# 1. What does bus trajectory do?

This project tries to find out the reachability of bus transit in Los Angeles. Say, it finds out how far a person

in a place can travel within 40 minutes by taking bus and walking in LA.

To achieve this goal, it first generates two networks. One is a transit network, which is generated by Urbanaccess. The other one is the pedestrian network, which is generated by Osmnet. Then Urbanaccess integrates these two network into one network. All of the following computations are based on this integrated network.

Bedsides this network, we also need information such as how many jobs each census block have and the shapefiles of those blocks. These two kind of information could be downloaded from internet.

With these data, we can calculate how many jobs a person in a place can reach within a given time by taking bus and walking, or how far we can reach from a block within a given time. These data are generated in json files, which are shown through web pages.

# 2. Dependencies

## 2.1 simpledbf

### 2.1.1 Introduction

simpledbf is a Python library for converting basic DBF files to CSV files, Pandas DataFrames, SQL tables, or HDF5 tables.

### 2.1.2 Install

pip install simpledbf

### 2.1.3 Usage:

```
In : from simpledbf import Dbf5


In : dbf = Dbf5('fake_file_name.dbf', codec='utf-8')
```

```
In : df = dbf.to_dataframe()
# df is a DataFrame with all records
```

## 2.2 shapefile

### 2.2.1 Inrotduction

The Python Shapefile Library (pyshp) provides read and write support for the Esri Shapefile format. The Shapefile format is a popular Geographic Information System vector data format created by Esri. The Esri document describes the shp and shx file formats. However a third file format called dbf is also required. This format is documented on the web as the “XBase File Format Description” and is a simple file-based database format created in the 1960’s.

### 2.2.2 Install

pip install shapefile

### 2.2.3 Usage

```
>>> import shapefile
>>> sf = shapefile.Reader("shapefiles/blockgroups")
```

## 2.3 OSMnet 0.1.4

### 2.3.1 Introduction

Tools for the extraction of OpenStreetMap (OSM) street network data. Intended to be used in tandem with Pandana and UrbanAccess libraries to extract street network nodes and edges.

OSMnet offers tools to download street network data from OpenStreetMap and extract a graph network comprised of nodes and edges to be used in Pandana street network accessibility calculations.

Documentations https://udst.github.io/osmnet/index.html.

### 2.3.2 Install

pip install osmnet

2.3.3 Usage

osmnet.load.network_from_bbox(lat_min=None,lng_min=None,lat_max=None,lng_max=None,bbox=None,network_type='walk',two_way=True,timeout=180,memory=None,max_query_area_size=2500000000)

Make a graph network from a bounding lat/lon box composed of nodes and edges for use in Pandana street network accessibility calculations. You may either enter a lat/long box via the four lat_min, lng_min, lat_max, lng_max parameters or the bbox parameter as a tuple.

Parameters:

**lat_min** : float

southern latitude of bounding box, if this parameter is used the bbox parameter should be None.

**lng_min** : float

eastern latitude of bounding box, if this parameter is used the bbox parameter should be None.

**lat_max** : float

northern longitude of bounding box, if this parameter is used the bbox parameter should be None.

**lng_max** : float

western longitude of bounding box, if this parameter is used the bbox parameter should be None.

**bbox** : tuple

Bounding box formatted as a 4 element tuple: (lng_max, lat_min, lng_min, lat_max) example: (-122.304611,37.798933,-122.263412,37.822802) a bbox can be extracted for an area using: the CSV format bbox fromhttp://boundingbox.klokantech.com/. If this parameter is used the lat_min, lng_min, lat_max, lng_max parameters in this function should be None.

**network_type** : {'walk', 'drive'}, optional

Specify the network type where value of ‘walk’ includes roadways where pedestrians are allowed and pedestrian pathways and ‘drive’ includes driveable roadways. Default is walk.

**two_way** : bool, optional

Whether the routes are two-way. If True, node pairs will only occur once.

**timeout** : int, optional

the timeout interval for requests and to pass to Overpass API

**memory** : int, optional

server memory allocation size for the query, in bytes. If none, server will use its default allocation size

max_query_area_size : float, optional

max area for any part of the geometry, in the units the geometry is in: any polygon bigger will get divided up for multiple queries to Overpass API (default is 50,000 * 50,000 units (ie, 50km x 50km in area, if units are meters))

**remove_lcn** : bool, optional

remove low connectivity nodes from the resulting pandana network. This ensures the resulting network does not have nodes that are unconnected from the rest of the larger network

Returns: nodesfinal, edgesfinal : pandas.DataFrame

## 2.4 Urbanaccess 0.1a

### 2.4.1 Introduction

A tool for computing GTFS transit and OSM pedestrian networks for accessibility analysis.

UrbanAccess is tool for creating multi-modal graph networks for use in multi-scale (e.g. address level to the metropolitan level) transit accessibility analyses with the network

analysis tool Pandana. UrbanAccess uses open data from General Transit Feed Specification (GTFS) data to represent disparate operational schedule transit networks and pedestrian OpenStreetMap (OSM) data to represent the pedestrian network. UrbanAccess provides a generalized, computationally efficient, and unified accessibility calculation framework by linking tools for: 1) network data acquisition, validation, and processing; 2) computing an integrated pedestrian and transit weighted network graph; and 3) network analysis using Pandana.

UrbanAccess offers the following tools:

- GTFS and OSM network data acquisition via APIs
- Network data validation and regional network aggregation
- Compute network impedance:
    - by transit schedule day of the week and time of day
    - by transit mode
    - by including average passenger headways to approximate passenger transit stop wait time
- Integrate pedestrian and transit networks to approximate pedestrian scale accessibility
- Resulting networks are designed to be used to compute accessibility metrics using the open source network analysis tool Pandana
    - Compute cumulative accessibility metrics
    - Nearest feature analysis using POIs

Documents, https://udst.github.io/urbanaccess/index.html.

2.4.2 Installation

1) Git clone the UrbanAccess repo
2) in the cloned directory run: python setup.py develop

2.4.3 Usage

1) Create A Transit Network

`urbanaccess.gtfs.network.create_transit_net`(gtfsfeeds_dfs, day, timerange, calendar_dates_lookup=None, overwrite_existing_stop_times_int=False, use_existing_stop_times_int=False, save_processed_gtfs=False, save_dir='data', save_filename=None)

Create a travel time weight network graph in units of minutes from GTFS data

Parameters:

gtfsfeeds_dfs : object

gtfsfeeds_dfs object with DataFrames of stops, routes, trips, stop_times, calendar, calendar_dates (optional) and stop_times_int (optional)

**day** : {'friday', 'monday', 'saturday', 'sunday', 'thursday', 'tuesday',

'wednesday'}

day of the week to extract transit schedule from that corresponds to the day in the GTFS calendar

timerange : list

time range to extract transit schedule from in a list with time 1 and time 2. it is suggested the time range specified is large enough to allow for travel from one end of the transit network to the other but small enough to represent a relevant travel time period such as a 3 hour window for the AM Peak period. Must follow format of a 24 hour clock for example: 08:00:00 or 17:00:00

calendar_dates_lookup : dict, optional

dictionary of the lookup column (key) as a string and corresponding string (value) as string or list of strings to use to subset trips using the calendar_dates DataFrame. Search will be exact. If none, then the calendar_dates DataFrame will not be used to select trips that are not in the calendar DataFrame. Note search

will select all records that meet each key value pair criteria. Example: {'schedule_type' : 'WD'} or {'schedule_type' : ['WD', 'SU']}

overwrite_existing_stop_times_int : bool, optional

if true, and if there is an existing stop_times_int DataFrame stored in the gtfsfeeds_dfs object it will be overwritten

use_existing_stop_times_int : bool, optional

if true, and if there is an existing stop_times_int DataFrame for the same time period stored in the gtfsfeeds_dfs object it will be used instead of re-calculated

save_processed_gtfs : bool, optional

if true, all processed gtfs DataFrames will be stored to disk in a hdf5 file

**save_dir** : str, optional

directory to save the hdf5 file

**save_filename** : str, optional

name to save the hdf5 file as

Returns:

ua_network : object

ua_network.transit_edges : pandas.DataFrame

ua_network.transit_nodes : pandas.DataFrame

2) Create a street network

`urbanaccess.osm.network.create_osm_net`(osm_edges, osm_nodes, travel_speed_mph=3, network_type='walk')

Parameters:

**osm_edges** : pandas.DataFrame

osm edge dataframe

**osm_nodes** : pandas.DataFrame

osm node dataframe

**travel_speed_mph** : int, optional

travel speed to use to calculate travel time across a distance on a edge. units are in miles per hour (MPH) for pedestrian travel this is assumed to be 3 MPH

**network_type** : str, optional

default is 'walk' for the osm pedestrian network. this string is used to label the osm network once it is integrated with the transit network

Returns:

ua_network : object

urbanaccess_network object with osm_edges and osm_nodes dataframes

ua_network.osm_edges : pandas.DataFrame

ua_network.osm_nodes : pandas.DataFrame

3) Network Integration

`urbanaccess.network.integrate_network`(urbanaccess_network, headways=False, urbanaccess_gtfsfeeds_df=None, headway_statistic='mean')

Parameters:

urbanaccess_network : object

ua_network object with transit_edges, transit_nodes, osm_edges, osm_nodes

**headways** : bool, optional

if true, route stop level headways calculated in a previous step will be applied to the osm to transit connector edge travel time weights as an approximate measure of average passenger transit stop waiting time.

urbanaccess_gtfsfeeds_df : object, optional

required if headways is true; the gtfsfeeds_dfs object that holds the corresponding headways and stops DataFrames

**headway_statistic** : {'mean', 'std', 'min', 'max'}, optional

required if headways is true; route stop headway statistic to apply to the osm to transit connector edges: mean, std, min, max. Default is mean.

Returns:

urbanaccess_network : object

urbanaccess_network.transit_edges : pandas.DataFrame

urbanaccess_network.transit_nodes : pandas.DataFrame

urbanaccess_network.osm_edges : pandas.DataFrame

urbanaccess_network.osm_nodes : pandas.DataFrame

urbanaccess_network.net_connector_edges : pandas.DataFrame

urbanaccess_network.net_edges : pandas.DataFrame

urbanaccess_network.net_nodes : pandas.DataFrame

## 2.5 Pandana 0.3.0

### 2.5.1 Introduction

Pandana is a neologism representing Pandas Network Analysis.

Pandana performs hundreds of thousands of network queries in under a second (for walking-scale distances) using a Pandas-like API. The computations are parallelized for use on multi-core computers using an underlying C library.

Beyond simple access to destination queries, this library also implements more general aggregations along the street network (or any network). For a given region, this produces hundreds of thousands of overlapping buffer queries (still performed in less than a second) that can be used to characterize the local neighborhood around each street intersection. The result can then be mapped, or assigned to parcel and building records, or used in statistical models as we commonly do with UrbanSim. This is in stark contrast

to the arbitrary non-overlapping geographies ubiquitous in GIS. Although there are advantages to the GIS approach, we think network queries are a more accurate representation of how people interact with their environment.

2.5.2 Install

Clone the pandana repo (Version 0.3.0)

Run `python setup.py develop`

2.5.3 Usage

1) Create a transportation network

class`pandana.network.Network`(node_x, node_y, edge_from, edge_to, edge_weights, twoway=True)

Parameters:

**node_x** : Pandas Series, float

Defines the x attribute for nodes in the network (e.g. longitude)

**node_y** : Pandas Series, float

Defines the y attribute for nodes in the network (e.g. latitude) This param and the one above should have the *same* index which should be the node_ids that are referred to in the edges below.

**edge_from** : Pandas Series, int

Defines the node id that begins an edge - should refer to the index of the two series objects above

**edge_to** : Pandas Series, int

Defines the node id that ends an edge - should refer to the index of the two series objects above

**edge_weights** : Pandas DataFrame, all floats

Specifies one or more *impedances* on the network which define the distances between nodes. Multiple impedances can be used to capture travel times at different times of day, for instance

**twoway** : boolean, optional

Whether the edges in this network are two way edges or one way ( where the one direction is directed from the from node to the to node). If twoway = True, it is assumed that the from and to id in the edge table occurs once and that travel can occur in both directions on the single edge record. Pandana will internally flip and append the from and to ids to the original edges to create a two direction network. If twoway = False, it is assumed that travel can only occur in the explicit direction indicated by the from and to id in the edge table.

2) Aggregate information

`aggregate`(distance, type='sum', decay='linear', imp_name=None, name='tmp')

Parameters:

**distance** : float

The maximum distance to aggregate data within. 'distance' can represent any impedance unit that you have set as your edge weight. This will usually be a distance unit in meters however if you have customized the impedance this could be in other units such as utility or time etc.

**type** : string

The type of aggregation, can be one of "ave", "sum", "std", "count", and now "min", "25pct", "median", "75pct", and "max" will compute the associated quantiles. (Quantiles are computed by sorting so might be slower than the others.)

**decay** : string

The type of decay to apply, which makes things that are further away count less in the aggregation - must be one of "linear", "exponential" or "flat" (which means no decay). Linear is the fastest

computation to perform. When performing an “ave”, the decay is typically “flat”

**imp_name** : string, optional

The impedance name to use for the aggregation on this network. Must be one of the impedance names passed in the constructor of this object. If not specified, there must be only one impedance passed in the constructor, which will be used.

**name** : string, optional

The variable to aggregate. This variable will have been created and named by a call to set. If not specified, the default variable name will be used so that the most recent call to set without giving a name will be the variable used.

Returns:

**agg** : Pandas Series

Returns a Pandas Series for every origin node in the network, with the index which is the same as the node_ids passed to the init method and the values are the aggregations for each source node in the network.

## 2.6 Others

### 2.6.1 Other dependencies

numpy 1.3.1 and pandas 0.20.3.

### 2.6.2 Installation

pip install

# 3 Input

## 3.1 Tiger Shapefiles

### 3.1.1 What is Shapefile?

A shapefile is a geospatial data format for use in geographic information system (GIS) software. Shapefiles spatially describe vector data such as points, lines, and polygons, representing, for instance, landmarks, roads, and lakes. The Environmental Systems Research Institute (Esri) created the format for use in their software, but the shapefile format works in additional Geographic Information System (GIS) software as well.

### 3.1.2 Structure

The 2010 Census TIGER/Line Shapefiles and associated relationship files are offered in a compressed format. One zipped file is available for each layer, with a file extension of .zip. Each zipped shapefile consists of the following five files:

.shp – the feature geometry

.shx – the index of the feature geometry

.dbf – the tabular attribute information

.prj – the coordinate system information

.shp.xml – the metadata

Each zipped relationship file consists of the following two files:

.dbf – the tabular attribute information

.dbf.xml – the metadata

### 3.1.3 Geographic relationships

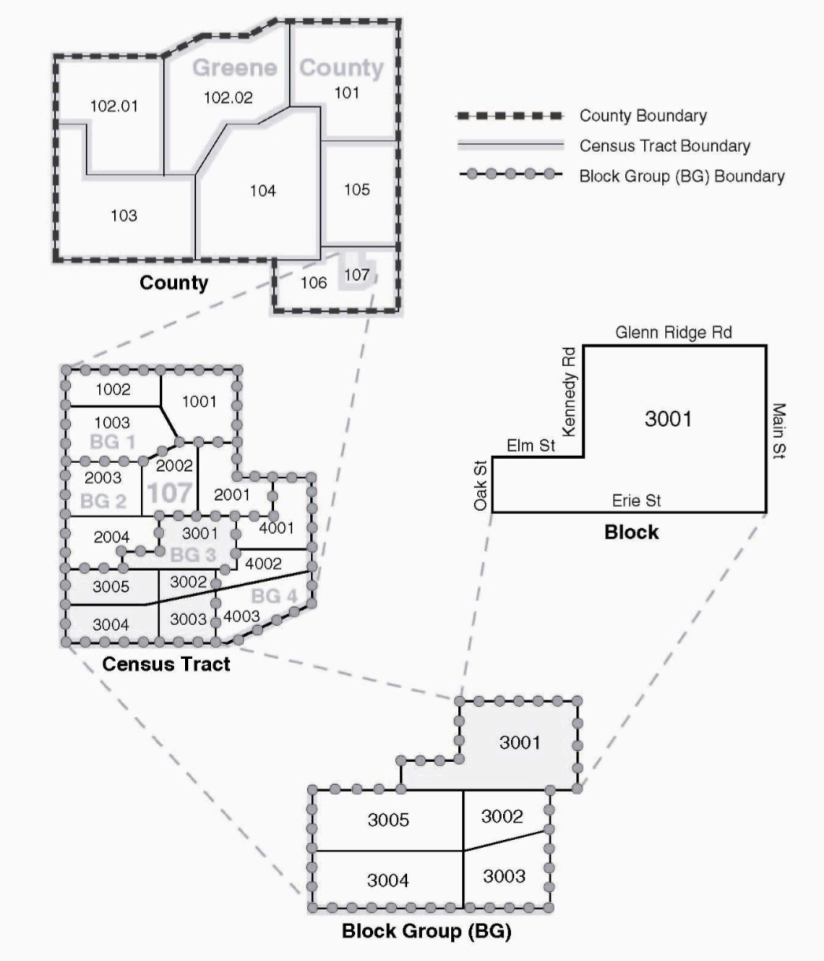



Figure 1. Geographic relationships

### 3.1.4 Block Shapefile

Census Blocks are statistical areas bounded on all sides by visible features, such as streets, roads, streams, and railroad tracks, and by non-visible boundaries such as city, town, township, and county limits, and short line-of-sight extensions of streets and roads. Generally, census blocks are small in area; for example, a block in a city. Census blocks in suburban and rural areas may be large, irregular, and bounded by a variety of features, such as roads, streams, and/or transmission line rights-of-way. In remote areas census blocks may encompass hundreds of square miles. Census blocks cover all territory in the United States, Puerto Rico, and the Island areas.

File Name: tl_2010_<state FIPS>_tabblock10.shp

| Field | Length | Type | Description |
|---|---|---|---|
| STATEFP10 | 2 | String | 2010 Census state FIPS code |
| COUNTYFP10 | 3 | String | 2010 Census county FIPS code |
| TRACTCE10 | 6 | String | 2010 Census census tract code |
| BLOCKCE10 | 4 | String | 2010 Census tabulation block number |
| GEOID10 | 15 | String | Block identifier; a concatenation of 2010 Census state FIPS code, county FIPS code, census tract code and tabulation block number. |
| NAME10 | 10 | String | 2010 Census tabulation block name; a concatenation of 'Block' and the current tabulation block number |
| MTFCC10 | 5 | String | MAF/TIGER feature class code (G5040) |
| UR10 | 1 | String | 2010 Census urban/rural indicator |
| UACE10 | 5 | String | 2010 Census urban area code |
| UATYP10 | 1 | String | 2010 Census urban area type |
| FUNCSTAT10 | 1 | String | 2010 Census functional status |
| ALAND10 | 14 | Number | 2010 Census land area |
| AWATER10 | 14 | Number | 2010 Census water area |
| INTPTLAT10 | 11 | String | 2010 Census latitude of the internal point |
| INTPTLON10 | 12 | String | 2010 Census longitude of the internal point |

3.1.5 Block Groups Shapefiles

Block groups are clusters of blocks within the same census tract that have the same first digit of their 4-digit census block number. For example, blocks 3001, 3002, 3003, ...., 3999 in census tract 1210.02 belong to Block Group 3. As with block groups delineated for Census 2000, block groups delineated for the 2010 Census generally contain between 600 and 3,000 people.

File Name: tl_2010_<state FIPS>_bg10.shp

| Field | Length | Type | Description |
|---|---|---|---|
| STATEFP10 | 2 | String | 2010 Census state FIPS code |
| COUNTYFP10 | 3 | String | 2010 Census county FIPS code |
| TRACTCE10 | 6 | String | 2010 Census census tract code |
| BLKGRPCE10 | 1 | String | 2010 Census block group number |
| GEOID10 | 12 | String | 2010 Census block group identifier; a concatenation of the 2010 Census state FIPS code, county FIPS code, census tract code, and block group number |
| NAMELSAD10 | 13 | String | 2010 Census translated legal/statistical area description and the block group number |
| MTFCC10 | 5 | String | MAF/TIGER feature class code (G5030) |
| FUNCSTAT10 | 1 | String | 2010 Census functional status |
| ALAND10 | 14 | Number | 2010 Census land area |
| AWATER10 | 14 | Number | 2010 Census water area |
| INTPTLAT10 | 11 | String | 2010 Census latitude of the internal point |
| INTPTLON10 | 12 | String | 2010 Census longitude of the internal point |

3.1.6 Where to download shapefiles?

https://www.census.gov/geo/maps-data/data/tiger-line.html

## 3.2 LEHD Origin-Destination Employment Statistics

### 3.2.1 Introduction

The LEHD Origin-Destination Employment Statistics (LODES) datasets are released both as part of the OnTheMap application and in raw form as a set of comma separated variable (CSV) text files. This document describes the structure of those raw files and provides basic information for users who want to perform analytical work on the data outside of the OnTheMap application.

### 3.2.2 File organization

At the root distribution level, each state (or state-equivalent) has a directory named by its lowercase, 2-letter postal code. Within each state directory are directories of data as well as metadata and geography files for the state. These are described below.

The CSV data files are released at the state level and they are organized into three groups within each state. The three groups of files are named as follows, according to their contents:

OD – Origin-Destination data, jobs totals are associated with both a home Census Block and a work Census Block

RAC – Residence Area Characteristic data, jobs are totaled by home Census Block

WAC – Workplace Area Characteristic data, jobs are totaled by work Census Block A typical directory tree looks as follows:

```
\
|-- ak
|   |-- od
|   |   `-- (individual origin-destination data files)
|   |-- rac
|   |   `-- (individual residence area characteristics data files)
|   `-- wac
|   |   `-- (individual workplace area characteristics data files)
|   |-- ak_xwalk.csv.gz
|   |-- otm_ak.md5sum
|   `-- version.txt
|-- az
|
(more states)
```

For this project, right now we only need OD files.

### 3.2.3 OD datasets

The structure of the OD files is as follows:

| Origin-Destination (OD) File Structure | | | |
|---|---|---|---|
| **Pos** | **Variable** | **Type** | **Explanation** |
| 1 | w_geocode | Char15 | Workplace Census Block Code |
| 2 | h_geocode | Char15 | Residence Census Block Code |
| 3 | S000 | Num | Total number of jobs |
| 4 | SA01 | Num | Number of jobs of workers age 29 or younger[12] |
| 5 | SA02 | Num | Number of jobs for workers age 30 to 54[12] |
| 6 | SA03 | Num | Number of jobs for workers age 55 or older[12] |
| 7 | SE01 | Num | Number of jobs with earnings $1250/month or less |
| 8 | SE02 | Num | Number of jobs with earnings $1251/month to $3333/month |
| 9 | SE03 | Num | Number of jobs with earnings greater than $3333/month |
| 10 | SI01 | Num | Number of jobs in Goods Producing industry sectors |
| 11 | SI02 | Num | Number of jobs in Trade, Transportation, and Utilities industry sectors |
| 12 | SI03 | Num | Number of jobs in All Other Services industry sectors |
| 13 | createdate | Char | Date on which data was created, formatted as YYYYMMDD |

### 3.2.4 Where to download OD files?

https://lehd.ces.census.gov/data/

## 3.3 Other parameters

### 3.3.1 Time range

Time range to extract transit schedule from in a list with time 1 and time 2. it is suggested the time range specified is large enough to allow for travel from one end of the transit network to the other but small enough to represent a relevant travel time period such as a 3 hour window for the AM Peak period. Must follow format of a 24 hour clock for example: 08:00:00 or 17:00:00.

### 3.3.2 Bounding box

Bounding box formatted as a 4 element tuple: (lng_max, lat_min, lng_min, lat_max) example: (-122.304611,37.798933,-122.263412,37.822802) a bbox can be extracted for an area using: the CSV format bbox from http://boundingbox.klokantech.com/.

### 3.3.3 How to set parameters?

All the parameters are set at the beginning of file la_urbanaccess.py

a) gtfs folder path(absolute path)

GTFS_PATH = '/Volumes/WD-zqh/bus_trajectory/input/gtfs_bus/'

set this variable to to your absolute path

b) dbf file of census block

DBF_PATH="./input/tl_2010_06037_tabblock10/tl_2010_06037_tabblock10.dbf"

c) number of jobs

JOB_PATH = "./input/ca_od_main_JT00_2010.csv"

d) shapefile of census block

SHAPEFILE_PATH = './input/tl_2010_06037_tabblock10/tl_2010_06037_tabblock10'

e) shapefile of block group

GROUP_SHAPE_PATH = "./input/tl_2010_06037_bg10/tl_2010_06037_bg10"

f) Time range

time_range = ["06:00:00","09:00:00"]

g) Bounding box

use one of them, or any bounding box you like.

i) almost all areas in LA

bounding_box = (-118.85, 33.67, -117.67, 34.32) # big

ii) a very small region, for test

bounding_box = (-118.30, 34.0, -118.25, 34.06) # small

# 4 Output

## 4.1 net.h5

The integrated network is saved to a Pandanas HDF5 file. Only the nodes and edges of the actual network are saved, points-of-interest and data attached to nodes are not saved.

### 4.2 block_closest_node.csv

This files saves the closest node in the network for each census block. Each row records a census block and its closest node in the network.

| Variable | Type | Explanation |
|---|---|---|
| GEOID10 | string | The geo id of a census block |
| INTPTLAT10 | double | Latitude of the central point of this census block |
| INTPTLON10 | double | Lontitude of the central point of this census block |
| node_id | int | id of the closest node in network |
| net_lat | double | Latitude of the closest node of this census block |
| net_lon | double | Lontitude of the closest node of this census block |
| in_box | int | Whether this block is inside the bounding box |

### 4.3 jobs_all_20.csv

This file records the number of jobs each node in the network can reach within 20 minutes. It is the results of aggregation of Pandana.

| Variable | Type | Explanation |
|---|---|---|
| id_int | int | id of a node in the network |
| node_id | int | id of a node in the network |
| 0 | int | Number of jobs this node can reach within 20 minutes |
| net_lat | double | Latitude of this node in the network |
| net_lon | double | Lontitude of this node in the network |

### 4.4 jobs_all_40.csv

This file records the number of jobs each node in the network can reach within 40 minutes. It is the results of aggregation of Pandana.

| Variable | Type | Explanation |
|---|---|---|

| id_int | int | id of a node in the network |
|---|---|---|
| node_id | int | id of a node in the network |
| 0 | int | Number of jobs this node can reach within 40 minutes |
| net_lat | double | Latitude of this node in the network |
| net_lon | double | Lontitude of this node in the network |

### 4.5 jobs_all_60.csv

This file records the number of jobs each node in the network can reach within 60 minutes. It is the results of aggregation of Pandana.

| Variable | Type | Explanation |
|---|---|---|
| id_int | int | id of a node in the network |
| node_id | int | id of a node in the network |
| 0 | int | Number of jobs this node can reach within 60 minutes |
| net_lat | double | Latitude of this node in the network |
| net_lon | double | Lontitude of this node in the network |

### 4.6 blocks.json

A json file that records some information for each census block. It includes the boundary nodes of this census block and its id. It also includes the number of jobs this census block can reach within 20, 40 and 60 minutes respectively. This file is used by the demo.


```
{
  "type": "Feature",
"properties": {
  "min20": 12075,
    "min40": 105884,
    "min60": 783110,
    "id": "060373006004026"
  },
20
```

```
  "geometry": {
      "type": "Polygon",
      "coordinates": [[[-104.99404, 39.75621], …]
  }
}
```


### 4.7 groups.json

It records the information for each block group. It includes the boundary nodes of this block group and its id. This file is used by the demo. It shows the information of groups.


```
{
  "type": "Feature",
"properties": {
      "id": "060372932023"
  },
  "geometry": {
      "type": "Polygon",
      "coordinates": [[[-104.99404, 39.75621], …]
  }
}
```


### 4.8 nearest20

A folder that contains many files, where each file records all the blocks a census block can reach within 20 minutes. This folder is also needed by the demo.


```
 { "nearest": [
    "060371011221009",
    "060371011101006",
    "060371011102001",
    "060371011101005",
```

"060371011102000"

]

}

# 5 Workflow

## 5.1 Compute network

In this step, we compute an integrated pedestrian and transit weighted network graph by using Urbanaccess.

### 5.1.1 Load GTFS data

Load raw GTFS data (from multiple feeds) into a UrbanAccess transit data object and run data through the validation and formatting sequence.

GTFS feeds are assumed to either be a single feed designated by the feed folder or multiple feeds designated as the root folder that holds all individual feed folders.

Read all GTFS feed components as a dataframe in a gtfsfeeds_dfs object and merge all individual GTFS feeds into a regional metropolitan data table. Optionally, data can also be validated before its use.

The input of this step is GTFS data.

The output is processed dataframes of corresponding GTFS feed text files.

The function of loading GTFS data is:

`urbanaccess.gtfs.load.gtfsfeed_to_df`(gtfsfeed_path=None, validation=False, verbose=True, bbox=None, remove_stops_outsidebbox=None, append_definitions=False)

### 5.1.2 Download and load OpenStreetMap data

Download OSM street network nodes and edges. Make a graph network (nodes and edges) from a bounding lat/lon box that is compatible with the network analysis tool Pandana.

The input of this step is a bounding box.

The output is nodesfinal, edgesfinal : pandas.DataFrame.

The function to download and load OSM data is

`urbanaccess.osm.load.ua_network_from_bbox`(lat_min=None, lng_min=None, lat_max=None, lng_max=None, bbox=None, network_type='walk', timeout=180, memory=None, max_query_area_size=2500000000, remove_lcn=True)

5.1.3 Create a transit network

Create a transit network from the loaded GTFS feeds with travel time impedance.

The input of this step is GTFS feeds.

The output is network data object stores the individual transit network node and edge components.

`urbanaccess.gtfs.network.create_transit_net`(gtfsfeeds_dfs, day, timerange, calendar_dates_lookup=None, overwrite_existing_stop_times_int=False, use_existing_stop_times_int=False, save_processed_gtfs=False, save_dir='data', save_filename=None)

5.1.4 Create a street network

Create a street network from the loaded OSM data with travel time impedance.

The input of this step is OSM nodes and edges dataframe.

The output is network data object stores the street network node and edge components.

`urbanaccess.osm.network.create_osm_net`(osm_edges, osm_nodes, travel_speed_mph=3, network_type='walk')

5.1.5 Network Integration

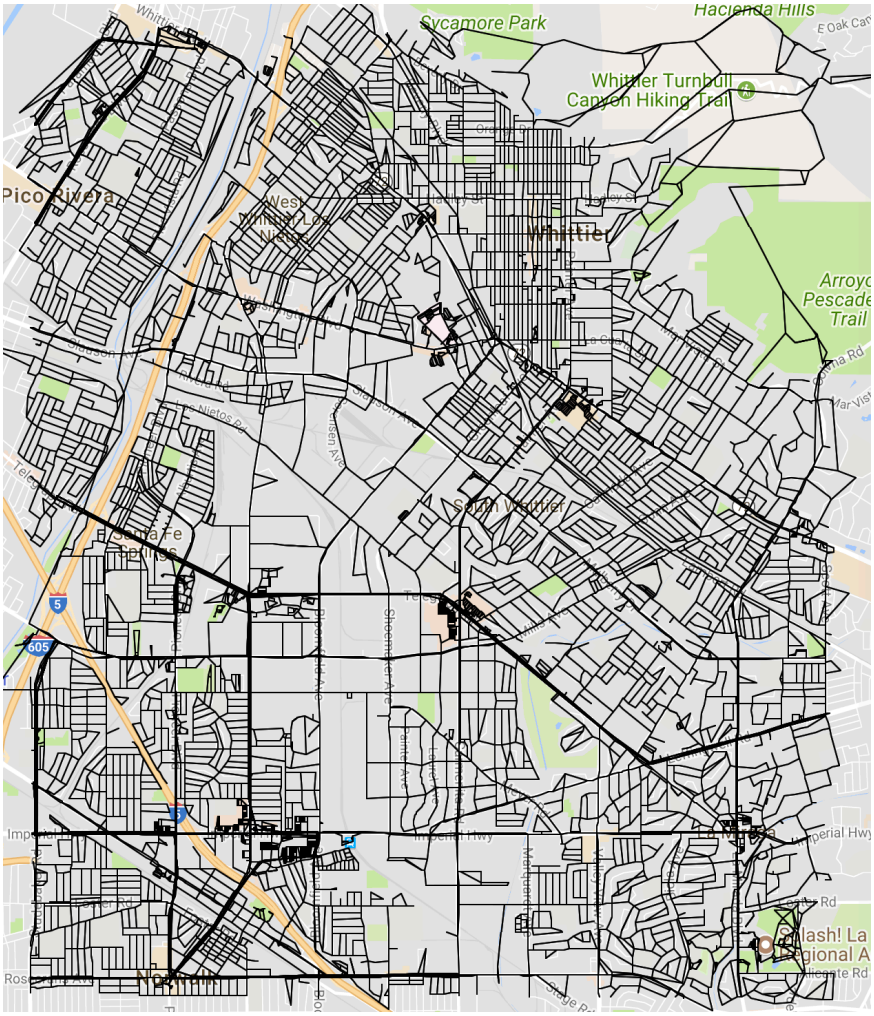
Create an integrated network comprised of transit and OSM nodes and edges by connecting the transit network with the osm network. travel time is in units of minutes.

The input of this step are transit network and street network.

The output is network data object stores the integrated network node and edge components.

`urbanaccess.network.integrate_network`(urbanaccess_network, headways=False, urbanaccess_gtfsfeeds_df=None, headway_statistic='mean')

A figure of the integrated network is




5.1.6 Save Network

Write a urbanaccess_network integrated nodes and edges to a node and edge table in a hdf5 file.

`urbanaccess.network.save_network`(urbanaccess_network, filename, dir='data', overwrite_key=False, overwrite_hdf5=False)

## 5.2 Compute the closest node for each census block

### 5.2.1 Load census block data

Use simpledbf to load the census block data .

Input is the file path of dbf file in the shapefile folder

Output is a dataframe stores the information of census block.

```
dbf = Dbf5('file_name.dbf', codec='utf-8')
```

### 5.2.2 Compute the closest node

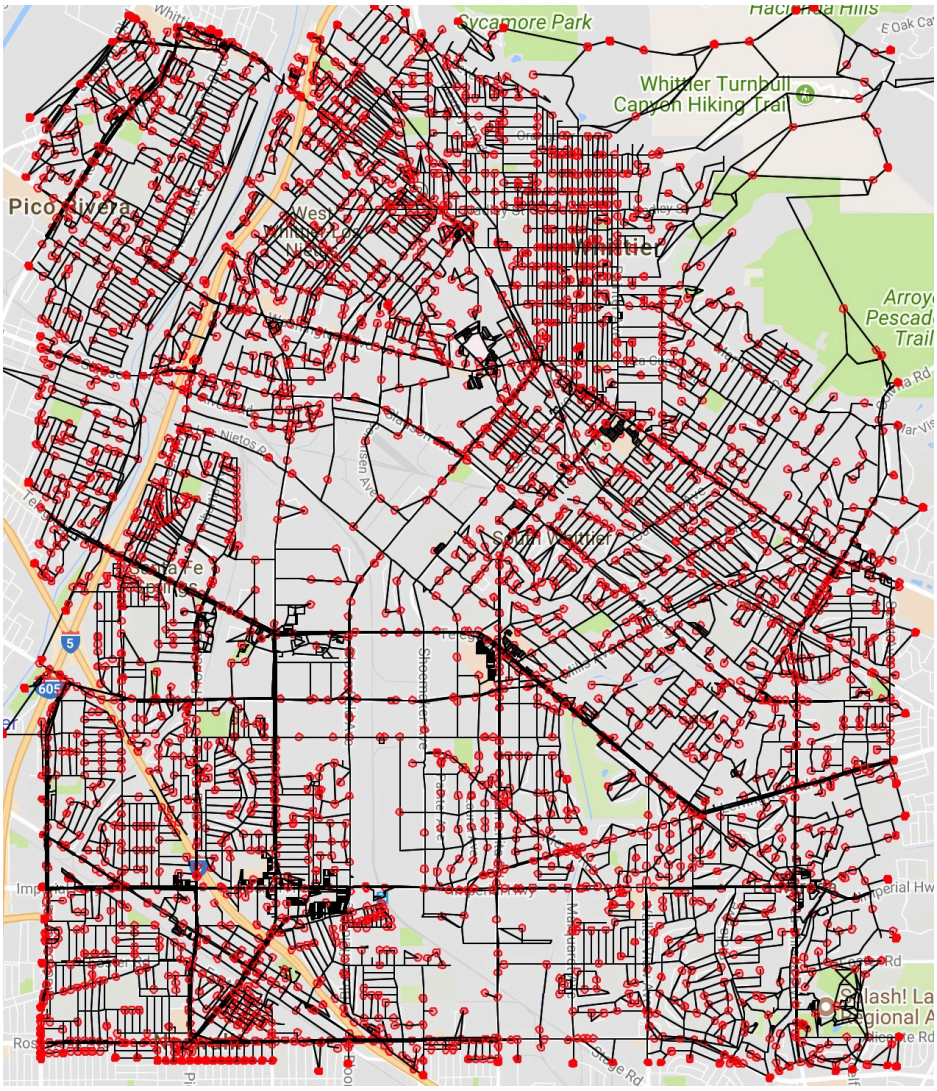
Use an api of Pandanas to Assign node_ids to data specified by x_col and y_col.

Input is a series of locations of the central points of census blocks.

Output is a Pandas Series of node_ids for each location in the input data.

`get_node_ids`(x_col, y_col, mapping_distance=None)

In the figure below, the red points are the chosen closest nodes to the central points of census blocks.




## 5.3 Compute number of reachable jobs

### 5.3.1 Load employment data

The employment data is a csv file. The python library pandas offer interface "read_csv" to read it.

The input is the file path of the employment file.

The output is a dataframe that stores the employ data in the file.

### 5.3.2 Aggregate number of jobs

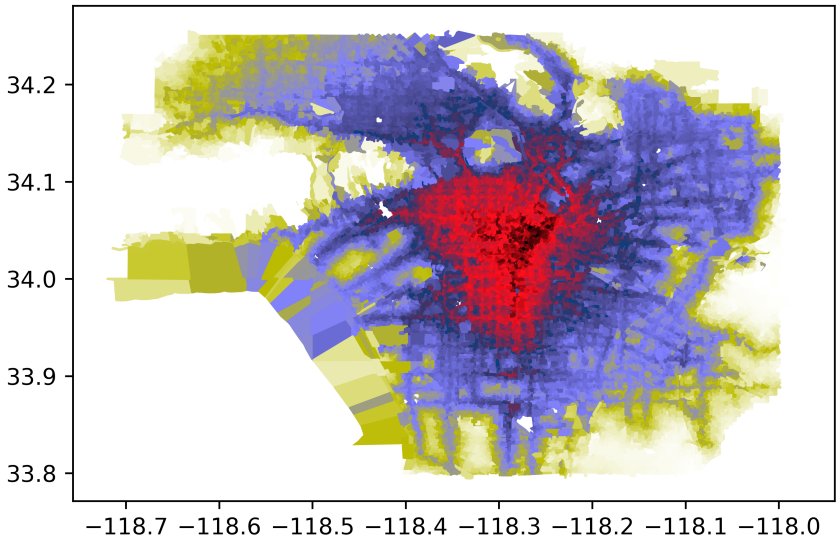
Pandana offers function to aggregate information for every source node in the network within the specified distance. In our case, we aggregate the number of jobs in the integrated network. We could set the distance as 20 minutes ,40 minutes and 60minutes.

The input is the dataframe that stores the employment data.

And output is a Pandas Series for every origin node in the network, with the index which is the same as the node_ids passed to the init method and the values are the aggregations for each source node in the network.

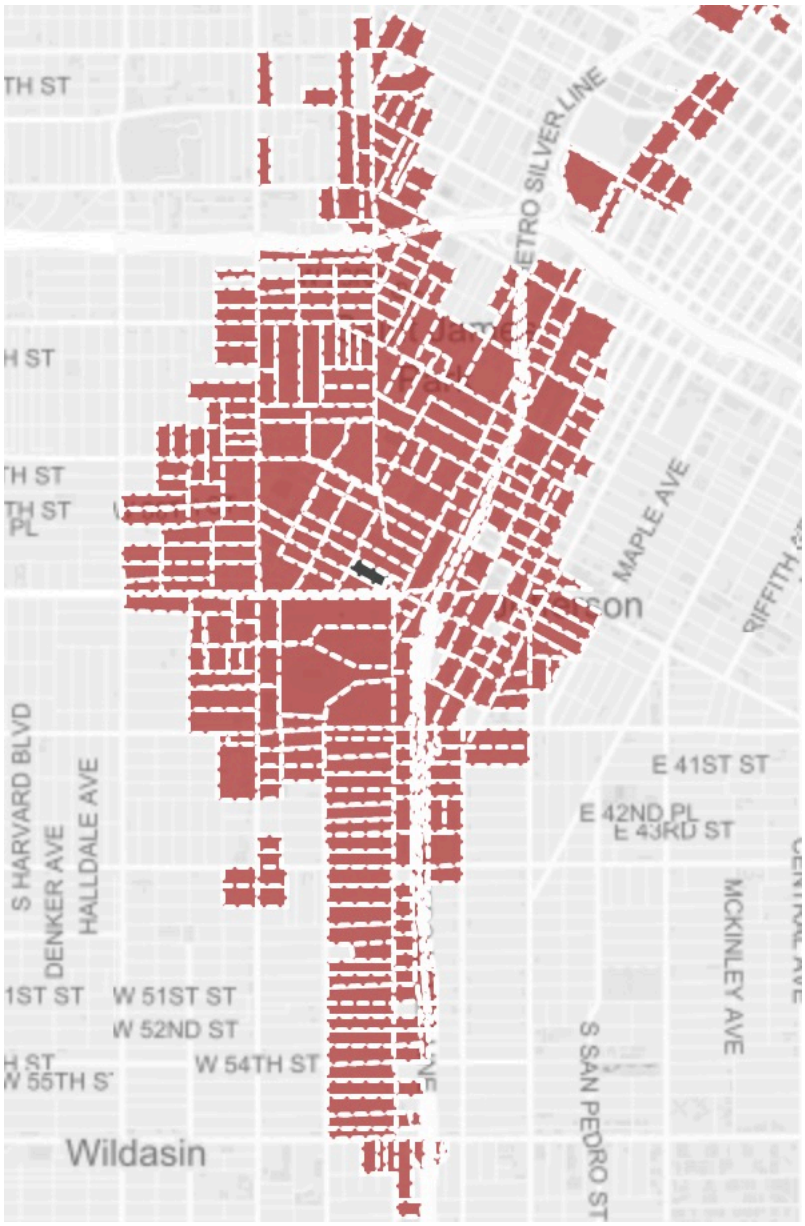
`aggregate`(distance, type='sum', decay='linear', imp_name=None, name='tmp')

The figure below is scatter graph of number of reachable jobs. The area in yellow can reach less jobs within 60 minutes. The area in red can reach more jobs within 60 minutes.




## 5.4 Compute number of reachable census blocks

In this step, we hope to get all the blocks a person can reach within a given time from a node in the network. For each node in the network, we run a bfs algorithm. When distance from source to a node is beyond a given distance, we discard it and would not put it in the stack. The algorithm will stop when the stack is empty. The result is as follow,




The black census block is the source block, all the brown blocks are those we a person in the black block can reach within 20 minutes by taking bus and walking.

## 5.5 Create json files for demo

Now we would generate some json files, which would be used by the demo.

5.5.1 Census block

For each census block, the demo would show the number of jobs it can reach within 20 minute, 40 minutes and 60 minutes respectively. To draw a block in the graph, we also need the boundary nodes of census blocks. The json file has the following format,


```
{
  "type": "Feature",
"properties": {
  "min20": 12075,
      "min40": 105884,
      "min60": 783110,
      "id": "060373006004026"
  },
  "geometry": {
      "type": "Polygon",
      "coordinates": [[[-104.99404, 39.75621], …]
  }
}
```


5.5.2 Block group

In Los Angeles county, there are more than 100000 census blocks. We add a level of block groups to help to navigate a census block and improve the performance of the demo.

For each block group, we need information such as block group id and boundary nodes of block groups. This json file has the following format,


```
{
  "type": "Feature",
"properties": {
      "id": "060372932023"
  },
  "geometry": {
```

```
    "type": "Polygon",
    "coordinates": [[[-104.99404, 39.75621], …]
  }
}
```


#### 5.5.3 Reachable census blocks

To show reachability, we need to show what are the blocks each census block can reach within a given time. For each census block, we save the reachable blocks as a list in a json file named with the id of these census block. All those files are save in a folder.

### 5.6 Structure of the project

#### 5.6.1 Structure

The structure of the project is as follows,

```
bus_trajectory
├── README
├── input
│   ├── ca_od_main_JT00_2010.csv
│   ├── gtfs_bus/
│   ├── tl_2010_06037_bg10/
│   └── tl_2010_06037_tabblock10/
├── la_urbanaccess.py
└── output
    ├── block_closest_node.csv
    ├── blocks.json
    ├── groups.json
    ├── index.css
    ├── index.html
    ├── index_one_level.html (optional)
    ├── jobs_all_20.csv
    ├── jobs_all_40.csv
    ├── jobs_all_60.csv
    ├── nearest20/
    └── net.h5
```

The main file of the project is la_urbanaccess.py. And the demo files(html and css files) are put in the output folder.

### 5.6.2 Run

```
python la_urbanaccess.py
```

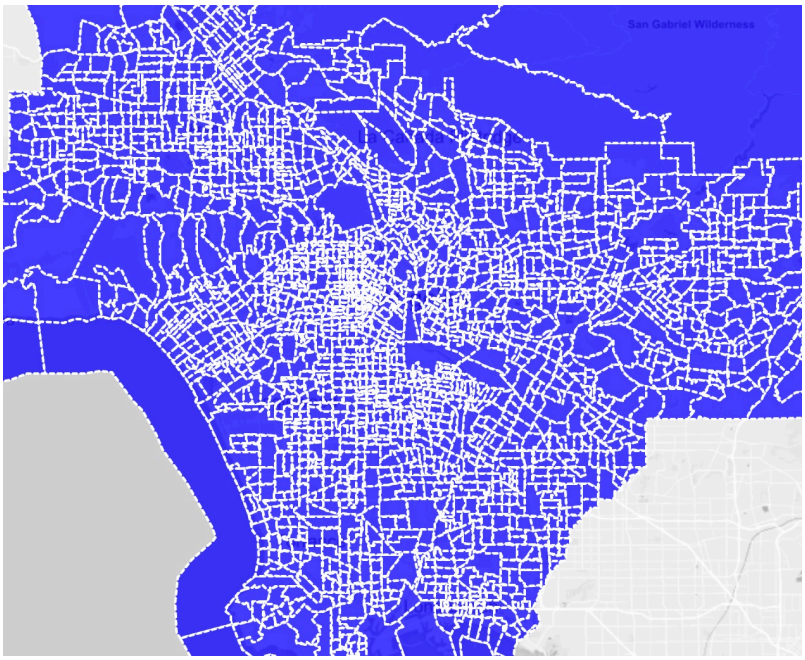
For small bounding box, it takes less than 10 minutes.

For big bounding box, it takes many hours.

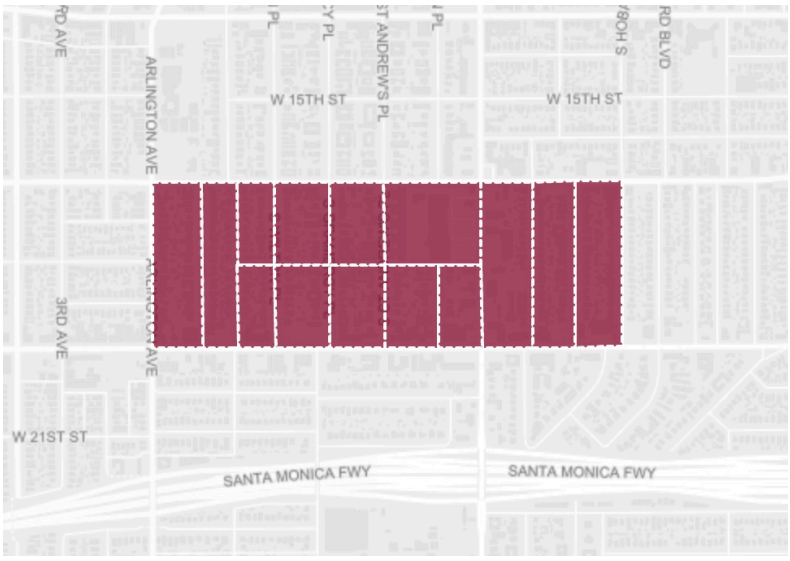
# 6 Demo

The files and folders that are used by demo are blocks.json, groups.json and nearest20, which would be generated in the output folder. After getting those output files,

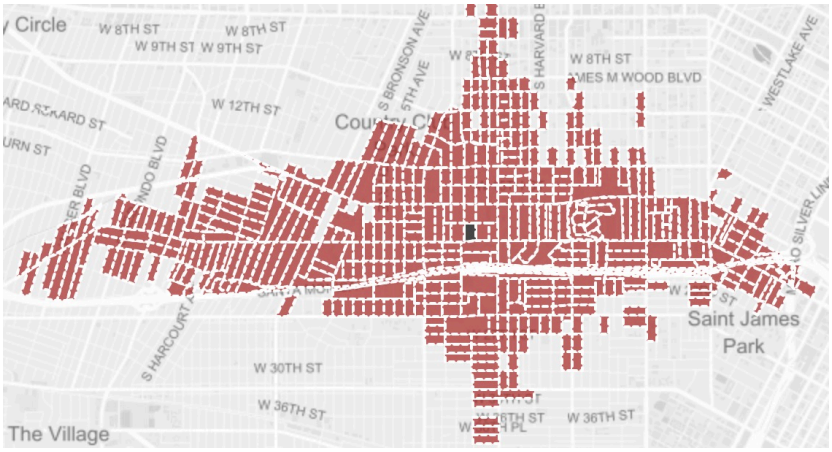
a) open the index.html with Firefox.




b) Click a block group, it shows all the census blocks in this block group.




c) Click a census block, it shows all the census block it can reach.




d) Now click any census block again, it will return to step 2.

e) In the census block level, double click, return to the block group level.