CMPLXSYS 530 Final Project

Alyssa Travitz

Due 30-Apr-2020

**Introduction**

I'm interested in modeling subway system layouts and how they affect different types of people, specifically people that [trip-chain](https://www.wctrs-society.com/wp-content/uploads/abstracts/lisbon/selected/02499.pdf). Trip chaining is predominantly exhibited by women with children, especially [single mothers](https://nhts.ornl.gov/1995/Doc/Chain2.pdf). Although trip-chaining is common, most public transportation systems are structured in a way that is believed to penalize trip-chaining. Travellers might have a time penalty from having to take circuitous routes to stop at their locations, or they have a monetary penalty by having to pay multiple fares (some cities do not have fare-transfers between stops or transportation modes). **I would like to create a model that captures how the topology of a transportation network affects the time and monetary cost of trip-chaining travelers.**

**Methods**

**Model Description**

Properties: The agents in this model represent passengers using the CTA train system. The agents interact only with the environment and not with each other. Each agent has the following properties:

* **transit time**: the time spent on travel, in time units
* **home**: the node representing the agent’s “home” station, randomly chosen.
* **itinerary**: the list of destination nodes the agent will visit, in the order they will visit them. Agents with more intermediate stops (more trip-chaining), will have more destination nodes in their itineraries. All itineraries begin and end with the **home** node. All desintation nodes are randomly chosen
* **itinerary index:** an index for tracking the agent’s progression in their itinerary
* **current node:** the node the agent is currently occupying
* **destination:** the node the agent is currently en route to (as determined by the itinerary)
* **completed:** a simple tracker to record if the passenger has visited all the nodes on their itinerary

Actions: At each time step, all agents follow the below protocol, iterated over in a random order each time:

1. If the agent has not completed its trip and is at a destination, compute the next destination
2. Compute the shortest path to the agent’s destination node
3. Move to the next node in the shortest path
4. Accumulate time penalties for transferring nodes or moving to a node that is over capacity (both 3x time penalty)
5. Add +1 to the agent’s transit time

Rules: I should note that I initially set up time penalization for nodes being at/over capacity by not advancing an agent until its next node was under capacity. However, this often led to a permanent gridlock in the central loop of the network, so I implemented the time penalty without actually delaying the agents. I would appreciate any feedback on how to explicitly model crowding without causing a gridlock. I realize that this is a result of me treating subway cars as implicit in this model.

Time: Time is implemented discretely, in that each agent can move one node at each time step. Time penalties are counted as multiples of a time step, typically adding 3 time step increments to a passengers **transit time**.

Environment:

I used the Chicago Transit Authority (CTA) train service as a reference for my model. The CTA provides route data in the General Transit Feed Specification (GTFS) format, which provides stations, stops, transport routes, and other necessary data. I converted the CTA’s GTFS data to a 2-D network (GTFS\_to\_netx.py) that keeps the spatial coordinates of the stations, as shown in **Fig. 1.**

A close up of a map

Description automatically generated

Figure 1 A networkx representation of the CTA train service

The nodes retain their information about which color line they are assigned to, but the connectivity of the graph is represented as single connections between stations, rather than accounting for multiple lines between stations. My goal was to represent the topology of the system in the most simplified format possible.

All of the 140 nodes on the graph are initialized with the same constant **capacity**, or the number of agents that can occupy that node at a given time step. At each time step a population balance is performed to update the nodes with the correct number of agents.

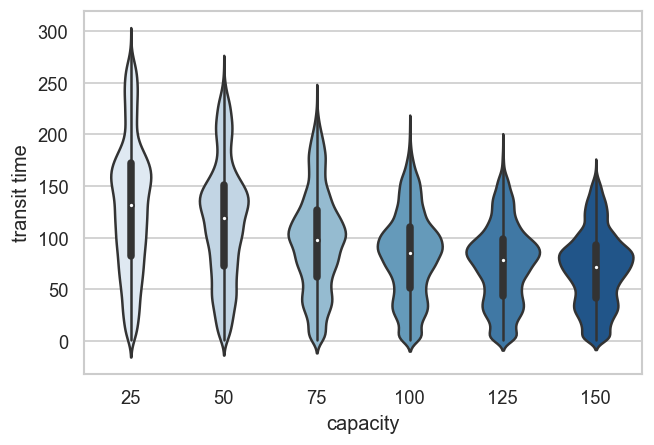
**Model Analysis**

All simulations were run with 10,000 passengers and a node capacity of 50 (except where stated otherwise). Simulations were run until all agents had completed their itineraries (with a maximum run time for 1,000 time steps in the case of grid locks). Any quantitative result is averaged over three simulations to account for random variance.

**Results & Discussion**

**Station Capacity and Crowding**

I first wanted to validate that I would see crowding effects on my network if I sufficiently reduced the capacity of the nodes. It would seem that for a system with 10,000 agents and 140 nodes that there would be a noticeable increase in transit times at a node capacity of 10,000/140, or 71 passengers per node. The limit of this would be when the node capacity is equal to the number of agents, which would represent a transit system where there never wait times. **Figure 2A** shows a violin plot of transit time distributions for each node capacity. From the distributions it seems that the peak of the distributions skews slightly higher than the mean, and that it broadens as the node capacity decreases. To more clearly show the trend of transit time as a function of node capacity, **Figure 2B** shows the mean values with error bars representing the 68% confidence interval (smaller than the marker sizes). As expected, we see a decrease in transit time as the node capacity increases.

 A screenshot of a cell phone

Description automatically generated

**Trip Chaining Effects**

The main motivation for this project is to see how trip-chaining affects travel times and if changes to the topology of the subway network can favor trip-chaining passengers.

**Adding an “Outskirts” Subway Line**

**Future Work**