Implementing EpiSimdemics in Haskell - Proposal

Collaborators: Andrew Schreiber ajs2409

November 22, 2021

1 Problem Statement

In the paper "EpiSimdemics: an Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks" [1] the authors describe an algorithm for predicting the spread of a virus through a given population, given a series of interactions between the members of the population. Traditional disease models work by simulating each event (a group of people, some of whom are infectious, at a location for a given period of time), in serial, constantly updating each individuals health status after each interaction. This means to compute the model, you either implement a global clock and process each event in order, or build a dependency graph of all the events and parse this graph as you compute the outcomes. Both of these models will be hard to parallelize and expensive to compute.

The novel idea in EpiSimdemics relies on a fact that we have seen play out throughout the Covid pandemic: diseases have a minimum latency period, D_{min}. Latency in this case refers to a period of time between when an individual is infected and when they are infectious (can infect other people). What this means from an algorithm perspective though, is that if you are at time t in your simulation, no one that is not already infected can become infectious until time $t + D_{min}$. This means that all the events in $(t, t + D_{min})$ are independent of each other, and hence can be processed potentially out of order, and in parallel. EpiSimdemics does just that, it works by iterating in intervals of $\Delta t < D_{min}$, and for each iteration, process all the events in parallel. Once that is complete, update each individuals health status (which can also be done in parallel), and then start the next iteration.

The goal of this project is to implement (a slightly simplified) version of the algorithm described in the paper in Haskell, and to be able to show the performance benefit gained by enabling parallelism in the algorithm.

2 Explanation of Algorithm

There are three main steps in the algorithm: transitioning each individuals health state according to a predecided disease model, computing the outcome of each event in the specified time window, and updating each individuals state based on the outcome of each event. As mentioned above, the algorithm works by iterating over intervals of Δt and within each interval performing the steps described in the below sections.

2.a Disease Model Health State Transitioning

When predicting the spread of a disease, there needs to be a model of how the disease progresses through an individual. In a simple model, the basic states are uninfected, infected and latent, infectious, and recovered. As we have previously discussed, EpiSimdemics relies heavily on the fact that the infected and latent period, lasts for a non trivial amount of time, usually 1-2 days. The first step per time period is to update each person, to make sure they are in the

appropriate health state for the next time period. I.E. if they have been infected and latent for the required number of days, they become infectious, and so on. The formal algorithm describes how someone can transition states part way through a time period, and in that case you split them up into two people, one for the time period where they are in the first state, and one for the second. To simplify the implementation for this project, I will not do this, and assume that people can only transition health states at the start of a new time period. I will use a simple model that roughly resembles Covid, using a 3 day latency period, and a 10 day infectious period.

2.b Event Outcome Computations

An event is a group of people in a particular location for a particular period of time. In order to compute the likelihood that a person, p_i, gets infected we first need define a few variables. Let τ be the length of the event, and N be the number of infectious people at the location. To resemble Covid, we will use τ in units of 15 minutes. Let r_j be the infectivity of the disease for person j, this is a percentage of how easy it is for a given person to spread the disease. To simplify the algorithm, we will assume infectivity is a constant, r. Let s_i be the susceptibility of person i, this is a percentage representing how susceptible person i is to getting infected. Let ρ be the transmissibility of the disease, which is how well the disease spreads from person to person. Some diseases spread in the air, like Covid, and are highly transmissible, while other diseases are less so. Putting all of this together we will use the following equation for determining the probability that person i gets infected with the disease, given they are in a room with N people for time τ:

$$p_i = 1 - \exp^{\tau N \ln(1 - r s_i \rho)}$$

What this equation looks like is for a fixed $y = \tau N$, if $y = 1$ (in a room with 1 person for 15 minutes), then as $r s_i \rho$ goes from 0 to 1, the percentage chance of getting infected increase linearly. For y between 0 and 1, the growth is exponential, meaning that you have a sub linear chance of getting the disease. As y increases past 1, the curve becomes logarithmic, steepening as y gets large, increasing your changes substantially of getting infected, even with low $r s_i \rho$.

This step of the algorithm will process each event, compute this equation for each person in the event, simulate the experiment to see if the person ended up getting infected, and store the outcome of that to a list of event of outcomes for each person.

2.c Post Events Per Person Processing

The post processing step is the most straightforward step. Given person i and a list of outcomes of the events for the person (which are 1 for newly infected, and 0 for not infected) just 'logical and' the results together and update the persons health state.

3 Experiment Design

One extra item worth noting, is that although in the papers associated with EpiSimdemics they describe performance stats on particular datasets that they generated, they do not publish the datasets. This will mean that I will have to generate the dataset for the simulation. My basic plan to do this is to pick a fixed number of people (hopefully around 1 - 10 million) and a

fixed number of locations (probably around 100K), and then for each person per day, pick a random number of events (uniformly distributed from 0-10) and randomly distribute them over the locations. Each location per person will have to have a time they are there, for this I will randomly assign a length of time, guaranteeing that no person is in two places at once. I plan to start off with a small dataset with less people and locations, and then adjust the parameters as necessary to get a dataset that is complex enough to see performance results of parallelism, but not too complex that it runs for hours on my machine.

References

[1] Barrett, Christopher L. and Bisset, Keith R. and Eubank, Stephen G. and Xizhou Feng and Marathe, Madhav V. *EpiSimdemics: an Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks*. SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing

[2] Talwar, Kunal, and Udi Wieder. *Overcoming the Scalability Challenges of Epidemic Simulations on Blue Waters*. 2014 IEEE 28th International Parallel and Distributed Processing Symposium, 2014