How Effective Distance Works to Predict Where and When an Infectious Disease Will Be

By: Kylie Elbert

Introduction

The aim for my project was to recreate the results from "The Hidden Geometry of Complex, Network-Driven Contagion Phenomena" by Dirk Brockmann and Dirk Helbing. They explore how air traffic contributes to the spread of modern infectious diseases and how major cities are connected around the world. Brockmann and Helbing illustrate that the crucial questions to answer after the outbreak of an infectious disease are: "(i) Where did the novel pathogen emerge? (ii) Where are new cases to be expected? (iii) When is an epidemic going to arrive at distant locations? (iv) How many cases are to be expected? (1337)"

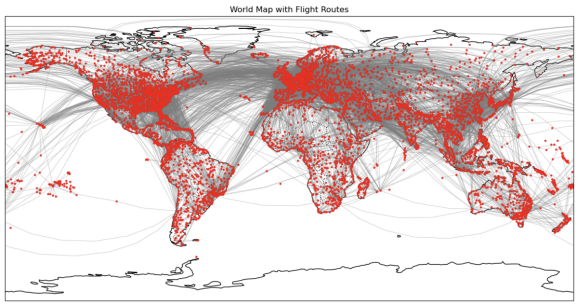
In the past, reaction-diffusion models have been helpful in answering their questions for cases like the Black Death in Europe. These models can provide a strong grasp on how diseases move and how fast they spread. However, they are not as helpful for modern diseases due to our ability to travel long distances. To address this, scientists have been developing complex computational models that take into account many parameters like population, how people move, and specific disease behaviors. "Models range from high-level stochastic metapopulation models to agent-based computer simulations that account for the behavior and interactions of millions of individuals in large populations" (Brockmann and Helbing 1337). These models all produce similar results despite making different assumptions and using different data. However, the models can't be combined because their foundational principles do not agree even though they are producing similar results. Due to this, the fundamental factors in an infectious disease's dynamics are unclear. Additionally, the introduction of so many parameters makes it difficult to see which factors are key in determining the dynamics. The models do not highlight which factors are relevant nor are they easy to initialize and calibrate which can lead to much uncertainty in results.

Brockmann and Helbing decided to create a more straightforward model that was a mix of reaction-diffusion systems and advanced computational models. Their method uses effective distance rather than geographical distance. Effective distance is based on the amount of air traffic between cities. A smaller effective distance means the cities are closer together which means there's more air traffic between cities. The separation of how diseases spread and how people move allows for Brockmann and Helbing to use their model to see clear wavefronts and answer the questions they introduced at the beginning of the paper.

Methods

Their model used the basic SIR dynamics between metapopulations in a Global Mobility Network. The point of the Global Mobility Network is to define all of the cities in the network and the direct flights between them to see how the metapopulations are connected.

I coded my simulation in Python. I started by gathering my data. I used the Global Air Transportation dataset from Kaggle, specifically their .csv files containing airports and routes. From the 'airports' file, I used each airport's IATA code, city, country, latitude, and longitude. The 'routes' file gave me direct flights from a source airport to a destination airport. I specifically used the IATA codes used to define the source and destination airports. I used these airports to define my Global Mobility Network.



After I had loaded the data, my second step was finding the populations of each city. I used a Python library called Geocoder that allowed me to look up cities' information like longitude, latitude, and population. I created dictionaries and numpy arrays to store my information for easier searches later. I created an array called 'cities' that contained every city in my Global Mobility Network where the indices would correspond to a city's index in future arrays. This was in the form 'City, Country.' My first dictionary, 'airport_city_lookup,' contained the airport IATA code as the key and the city and country also in the form 'City, Country' as the value. I iterated through the rows of the 'airports' database and added the IATA code with the city and country to the dictionary. However, some of the airports didn't have a city listed. They only had a country. Using the Geocoder Library, I used the airport name and its country to find a city. It took the name of the airport and looked it up in Geocoder. Geocoder returned a list of possible cities it could be. I iterated through the list to find the city where the country matched the airport's country. If the results list was empty then I removed a word from the end of the airport name. When a city was found with the correct country, I checked the city name. Some of them were empty, so I either used the city name it gave me or if it was empty, I used the airport name as the city knowing it would return a result if looked up in Geocoder. The next dictionary I created was to store the populations, called 'populations,' where the city and country, in the same format as the airports dictionary, were the key and its population was the value. I iterated through the 'cities' array and looked its population up using Geocoder. If it didn't return a population, then I assumed the population was zero. I had 7,144 cities in my Global Mobility Network.

I used the Global Mobility Network of direct flights between airports and my 'cities' array to create my P matrix. My P matrix was a 2D numpy array that was the number of cities by the number of cities. The columns corresponded to the source city and the rows were the destination city. The city's index in the 'cities' array corresponds to the city's index in the P matrix. The P matrix is defined as:

$$P_{mn} = F_{mn}/F_n \text{ where } F_n = \sum_m F_{mn}$$

F_{mn} is the number of passengers per day from the source node n to the destination node m and F_n is the total passengers per day that leave source node n. To find the P matrix, I first had to create what I call the F matrix. To do this, I iterated through all of the rows in the 'routes' database. I used the airport IATA code in 'routes' to look up the city, country in my 'airport_city_lookup.' From there I indexed the cities to see where they would go in my F matrix. I used average passengers for domestic and international flights from Quora to add the number of passengers from the source city to the destination city index in my F matrix. Once I has total passengers from all sources to all destinations, I summed the columns and divided each column by its sum to create my P matrix because the P matrix contains the proportion of passengers from each source going to the destination. While I was doing this, I found the total passengers per day, ϕ, and the total population in the system, ω, by summing my F matrix and summing all of the populations from my 'populations' dictionary. This gave me my average mobility rate, γ, which I would use later.

I wanted to visualize my air travel on a map, so I used the cartopy Python library. I

started by creating a simple NetworkX graph where each airport was a node and an edge signified a direct flight between the airports. This led me to my next dictionary, 'airport_locations,' that contained the IATA code as a key and its latitude and longitude, stored as a tuple, as the value. I used cartopy, the graph, and the dictionary to visualize air travel on a global map. My network contained 6,236 airports with 19,257 connections.

The next part was to calculate the effective distance between each city. My d matrix stored all of those effective distances. $d_{mn} = (1 - \log P_{mn}) \geq 1$. My d matrix was the same size as my P matrix and used the same indices for the cities so that everything would match. For each index in my P matrix, I calculated the effective distance if the value in the P matrix was greater than zero. If there was no passenger flu between the cities, then I used the effective distance 10.0 to signify that. After that, I created a directed and weighted NetworkX graph to connect all of the cities based on their effective distance. If the effective distance was less than ten, then there would be an edge between the cities with the weight being their effective distance.

Once I had all of the data organized for my simulation, I could code the simulation. While Brockmann and Helbing were using an SIR model, instead of using $S,\ I,$ and R to represent the number of susceptible, infected, and recovered in a population, they used s_n to represent the proportion of susceptible people in a population, j_n to represent the proportion of infected people in a population, and r_n to represent the proportion of recovered people in a population where n is a population in the Global Mobility Network.. To store j_n, s_n, and r_n, I created a dictionary for each one where the city, country would be the key and an array that was the length of the number of timesteps would be the value to store the proportion at each timestep. To try to replicate Brockmann and Helbing's simulation, I used their parameter values. R_0 is the basic reproduction ratio and is equal to α/β where α is the mean rate at which an individual infects others and β is the mean recovery rate of individuals. Their $R_0 = 1.5$ and their $\beta = 0.285$, so I used those values. The equations they used to calculate the rate of change for j_n and s_n were defined as:

$$\partial_t j_n = \alpha s_n j_n \sigma(j_n/\varepsilon) - \beta j_n + \gamma \sum_{m \neq n} P_{mn}(j_m - j_n)$$

$$\partial_t s_n = -\alpha s_n j_n \sigma(j_n/\varepsilon) + \gamma \sum_{m \neq n} P_{mn}(s_m - s_n)$$

The sigmoid function and r_n are defined as:

$\sigma(x) = x^{\eta}/(1 + x^{\eta})$ and $r_n = 1 - j_n - s_n$.

I continued to use their values for ε and η which were 0.000001 and 4. My function took in the outbreak location, a shortest path tree where the source is the outbreak location, positions for the nodes to visualize the tree, and the number of timesteps. Set variables in my function were the 'cities' array, the P matrix, the 'airport_city_lookup' dictionary, and the 'airport_locations' dictionary. This is because I used visualization functions to visualize the infected cities using the shortest path tree and the infected cities on a map. To visualize the infected cities in the form of a shortest path tree, I had to create a shortest path tree based on the outbreak location. I used Dijkstra's algorithm to find the shortest path to the outbreak location based on my NetworkX graph that connected cities based on effective distance. I went through the algorithm and added only the edges included in each city's shortest path to the source to my shortest path tree. To get my visualization, I created levels that ranked cities based on how many edges it needed to be connected to the source. I attempted to use trigonometry to keep it all organized and create a circular graph with

the source at the middle. The nodes were at a distance equal to their distance from the source which was the sum of the effective distances to get to the source. The reason my simulation function had default arguments was because I needed them for my visualizations. My first visualization took in the shortest path tree and its positions for nodes and drew the graph by coloring the infected nodes. This made it easy to see which cities were being infected first and how close they were to the outbreak location based on their effective distance. The second visualization was a global map with all of the cities where the infected cities were colored. This didn't highlight the effective distance, but it helped to show where the cities were. Back to my simulation, I initialized no infection in my j_n, s_n, and r_n dictionaries. Then, I initialized an outbreak location. After some trial and error I realized I needed a sufficient number of initially infected people in my outbreak location. The paper didn't say what they used for their simulation pandemic, so I used 500 people. To be able to analyze my results of when each city became infected, I used an array where each index corresponded to the city's index in the 'cities' array as well as my matrices. It would hold the day in the simulation that the city became infected. I also created an array just to keep track of which cities were already infected. Next, it was time to start my simulation. On day 0, only the outbreak location was infected. On day 1, I started running the simulation. For each city, I calculated its rate of change for j_n and s_n using the equations above. Once I did that, I used the rate of change to calculate its new j_n, s_n, and r_n. If the cities had a flight connecting them, then they would automatically have an infected and susceptible proportion of the population, so I checked if the infected and recovered population would be equal to at least one person. If it was, then the city was infected. If it wasn't, then I assumed that the city hadn't yet been infected but that the infection was on its way. After the new infected, susceptible, and recovered proportions had been calculated for the time step, I ran my visualizations to see what the infection looked like. The simulation returned the array that contained each city's infection day. If the city hadn't been infected, then it had a value of -1 in the array.

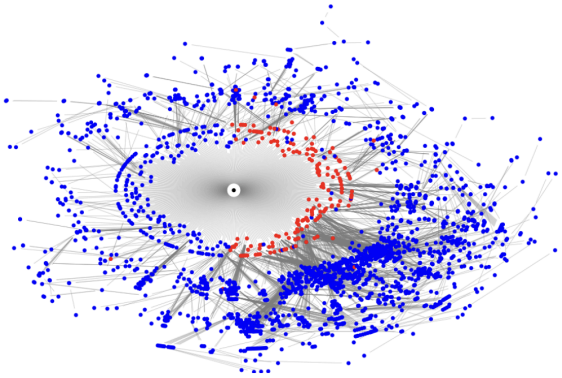
Results

When Brockmann and Helbing looked at their results, they compared R^2 for the line of best fit for effective distance versus arrival time and geographical distance versus arrival time. They found not a lot of variance for geographic distance and found that it was not a good fit for predicting when an infection would reach a city. I found the same thing. When I was looking at my Global Mobility Network, I found that London, United Kingdom had the biggest passenger flux, so I wanted to use it in my simulation. Brockmann and Helbing used Beijing, China, so I decided to use that city as well. I ran the simulation on both cities and analyzed my results.

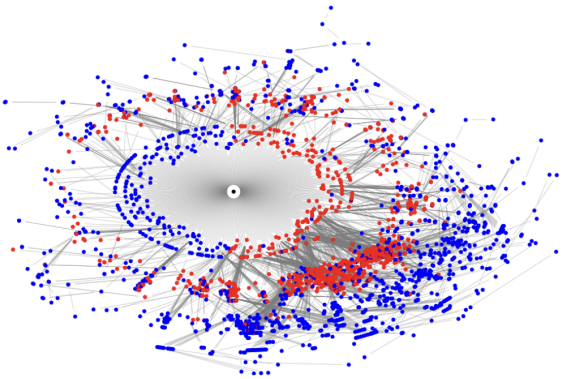
For both cities, my shortest path tree showed that nodes with a larger effective distance were getting infected before some of the nodes with a shorter effective distance. I thought that maybe because I was using my condition to see if the proportion of infected and recovered were equal to at least one person in the simulation could have been affecting that. The cities with shorter effective distances might have larger populations, and due to the small rate of change for infected and susceptible, the infection might not be showing up yet. However, that is why the equation multiplies $j_m - j_n$ and $s_m - s_n$ by P_{mn}. In theory, larger cities probably have more passenger flux between them. I also thought about that. It is possible that I was not using the most accurate data. I wondered if my 'routes' database had multiple flights between

cities or if it just counted one flight per source and destination. So while a city can have multiple airports that would allow for more passenger flux between it as a source and a destination, that might not be enough to offset only counting one flight "per day."

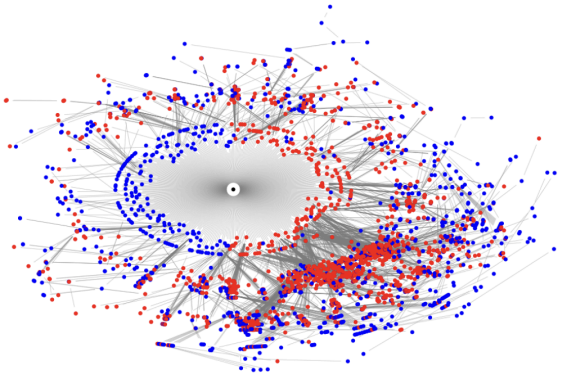
London, United Kingdom's progression of effective distance shortest path tree:
Day = 40



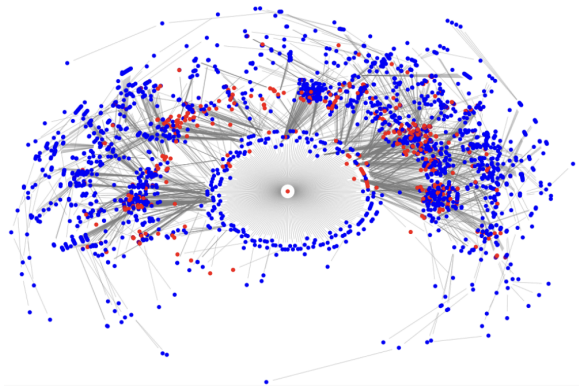
Day = 80



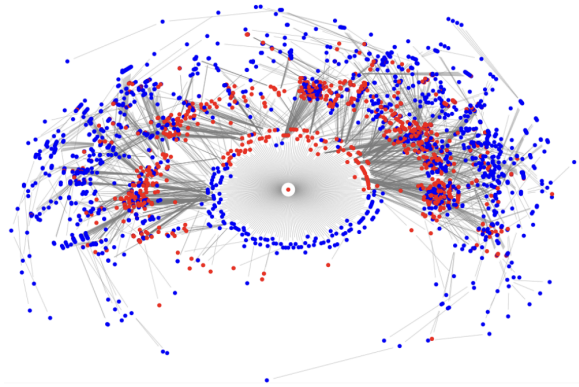
Day = 149



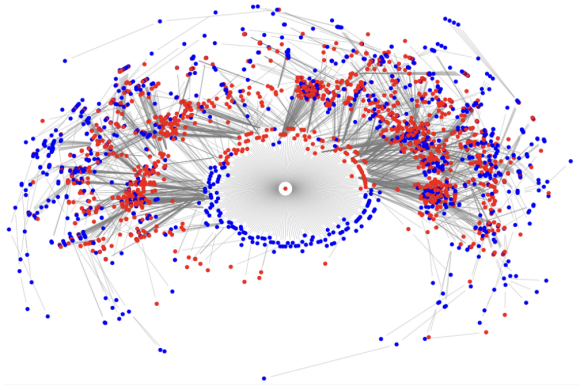
Beijing, China's progression of effective distance shortest path tree:
Day = 70



Day = 100

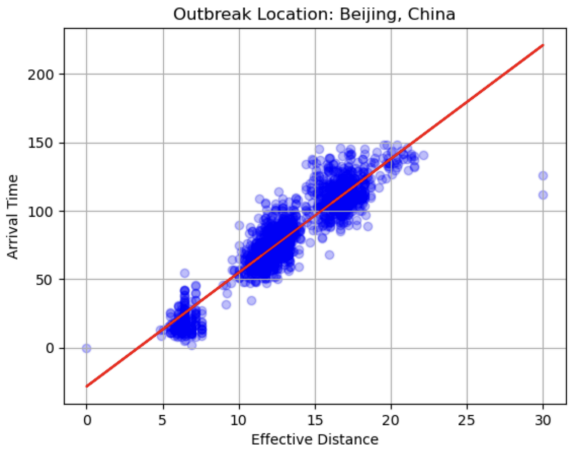


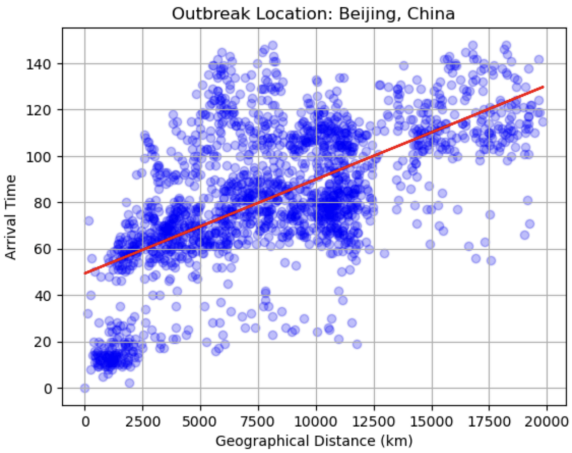
Day = 149



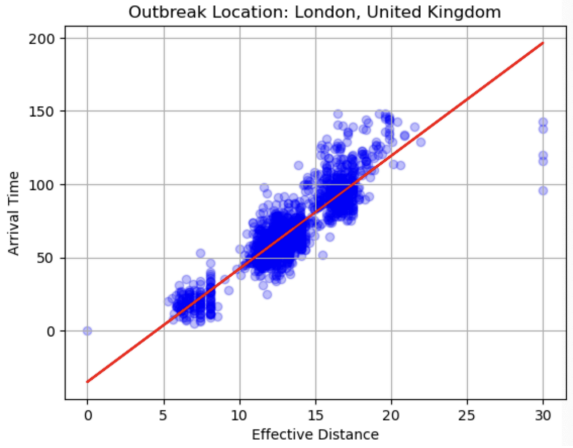
Even if my data wasn't as accurate as Brockmann and Helbing's, I still got similar results as them. Once I had run my simulation, I graphed effective distance and geographical distance versus arrival time. I only used cities that had been infected. With that information, my R^2 for effective distance was significantly

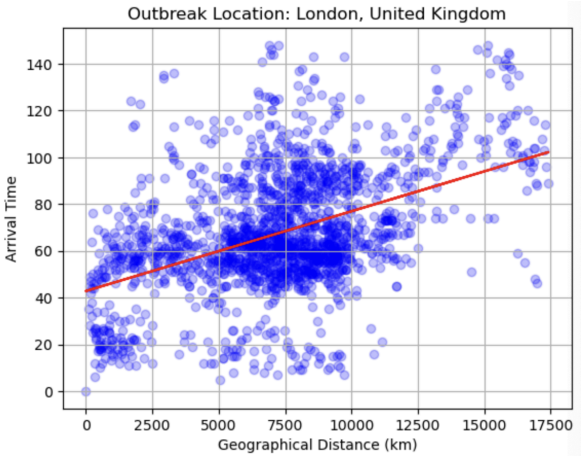
higher than my R^2 for geographical distance. For Beijing, China, effective distance versus arrival time had an R^2 of about 0.85 and geographical distance versus arrival time had an R^2 of about 0.38.





For London, United Kingdom, effective distance versus arrival time had an R^2 of about 0.79 and geographical distance versus arrival time had an R^2 of about 0.21.





Additionally, in their paper, Brockmann and Helbing noted that a world view of infected versus uninfected cities was not helpful in seeing wavefronts and determining outbreak location. I used it to show that even if countries had a larger geographical distance, they could still be infected well before cities closer to the outbreak location. What was really meant to show how the infection was spreading was the shortest path tree based on effective distance. You can see clearer paths in the trees.

Discussion

There were a few problems that I ran into. The first being that I was doing this without spending money. I only had so many Geocoder credits. I had to wait for them to reload before I got all of my data. This is why I started storing my data in dictionaries and arrays. To avoid this problem when mapping out infected cities, I

used the latitude and longitude of a random airport in the city. I believe this still produced an accurate world map.

Another problem I ran into was organizing the shortest path tree. I used trigonometry and effective distance to try to get accurate positions for the cities. I believe that in the end I was able to create a somewhat accurate representation specifically because I used effective distances to place nodes at the correct coordinate to demonstrate their effective distance from the outbreak location.

Other problems like city populations being too big to be affected by the rate of change and the ‘routes’ database have already been discussed. Even with the problems that I faced, I seemed to get very similar results as Brockmann and Helbing. Overall, effective distance is a much better match for predicting arrival time and has a lot less variance than geographical distance.

Bibliography

Brockmann, Dirk, and Dirk Helbing. “The hidden geometry of complex, network-driven contagion phenomena.” *Science*, vol. 342, no. 6164, 13 Dec. 2013, pp. 1337–1342, https://doi.org/10.1126/science.1245200.

Devastator, The. “Global Air Transportation Network.” *Kaggle*, 6 Dec. 2022, www.kaggle.com/datasets/thedevastator/global-air-transportation-network-mapping-the-wo/data?select=routes.csv.

“GeoNames.” *GeoNames - Geocoder 1.38.1 Documentation*, geocoder.readthedocs.io/providers/GeoNames.html. Accessed 5 May 2024.

“Geonames.” *GeoNames.Org*, www.geonames.org/v3/. Accessed 5 May 2024.

What Is the Average Amount of Passengers on a Plane? - Quora, www.quora.com/What-is-the-average-amount-of-passengers-on-a-plane. Accessed 5 May 2024.

Appendix- Code