `threshold`, lockdown becomes "Off" and interactions are back to normal. For various pre-defined `OnOffLockdowns`, check the `lockdown_measures.py` module.

`ComplexLockdown` Similar to `OnOffLockdown` but can contain multiple `thresholds` and multiple `effects` (which is now no longer a scalar but a dictionary mapping each activity-kind to a list of scalars corresponding to the various effects that take place for an activity of this kind for the various lockdown-levels that result from crossing the various thresholds. A predefined complex lockdown example is given as `ComplexLockdownExample()` in the `lockdown_measures.py` module.

`ContactTracingMeasures` An abstract class holding information which relates to contact-tracing. Several kinds of contact tracing methods exist, specified by "kind". This argument ("kind") should be one of "None", "mild", "'medium", "intense" and "manual", and a subclass for each of these kinds exists. Choosing any of these kinds (by initializing the corresponding subclass) will result in this class holding specific rates regarding the contact-tracing procedure that will be applied to the dictionary of all parameters. If one wishes to manipulate these rates directly instead of relying on one of the predefined aforementioned three kinds, they should choose the "manual" kind and modify the rates manually in the parameters dictionary themselves. The relevant rates are:

`rate_for_time_from_identified_location_until_person_gets_contact_traced`

`rate_for_time_from_positive_test_until_location_gets_contact_traced`

`rate_for_time_from_identified_location_until_person_gets_contact_traced`

`contact_tracing_weight_multiplier`

`testing_due_to_CT_and_no_symptoms`

`testing_due_to_CT_and_symptoms`

the specific values they receive for the predefined policies can be seen in the code (some somewhat arbitrary yet reasonable values were chosen, as in other places).

5.3.0.1 Pausers

A Pauser is an *abstract* object that can be used to control and affect the simulation in arbitrary, user-defined, ways (at every discrete integer time-step, namely, every simulation day). To be more specific, each Pauser is associated with a function (called `if_then()`), and at every new simulation day, this function is evaluated. Any behavior could be used by implementing a Pauser with the required functionality. To do so, it must implement the `if_then(simulator, current_simulation_time)` → `bool` method, which returns whether the pauser activated (i.e., any changes have been made by this function). Note that some familiarity with the `Simulator` object is typically required in order to implement a useful Pauser, if one wishes to modify its internal state. We highlight the fact that pausers should not be created using this class, but only through the *SpecificTimesPauser* class below.

SpecificTimesPauser A `SpecificTimesPauser` module exist. All pausers within our system should be of this type. This module provides the means to easily create `Pausers` which should take effect as a function of the *current time* of the simulation. For example, lockdown measures can be implemented by checking for certain conditions (the number of infected people, say) *on a weekly basis*, and if the conditions are met the pauser imitates a government deciding to increase lockdown measures in its weekly cabinet meeting.

A `SpecificTimesPauser` takes as arguments a list of simulation-times, and a list of functions, and preforms the corresponding function once each time-step has arrived. If all functions are identical (as is usually the case), one can use `single_func_specific_times_pauser()` function, which takes a list of simulation times and the unique function. To simplify even further, one can use the `every_x_days()` and default
`days_in_a_month, weekly, biweekly, monthly, yearly` functions in order to easily generate lists of simulation-times which repeat every certain number of days.

The `if_then()` method of such a pauser is implemented to allow the above functionality. Furthermore, it is even implemented to modify the internal state of the `Simulator` by itself - all it requires is for each supplied function (or the unique one if `single_func_specific_times_pauser()` is used) to return a pair, `nodes_to_calc_rate_for, nodes_to_modify_state`; `nodes_to_calc_rate_for` is a list of nodes that might need their rates recalculated, and `nodes_to_calc_rate_for, nodes_to_modify_state` is a list of 3-tuples of the form (`node, previous_state, new_state`).

Concrete Pausers We now describe some of the pre-built Pausers that can be (and are, in most cases) used in the simulation:

`restriction_level_pauser()` A function returning a Pauser which raises (or lowers) the lockdown level by preforming a check at any given time. This check uses the given function and compares its

result to the given thresholds If the value is higher, the lockdown measures are increased (or decreased) accordingly.

While the behavior can be easily modified, a default implementation checking the total percentage of critically ill people, or alternatively, the percentage of used ICU beds within the system, is supplied.

`update_behavioral_state_pauser()` Routinely updates agents' behavioral states.

`expose_random_susceptibles` Takes an infection policy, a list of times and a number of people to expose (per disease, if multiple diseases are enabled) in the form of a `OpenSystem` object and exposes this many randomly selected (susceptible (to that disease)) people to the disease (at every given time). Can be used to "open-up the system" to unaccounted-for infections and introduce new infections at arbitrary times.

`vaccinate_individuals` Takes a `VaccinationProgram` (list of times, policy, and a number of people to vaccinate) and vaccinates this many randomly selected (susceptible (to that disease)) people to the disease (at every given time) with priority given according to the policy (age-based, degree-based, or uniform).

`population_testing_pauser` Takes a `PopulationWideTestingPolicy` and tests people for diseases every so often (based on the policy).

5.3.1 Simulation States

PersonState A class holding information regarding the person's current (disease-related) state in the simulation.

In order to allow greater flexibility (at the cost of slightly more computation performed at run-time) we use an *attribute-based* model, where each state is associated with various attributes rather than fixed compartments. In our case, each `PersonState` is an object which has three "fixed" fields: `dead`, `critical`, and `bed`, indicating whether the person is dead, whether they are critically ill, and whether they are using a hospital/ICU bed. In addition, each state has a `diseases` attribute, which is a collection of `PersonDiseaseState` for each of the diseases simulated in the system, where their per-disease state is recorded.

PersonDiseaseState This namedtuple-like object holds various attributes relating to a certain disease, such as whether the person is currently infected, if they are symptomatic, did they tested for the disease, the test result, etc. Consult the code for the current full list of attributes (which can be modified).

PersonStateRates and PersonDiseaseRates These hold the various rates that will be used (in the simulation) for this node. Each attribute of an `PersonState / PersonDiseaseState` can be associated with a rate, and a node's total rate will be the sum of those rates. When deciding which event happened to this node, a sub-event associated with an attribute is chosen with probability proportional to the corresponding rate (using the `pick_rate_at_random()` method).

ActivityState, ActivityStateRate, ActivityDiseaseState, and ActivitiyDiseaseRates These are simulation-states classes for Activities. As they are extremely similar to the Person simulation-states above, we will not describe them in great detail. However, it is worth mentioning that an `ActivityState` simply contains a tuple of `ActivitiyDiseaseState` (one for each disease in the system), and each `ActivityDiseaseState` has two fields: whether the activity is currently infecting, and whether it is currently closed (due to contact-tracing) for a "cleanup" process for a brief period of time.