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 for the transmission of epidemics is in modeling the dispersal of infection through the physical travel networks, in 2014 this modeling helped the Centers for Disease Control (CDC) close, and increase monitoring on select ports to slow or prohibit the possibility of widespread transmission.

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

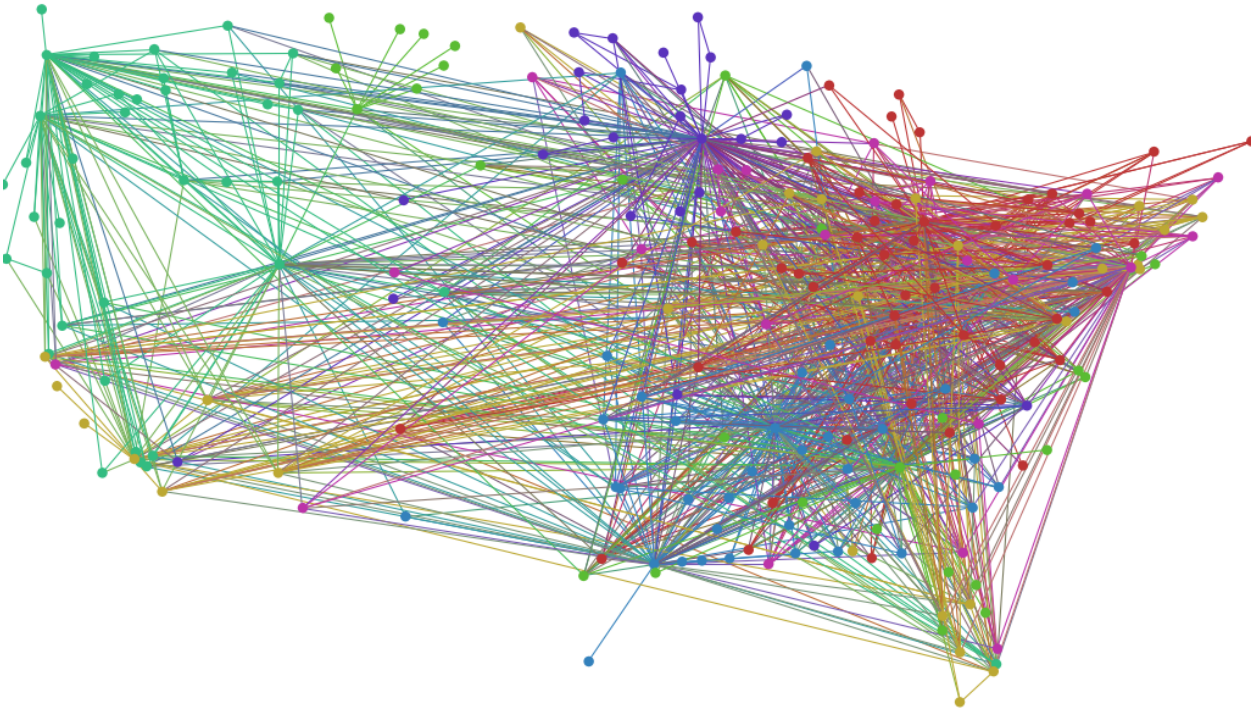

Figure 1.1: US Domestic Airport Network[3]

Potential Datasets

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

Figure 1.1 shows 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

Transportation networks, in general, are pretty ubiquitous. The SNAP Labs[1] are 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 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 1.3.1.

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.

1.4 Random Walk-A Key Graph Kernel

In the disease transmission scenario described above often to build these transmission models, probabilistic random walks are used to represent individuals (infected or otherwise) moving through the travel networks. Over time these random walks help tell the story of the ports that these

[1] [2]https://snap.stanford.edu/data/reachability.html
[2] osav-usdot.opendata.arcgis.com/

individuals pass through and in doing so, give clues to health authorities on which ports to close or monitor more closely[4].

When considering all major travel networks in the world, the size of the networks can become time 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 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 embarrassingly parallel version of a random walk, to reduce memory usage and minimize the updating required to keep all network stores consistent our algorithm partitions the network into distinct subgraphs and distributes those subgraphs to each separate random walker. 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

This is incredibly tentative! When a random walker r_1 on a single partition p_1 attempts to traverse an edge that leads to another partition, say p_2, r_1 must communicate this intent to r_2 and r_2 determines how to handle the intent and respond to r_1. During this time, r_1 busy waits for a response from r_2. To avoid deadlock, if r_2 receives a new message while handling the current message, it will *Do Something Surely.*

1.5 Prior and Related Work

Forthcoming.

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

1.7 A Reference Sequential Implementation

To compare most evenly with the parallel version of the code, the sequential implementation was written with the Boost Graph Library using no parallel options. The Parallel Boost Graph Library will be used for the parallel implementation of the code.

1.8 Sequential Scaling Results

Discuss here results from your sequential implementation. Include software and hardware configuration, where the input graph data sets came from, and how input data set characteristics were varied. Did the performance as a function of size vary as you predicted?

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[5]. 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.

Results from the sequential implementation are forthcoming, dealing with some slippery errors during optimization.

1.9 An Enhanced Algorithm

Forthcoming.

1.10 A Reference Enhanced Implementation

Forthcoming.

1.11 Enhanced Scaling Results

Forthcoming.

1.12 Conclusion

Forthcoming.

1.13 Response to Reviews

Bibliography

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

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

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

[4] 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.

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