CS224W Project Final Report

Yuetong Wang, Jack Jin, Alice Zheng
Stanford University, Stanford, CA 94305
yuetong@stanford.edu, jackjin@stanford.edu, alicezhy@stanford.edu

Dec 2018

1 Introduction

Transportation system is one of the most common types of networks we interact within daily life. Neither economic growth nor technological revolution could happen without a properly designed transportation system. In this project, we are going to explore the railway system in China for passenger transportation and investigate its robustness. In particular, we designed a possible criterion for measuring that robustness: Given the list of trains, and a number k, if we are allowed to destroy a maximum of k railroad blocks, what is the maximum influence that we could get? We will investigate some possible algorithms for calculating this criterion, compare their results.

2 Related Work

There are plenty of existing work discussing the robustness of transportation systems, but work investigating the robustness of a railway network is a bit lacking. Therefore we looked at papers on the robustness of general transportation for inspiration. For example, *Robustness Tests for Public Transport Planning*[1] argues that the traditional bi-criteria of cost and quality do not suffice the need to evaluate a transportation planning system and introduces a third criteria to measure robustness. The paper proposes three different robustness tests that can be conducted on a transportation schedule, and discusses their intricacies with experiments. Although the paper discusses a completely different transportation system and focuses on planning, the idea of a robustness test helped us to formulate our own research question.

Additionally, we also looked at papers on graph clustering and optimization algorithms to help with the design of our own algorithms. For example, *Graph based k-means clustering*[2] discusses the running graph clustering using the k-means algorithm as an alternative to other methods such as minimal spanning trees. Clustering algorithms could reveal the underlying

structure of our network, and help us in finding a solution to our research question. Furthermore, since our question is essentially an optimization problem, we also looked at papers on optimization algorithms. *Markov chain Monte Carlo methods applied to graph clustering*[3] discusses the application of MCMC algorithm on graph clustering and flow optimization, and shows that MCMC can not only reduce the computation time for such problems but also potentially produce better results.

3 Define the context

3.1 Define the "railroad map"

The railroad map is an undirected graph $G = (V, E)$ with V indicating all the stations and E indicating all the "primitive railroads". An edge (u, v) exists if and only if there's a block of railroad directly connecting the two stations u and v, with no other stations in the middle. The weight of an edge is defined to be the length of this railroad block. In the **Data Collection and Preprocessing** section, we will explain on how to estimate this value.

3.2 Define the "trains"

Trains could be viewed as **simple paths** on the graph. In the dataset we have, we could only see all the stations that the specific train **stops** at, but it could actually **pass** many other stations between two stops. Between two stops, we assume that the train follows the path with shortest distance (if there are ties, randomly pick one, and ties rarely happen in reality).

4 Define the Problem

Here is the problem that we are tackling in the next several sections: Given the list of trains, and a number k, if we are allowed to destroy a maximum of k railroad blocks, what is the maximum influence that we could get?

To further clarify the problem:

(i) Each train has an "importance", which is defined to be the time for a complete trip times a **speed coefficient**. The speed coefficient is a constant for each type of trains - this is indicated in the name of that train. For example, a train starts with **G** will have a higher **speed coefficient** than a train starts with **T**. The reason for not using the distance travelled is that we don't have a precise dataset about this.

(ii) Each train has a fixed route (a **simple path** on the graph). If any edge on this path is destroyed, the whole train is affected.

(iii) The total influence is the sum of importance of all affected trains.

We believe this problem to be interesting for a variety of reasons:

(i) It is a good measure of the vulnerability of the system. There are many algorithms targeting at measuring vulnerability based on the single-failure assumption (e.g. Bi-connected component detection for the detection of bridges and articulation vertices). However, we want to extend the scope to a multiple-failure case. It can serve as a metric to evaluate "how should we make this system more robust".

(ii) Both the railroad map and the trains are engaged in this problem, making the result more realistic. It's hard to tell the importance of different parts of the railroad map by looking at the map itself; however, trains give us more information, as the density of trains will indicate the importance in general.

(iii) The problem is not trivial. If custom railroad map and trains are allowed, it's not hard to prove that the original problem is NP-complete, by reducing it to the vertex-cover problem. We're looking for approximation algorithms.

5 Data Collection and Preprocessing

5.1 Recovery of the Railroad map

We obtained two sets of data online. The first is a list of all the Chinese railway stations, including the station name and the geographical location (in longitude and latitude). Notice: in China, no two railway stations have the same name. The second data set is s list of all trains currently in operation. For each train, there is a list of all the stations visited in order, and the time needed to travel between any consecutive pairs of stations. Then we recover the railroad map using the algorithm below. Notice that this is not meant to be an exact recovery, but should be a good enough approximation:

(i) Iterate through each train. Assume that the current train has t stops $a_1, a_2, \cdots, a_t$. We add edges (a_i, a_j) for all $i+1 < j$ to a set T. The edges in T means that "potential pair of consecutive stops of some train, but they **do not** form a primitive railroad block" (because we already know that there's a station in the middle).

(ii) Iterate through each train again. Now, for each pair of consecutive stops (a_i, a_{i+1}), add an edge between these two stops if and only if it does not appear in the "impossible set" T.

This algorithm is based on the assumption that, for each railroad block (u, v), at least one train stops at both stations. This is a reasonable assumption given that there are many local trains. This assumption also facilitate the process of estimating the length of a block of road - proportional to the minimal possible time for a train to travel from u to v (this information is available - we know that when will each train stop at each station). The estimation of length is not that ideal, given the design of China's railway system - some railroads are

dominantly for High-speed trains, while some railroads do not have High-speed trains at all. However, this is the best estimation we could come up with.

The result map we got is pretty sparse. It has 2576 vertices and 3075 edges. Figure 1 demonstrates the degree distribution of this graph. Given the nature of the railroad map, it's not surprising that all vertices have super low degree; most of them actually have a degree of 2, indicating that they are just "chain" together.

5.2 Track the route of a train

This problem could be reduced to a pairwise shortest path algorithm, according to the assumption that each train will travel the shortest path between two consecutive stops. Here's a slight variation of this idea so that it facilitates us in solving the problem:

(i) Calculate the pairwise distance using Queue-based Bellman-Ford. The reason for not using Floyd is that the graph is too sparse.

(ii) For each train, enumerate all the pairs of consecutive stops. For each two consecutive stops (x, y) and each railroad (u, v), if $dist[x, u] + len(u, v) + dist[v, y] = dist[x, y]$, or $dist[x, v] + len(v, u) + dist[u, y] = dist[x, y]$, then the railroad (u, v) will affect this train.

(iii) Now, for each edge (u, v), we get a set of trains that will be affected.

The problem then becomes: pick a set of k edges, calculate the total importance of trains affected, and maximize this total importance.

6 Algorithms

6.1 Greedy

This problem has a very similar format compared to the **Influence maximization** problem. Inspired by that, we describe the following algorithm:

(i) Initially, the set of edges chosen is empty.

(ii) Each time, pick the edge that will maximize the **marginal gain**. Intuitively, the increase of total influence should be as large as possible after each edge added.

(iii) Repeat this process k times.

Interestingly, the simple greedy algorithm performs surprisingly well on the realistic data - Of course, we noticed this only after we designed and implemented several other algorithms. Figure 2 demonstrates the result. To understand the result, we see that the total influence on the entire graph (when we remove all edges) is 9546151.5, so it's possible to disable over 75% of the railway network capacity by destroying only 20 edges in the network.

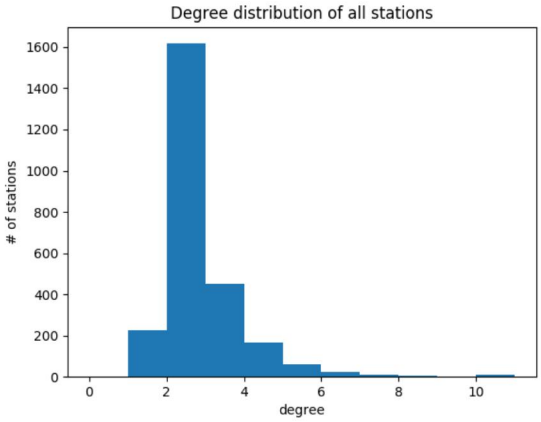

Figure 1: Degree Distribution

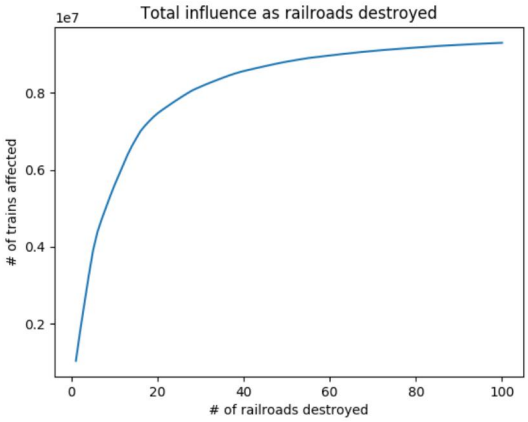

Figure 2: Result Achieve by Greedy

6.2 Dividing the Graph

In the above algorithm, we ignore the underlying structure of the trains - they are all paths on some graph we know. The trains affected by each edge is not a random set! Therefore, we decide to leverage this information and design some new algorithms.

Usually, if greedy algorithm could not get an ideal result, it's stuck on some local maximum and "lose the big picture". The following algorithm is inspired by the K-Means algorithm: since the railroad graph is almost planar, we would like to find K centers of the graph, dividing the graph into K clusters. We only destroy one edge in each cluster, so that intuitively there's some kind of "global coordination". Here is the algorithm:

(i) Define the weight of each edge to be a constant c minus the number of trains affected by it. Calculate the pairwise shortest distance based on this metric.

(ii) Random sample K nodes as the initial center.

(iii) For each node, assign it to the center that it's closest to.

(iv) For each center, re-calculate the optimal center based on the nodes assigned to it. Go back to (iii), iterate this process until convergence.

(v) Now we get K clusters. Greedily add edges as we did before ("maximize marginal gain"), but each cluster could only contribute one edge. The order of clusters are random.

In the above algorithm, intuitively, we want to identify the clusters with largest train density. Unfortunately, we didn't see this algorithm outperforms the greedy one, as shown in Figure 3.

Nevertheless, it does successfully identify the clusters and their centers. Without prior knowledge, the centers calculated by this algorithm are usually the major "railway junction" cities in China. Some of the stations that appear the most frequently are: Zhengzhou,

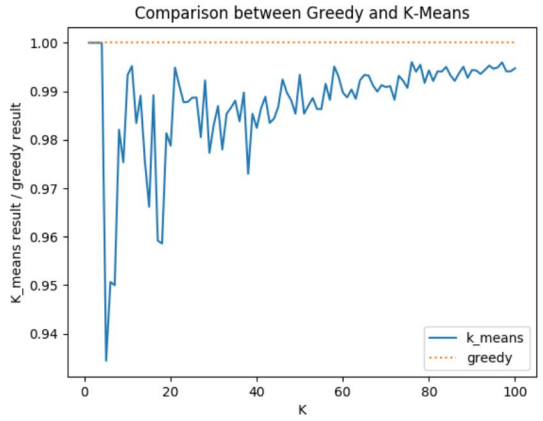

Figure 3: K-means v.s. Greedy

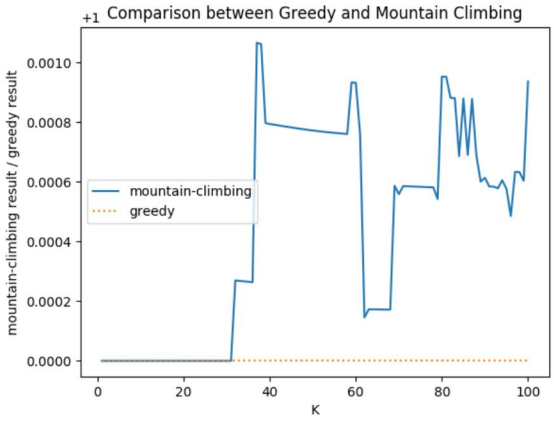

Figure 4: Greedy with Mountain Climbing

Hankou/Wuchang[1], Beijing, Shanghai, Nanjing, Guangzhou, Chengdu. The result really depends on K. If K is 3 or 4, usually the result is like one major city from each part of China - North, East, South, Central/Central West. If K is around twenty, the top largest cities are usually detected. If K is larger, some "local" junctions will also be included. Sometimes there are multiple stations from the same city, and only one of them will appear. Notice that, among the 2576 stations, only a small amount of them are stations of major junctions, but this algorithm could detect them reasonably.

6.3 Railroad-based Mountain Climbing

Given that our attempt to "coordinate globally" was not very successful, we decided to go back to the original greedy algorithm, but incorporate the graph structure inside the algorithm.

One way to improve the greedy algorithm is through local search (we call it Mountain Climbing; it's fundamentally the same as MCMC). We start with the greedy algorithm, but each time, we randomly "twist" the current result by a little bit; if we get a better result, then that will be the new "current result", while if we get a worse result, we accept it with a certain probability depending on how worse it is.

Here is the full algorithm:

(i) Initialize with the result obtained from greedy.

(ii) Each time, we "twist" in the following way: move an edge to one of its neighboring edges. If we detect that we're moving on a chain, keep moving in that direction until we reach the end. Also, remember the last time when this edge was moved; don't move back.

[1]Hankou and Wuchang are two stations located very close to each other, both belonging to the city of Wuhan

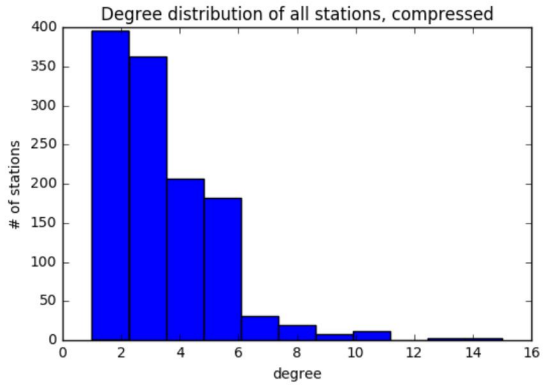

Figure 5: Degree Distribution of Compressed Network

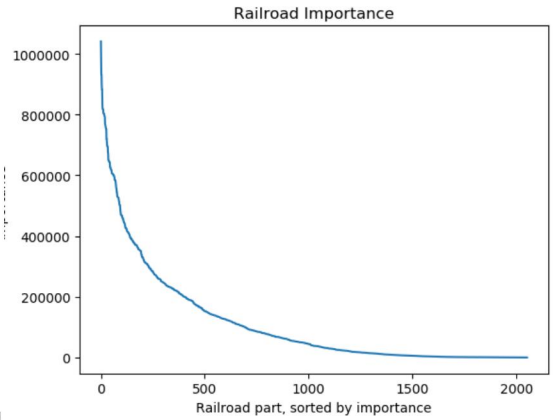

Figure 6: Influence of Each Edge

(iii) If the new result is better, accept it. Otherwise, accept it with probability $0.5 * e^{\eta * \delta}$. η is some constant about the scale of the answer; in this problem, we pick $\eta = 0.01$ after experiments.

This algorithm could achieve better result on larger K's. The improvement is not much, though (mainly because the answer itself is too large). Figure 4 shows this result.

7 Analysis

We see that the greedy algorithm performs very well and other proposed algorithms have little to no success in making any improvement. It sounds quite strange in the first place, but it's quite expected once we look further into the structure of the network.

In order to more efficiently analyze the network, we first do the following "compression" on the network: if a station has degree 2, and the two railroads which it connects to have exactly the same set of trains running on them, then we can effectively contract the station and merge the two edges into one. This process reduces the network to having 1222 nodes and 2053 edges. The degree distribution is seen in Figure 5. The degree of most nodes are still low, so we would still expect structures that are similar to "links". An inspection of the influence of each edge (Figure 6) shows that not all edges are "born equal": in fact, influence seems to be distributed on a logarithm scale, and most edges have little influence compared to the top edges. Therefore This suggests that even though greedy algorithm would not guarantee a optimal solution, the improvement of switching certain latter choices would make little improvement and the result of a greedy algorithm will be quite similar to the optimal solution. This is also proven in Figure 7, where we show all of the possible combination of our choices in the first two steps with the color denoting the possible increment on influence where we see that the majority of choices result in an insignificant increment of influence.

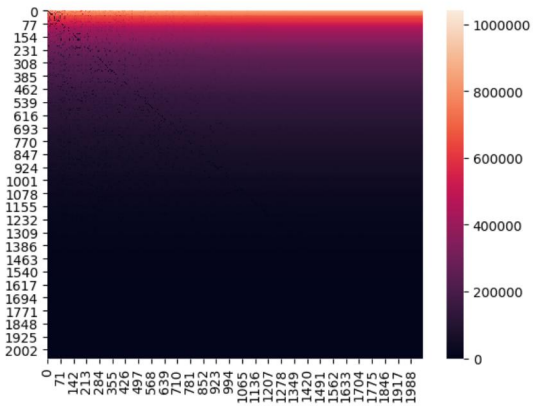

Figure 7: Possible Increment of Influence in the Second Step of Greedy Algorithm

8 Member Contributions and Link to Project Repository

All members of the team are involved in the entire process of the project, and made equal amounts of contribution to this project.

Link to project repository: `https://github.com/alicezhy/poisonous-tickets`.

We crawled the data ourself and generated the Chinese railway network. The data set is available under folders “raw data” and “generate railroad map”.

References

[1] Friedrich, M., Müller-Hannemann, M., Rückert, R., Schiewe, A., Schöbel, A. (2017). Robustness Tests for Public Transport Planning. In OASIcs-OpenAccess Series in Informatics (Vol. 59). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[2] Galluccio, L., Michel, O., Comon, P., Hero III, A. O. (2012). Graph based k-means clustering. Signal Processing, 92(9), 1970-1984.

[3] Mullor, E. (2015). Markov chain monte carlo methods applied to big graphs clustering (Master's thesis, Universitat Politècnica de Catalunya).