Chapter 1

Distributed Random Walks on Dynamically Weighted Graphs

Contributed by Trenton W. Ford

1.1 Introduction

In 2014 the world saw the most massive resurgence of the Ebola virus in history. The epidemic was mainly localized to West Africa but spread to other parts of Africa, and even other countries through a myriad of transportation methods. All told, nearly 30,000 people were infected worldwide, of which 11,300 died. These numbers are terrible, but they could have easily been worse. A large part of planning against the transmission of epidemics is in modeling the dispersal of infection through physical travel networks. In 2014 this modeling helped the Centers for Disease Control (CDC) increase monitoring on select ports to slow or prohibit the possibility of widespread transmission. The goal of this paper is to investigate models of disease transmission that utilize Random Walks as their underlying kernel.

1.2 The Problem as a Graph

We can formalize the above problem as a complex, heterogeneous network with the following structure:

Node Types :

1. Airport
2. Seaport
3. Rail Station

Edge Properties :

1. Direction
2. Transition Probability ($p_1 ... p_k$)
3. Distance
4. Travel Time

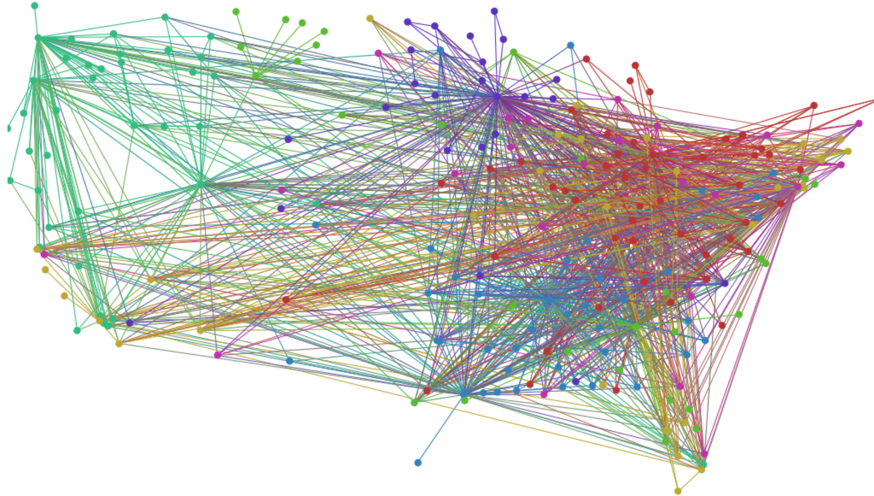

Figure 1.1: The US Domestic Flight Network[4] - a network containing only nodes of type *airport*, with edges weighted and colored by the frequency of travel between two airports.

1.3 Some Realistic Data Sets

1.3.1 Example Data Sets

Table 1.1: Potential datasets where order is the number of vertices, and size is the number of edges.

Dataset	**Approximate Order**	**Approximate Size**
DOT Railway Data	196K	250K
SNAP Airport Data	456	71K
KNB Shipping Data	3700	15K

Transportation networks, in general, are pretty ubiquitous. The SNAP Labs[1] usually maintain stable data repositories for network data, and here is no exception. The **Airline Travel Reachability Network** contains both US and Canadian airport travel nodes and edges, including plenty of metadata such as travel time, geodata, and other potentially useful things.

Maritime data is more difficult to find, but there are multiple years worth of shipping data maintained by The Knowledge Network for Biocomplexity (KNB)[1], and increasing amounts of data in this space is becoming open access. Railway datasets are less convenient as the datasets are

[1] [3]https://snap.stanford.edu/data/reachability.html

generally maintained by the respective governments for which the railway belongs. For instance, US railway data can be found at either Data.gov, or the Department Of Transportation data page[2].The size and order of these datasets can be found in table 1.1.

1.3.2 Constructing A Complex Network

If we were to construct a complex network from the above transportation network datasets, we would find that the size and order of the network would increase greatly. For instance, if an airport terminal also contains a train terminal (not irregular) then the airport and train station nodes must either merge, or copy the inbound and outbound edges of their counterpart. This sort of node merging or edge duplication activity results in quite irregular networks. This irregularity, coupled with the network's heterogeneity, makes it difficult to build network generators that sufficiently capture the depth of interactions and metadata represented in real-world data.

Moreover, as stated above, many of the datasets are maintained by separate entities and represent transportation networks that serve different goals. For instance, the difference between passenger trains and freight trains means a great deal when attempting to capture human vectored disease transmission. For these reasons, interested parties such as the World Health Organization (WHO) and the CDC are working with countries all over the world to increase the accuracy of and access to this data.

1.3.3 Generating Representative Datasets

Given the difficulties discussed above related to wrangling the real-world datasets, we chose to use synthetically generated datasets of several types. We used several sizes of grid networks where vertices were given random initial populations, the edges outgoing from a vertex is set as a proportion of the vertex population and out degree. A vertex is then selected at random and some small portion of their population is infected initially.

When using graphs that are not grid structured, ErdsRnyi for example, because the vertex degrees is irregular we use them to inform the vertex population size. Greater degree vertices have greater population sizes. A vertex is still selected to have an initial infected population, but instead of random uniform sampling, the sampling is weighted by vertex population. Higher population vertices are more biased towards having the initial observed infective population.

We plan to investigate other real-word adjacent graph generation techniques in further work.

1.4 Random Walk-A Key Graph Kernel

A random walk on a graph, in simple terms, is a sequence of traversals(movements) from one vertex over and edge to another vertex. When a problem can be reasonably represented as a graph, a simple random walk on that graph represents unbiased movement through the graph where only structure is considered. For this reason, random walks are often used to determine if real observed traversal through a network is being influenced by non-structural factors. In this way, random walks serve as a traversal baseline given no prior knowledge of the graph.

From the baseline of the truly random walk where traversal from a node v to some node x_1 is $1/deg(v)$, we can add prior knowledge to our graph structure, and we can to test if our prior knowledge accounts for the differences in real observed traversal data and the random walk's traversals. For example, of the likelihood of traversing from one vertex to another is known apriori,

[2] osav-usdot.opendata.arcgis.com/

so we can add that knowledge to our network and when traversing edges with this likelihood information we could choose to bias for or against traversing over those edges based on the likelihood values. The closer we are able to get our model to fit the real data the better. In this way we can begin at a simple random walk model, and build it to fit our application. Once a good model is trained the goal would be using that model to better understand the real traversals observed in the network through understanding the model, and using the model to make predictions.

In the disease transmission scenario described above, often probabilistic random walks are used to represent individuals (infected or otherwise) moving through travel networks. Over time these random walks help tell the story of the ports that these individuals pass through and in doing so, give clues to health authorities on which ports to close or monitor more closely[5].

When considering all major travel networks in the world, the size of the networks can become time and space prohibitive. In the normal case of random walks this can be overcome with an “embarrassingly parallel” variant of random walks effectively copies the network and runs separate walks on many different machines at once. The results of the separate random walks are then merged into a usable solution. Unfortunately, an active transmission scenario presents a dilemma for distributing the random walks; edits the underlying graph(port transmission potential), or the walker itself carries some accrued information(transmission potential) means that node and edge state is no longer guaranteed to be consistent across distributed random walkers.

1.4.1 Proposed Solution

This research proposes to extend the random walk kernel using a network specific distribution scheme that considers the interconnectedness of the network to determine how to distribute the network to compute nodes maintaining their own random walkers, and suggests an optimizing function to shift or duplicate nodes based on the minimization of communication.

1.4.1.1 Network Partitioning

Unlike the standard parallel random walk algorithm, we cannot arbitrarily partition our graph and distribute the partitions to distinct compute nodes and merge their individual results. The dynamism of the disease transmission random walk algorithm requires that changes to the underlying network such as edge and vertex properties are expected. Moreover, the compute nodes themselves can change the underlying graph attributes. Optimally done, this partitioning would minimize the number of messages passed between distributed random walkers. To approximate this optimal partition, metrics such as min-cut and spectral clustering will be tested in small networks against a baseline of randomly distributed nodes.

1.4.1.2 Communication

Communication will be handled by the Parallel Boost Graph Library’s process group. I’ll expand upon this in the next revision.

1.5 Prior and Related Work

1.6 A Sequential Algorithm

One of the major advantages of the random walk process is that it is easy to understand in the uniformly weighted, non dynamic case. The algorithm is relatively simple. The input is a graph

$G(V, E)$ where the edges $e \in E$ have associated weights, or transition probabilities w_i. If we consider a simple case where we select one starting vertex say u, and wish to perform a random step we would consider all adjacent vertices, say $x_1 \dots x_k$, connected to u over edges $e_1 \dots e_k$, with associated weights $w_1 \dots x_k$. Before taking a step, the weights of the adjacent vertices must be normalized as follows:

$$p_i = \frac{w_i}{\sum_1^k w_i}$$

Where each of the p_i will represent the probability of traversing across a given edge e_i. Given the normalization process, we are guaranteed that the probability of all adjacent edges will sum to 1, so we can use a standard uniform random number generator to sample from the edges. At that point we may successfully take a step. We can take as many of these steps as are available, and upon completion we are returned a vector containing the path of the random walk, starting with vertex u.

Algorithm 1 Simple Random Walk

```
Data: Graph G(V, E)
Result: Path Of Random Walk
Select A Start Node u Randomly
 Initialize path vector P.append(u)
 while True do
   edges=u.adjacent
    if edges.length > 0 then
      edges.prob = normalize(edges.weights)
       u = uniformSelect(edges.prob)
       P.append(u)
   else
      return P
   end
end
```

To tailor this algorithm to the requirements of the disease transmission model we introduce variables mirroring those used in Kelker's seminal work[2] published in 1973 that used a simple grid model and a minimal set of transition parameters to model the spread of measles and ferret distemperment. Put simply, the model used a 2x2 grid of vertices within which 4 infective individuals were located at time zero. At each successive time step each individual has an equal opportunity of staying in place, or moving to one of the adjacent nodes. Let this probability be λ. If an individual moved to a vertex that contains uninfected, or susceptible individuals, there is a probability p that each may become infected. The paper also introduced a variable $\mu$to represent the probability that at each time-step a person recovers, or changes from an infective state to a susceptible state.

Within our sequential implementation we modify Kekler's algorithms such that the graph is no longer a 2x2 grid, but graphs generated from real-world data and graph generation models. We also improve accuracy by λ, μ, and p be sampled from distributions instead of being variables set based on expert opinion or solely on prior data.

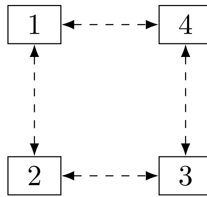

Figure 1.2: Kekler's 2x2 Grid Graph For Epidemic Simulation [2]

1.6.1 Pseudocode

Listing 1.1: Selecting a Vertex to be Infected

```
1  // Define a grid_graph where the second dimension doesn't wrap
2  boost::array<std::size_t, 2> lengths = { { DIMENSIONS, DIMENSIONS } };
3  boost::array<bool, 2> wrapped = { { false, false } };
4  GraphType graph(lengths, wrapped);
5
6  // Get the index map of the grid graph
7  typedef boost::property_map<GraphType, boost::vertex_index_t>::const_type indexMapType;
8  indexMapType indexMap(get(boost::vertex_index, graph));
9
10 // Create a float for every node in the graph
11 boost::vector_property_map<VertexProperties , indexMapType> dataMap(num_vertices(graph), indexMap);
12
13 // INIT VERTEXPROPERTIES
14 for (uint i = 0; i < DIMENSIONS; ++i)
15   for (uint j = 0; j < DIMENSIONS; ++j)
16       put(dataMap, Traits::vertex_descriptor {{i, j}}, VertexProperties{i,j,1});
17
18 // SELECT INFECTED
19 Vertex init_infected = random_vertex(graph, rng);
20 auto init_infected_vp = get(dataMap, init_infected);
21 init_infected_vp.set_current_values(1,1,0);
22 put(dataMap, init_infected, init_infected_vp );
```

I'm still working on writing some pseudocode for the actual sequential algorithm, putting the C++ code here would be terrible to look at.

1.7 A Reference Sequential Implementation

The current sequential implementation generates grid networks due to their regular network structures as the order of the network increases. The aim of this grid based implementation is to regularize the scaling results and make easier the comparison between the sequential and the parallel implementations of the disease transmission model.

For the sequential representation, our implementation will roughly copy that put forth by Kelker in 1973[2]. Modifications were to increase the accuracy of the model by allowing the three variables (λ, μ, and p) to be samples from non-uniform distributions. We also allow graphs that are not grids. Kelker's simulation model was the groundwork for more recent epidemic transmission simulations, but the complexity that they add to the random walk process makes it difficult to do meaningful complexity analysis of the algorithms.

We also adopt Kelker's stopping criteria, in that we run the model until all of the individuals reach a homogeneous state. That is to say that either all people are infective, or all people are susceptible. Reasonable thresholds can be used instead of absolute convergence, but for the parameters that we set, absolute convergence should always be possible.

1.8 Sequential Scaling Results

1.8.1 Experimental Configuration

All scaling tests were performed on an Intel Core i5-4278U @ 2.60GHz with 3MB of cache, and 16GB of DDR3 RAM. The sequential version of the algorithm was implemented using The Boost Libraries version 1.68. The application was compiled using GCC version 8.1.

1.8.2 Stopping Criteria: BFS

Still working on plots

1.8.3 Stopping Criteria: DFS

Still working on plots

1.8.4 Stopping Criteria: Vertex Cover

When discussing the scaling results of the sequential random walk implementation, what must first be imposed is a *stopping criteria*. Without a stopping criteria a random walk has no definite end state. For the purposes of the results here, the algorithm will stop once all nodes have been visited at least once, also known as the *node cover* stopping criteria. Even with this imposition, it has been shown that the speed of random walks changes with the structure of the network being traversed[6]. This limits the inferences that can be gleamed from the results from different graphs in relation to each other, but during the parallel implementation results analysis we can compare the two algorithms directly over each of the networks.

Still working on plots

1.8.5 Stopping Criteria: Homogeneous State

Still working on plots

1.9 An Enhanced Algorithm

1.10 A Reference Enhanced Implementation

Forthcoming.

1.11 Enhanced Scaling Results

Forthcoming.

1.12 Conclusion

Forthcoming.

1.13 Response to Reviews

From initial responses, I added within the introduction a bit about how the random walk kernel is used to help model epidemic disease spread and what the paper aims to do. I made some grammatical edits based on feedback, but given that I've written much more and I'm still in the drafting process I imagine there are many more grammatical changes that need to be made. I made a clarification with the table containing the potential datasets. I will still need to go through and make sure to better define terms such as order, size, and transmission potential. I attempted to better explain how the kernel helps solve the problem by referencing a simple previous work by Kelker quite often since it is easily comprehensible. It seemed to go over well during the most recent presentation, so I imagine it should help here as well.

There are more recent works that I still need to work into the paper somehow as well.

Bibliography

[1] John Potapenko Kenneth Casey Kellee Koenig et al. Benjamin Halpern, Melanie Frazier. Knowledge network for biocomplexity, 2015.

[2] Douglas Kelker. A Random Walk Epidemic Simulation. *Journal of the American Statistical Association*, 68(344):821–823, 1973.

[3] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[4] Elijah Meeks. Visualization of network distance, Nov 2011.

[5] Krista C. Swanson, Chiara Altare, Chea Sanford Wesseh, Tolbert Nyenswah, Tashrik Ahmed, Nir Eyal, Esther L. Hamblion, Justin Lessler, David H. Peters, and Mathias Altmann. Contact tracing performance during the ebola epidemic in liberia, 2014-2015. *PLOS Neglected Tropical Diseases*, 12(9):1–14, 09 2018.

[6] Blint Virg. On the speed of random walks on graphs, 1999.