Overview of the Framework

Data Reader (reader.py)

This script reads in demand data, time matrices and current system scheduling and stores each respective data source

First Stage (first_stage.py)

This script runs the first stage of the optimization process and outputs the generated routes to a json file

Second Stage (framework.py)

Central file that contains all of the functions, algorithms and classes needed to perform analysis or update data

Implements an Optimizer Class with input variable *current_system* which is a boolean that, if it equals True, makes the Class using the current routing system, and if it equals False, makes the Class using the output data from the first stage. The Optimizer Class has a subclass, Route, which contains detailed information and functions regarding individual routes.

Optimizer Class Functions and Attributes

Attributes:

self.routes: list of Route Objects

Route Class

self.routes: list of stop ids

self.routes.schools: list of school ids the route was created to serve

self.routes.old_routes: list of routes that were combined to create this route

self.routes.load: load metric (from maximum demand data)

Functions:

optimize(n):
This function combines routes until the number of routes equals n.

clean_all(n, aggressive):
If aggressive is set to True: The function rearranges each route n times by iteratively choosing the next closest stop in the route to the current stop. This will likely drastically change the ordering of each route.

If aggressive is set to False: The function rearranges each route n times by looking at each stop in the route and seeing if switching that stop with any other stop on the route would lead to a reduction in total time of the

route. While this function is not guaranteed to preserve each stop to school requirement in the route, in general it seems to do so.

summary():

This function outputs summary statistics for the system including

- Number of Routes
- Total Travel Time of the system
- Average Travel Time for the system
- Longest RouteTravel Time
- Unsatisfied Demand
- Unsatisfied Demand AC

graph(routes):

This function plots the given routes over a map of DPS

remove_stop(stop):

This function iterates through all of the routes in the system and removes the stop from all of the routes

output_system(workbook, worksheet):

Takes the current route and summary information for the system and outputs the data to an excel file

output_routes_vrp(workbook, worksheet):

Takes the current route and summary information for a generated system and outputs the data to an excel file

output_routes_curr(workbook, worksheet):

Takes the current route and summary information for a current system and outputs the data to an excel file

load_summary():

Returns summary statistics pertaining to the load of each route in the system, including

- Load for each route
- Load per stop for each route
- Load per school for each route

Here, load is the estimated number of students who would use the route, calculated from the maximum demand data.

Other Functions

reader():

Calls the reader.py file and stores data files
Only needs to be called if the current route excel file (Shuttle Schedule with Targets) has been changed

first_stage(t):

Calls the first stage of optimization and updated data.json. The higher t is the better the optimization will be, but the longer it will take. We recommend t no lower than 3 and usually use t=10

unsatisfied_demand(routes):

This function iterates through each stop to school requirement in the demand data file and checks to see if, given infinite bus capacity, the students at each stop could get to their respective schools on any of the system's routes. It returns the number of students that were unable to get to their schools and how many stops those students were left at.

unsatisfied_demand_ac(routes):

This function is the same as unsatisfied_demand() except that each time the function determines that there is unsatisfied demand at a given stop to school pair, it checks to see if the current system meets this demand and only includes those students as unsatisfied if the current system does satisfy the demand.

verify_system(routes):

Checks the maximum demand data to see if the given routes would satisfy all of the demand (assuming buses have infinite capacity). Returns True if the routes do satisfy the demand and False otherwise. *verify_system(routes) will return True if unsatisfied_demand(routes) returns 0 students left behind

verify_against_current(routes):

Checks the maximum demand data to see if the given routes would satisfy all of the demand (assuming buses have infinite capacity). If there are students who are not served by the routes, it checks to see if the current system serves them. Returns True if the routes do satisfy all the demand that the current routes do and False otherwise. *verify_against_current(routes) will return True if unsatisfied_demand_ac(routes) returns 0 students left behind

Files Necessary for the Framework

Student Count by Bus Stop

By editing this file you change the demand data and time matrices used for both the current and new system analysis

Shuttle Schedule with Targets

By editing this file you change the current routes

Compiled Shuttle Count

Created to match stop ids and school ids

Files created by the Framework

current_routes.json

Contains the routes of the current system (to update, change Shuttle Schedule with Targets and then call reader())

data.json

Contains the data from the first stage of optimization (to update, call first_stage())

denver_entire.graphml

Used to graph the routes

Other Files

FNE Shuttle Ridership SY 19-20 and Key

Both contains useful summary information about the current system

To Update Current System or Alter Input Data

1. Change the Data File "Shuttle Schedule with Targets" to the new times
2. Change the Data File "Student Count by Bus Stop" to the new demand data
3. If any stops have been added then the following files need to be updated accordingly
    a. Compiled Shuttle Count
        i. Stop Key Sheet
        ii. School Key Sheet
    b. Student Count by Bus Stop
        i. Time Matrix Sheet
        ii. Success Express Summary Sheet
4. Run the following commands in Python
    a. first_stage(10)
    b. reader()

To Change Region to NNE

1. Recreate the following files with NNE data
    a. Student Count by Bus Stop
    b. Shuttle Schedule with Targets
    c. Compiled Shuttle Count
2. Edit* first_stage.py to read in NNE excel files on the following lines
    a. Line 35 (Time Matrix in Student Count by Bus Stop)
    b. Line 39 (Demand Data in Student Count by Bus Stop)
    c. Line 52 (School Key in Compiled Shuttle Count)
3. Edit* reader.py to read in NNE excel files on the following lines
    a. Line 34 (Shuttle Schedule in Shuttle Schedule with Targets)
4. Edit* framework.py to read in NNE excel files on the following lines
    a. Line 75 (Time Matrix in Student Count by Bus Stop)
    b. Line 80 (School Key in Compiled Shuttle Count)
    c. Line 81 (Success Express Summary in Student Count by Bus Stop)
    d. Line 641 (Success Express Summary in Student Count by Bus Stop)
    e. Line 662 (Demand Data in Student Count by Bus Stop)

f. Line 663 (School Key in Compiled Shuttle Count)

* Documentation for read_excel (Pandas) can be found online. Note that sheet_name parameter must be correlated to the correct sheet (the first sheet is selected with sheet_name = 0, the second sheet is selected with sheet_name = 1, etc.)

General

To use the framework, you first have to create an object. This is done by

new=Optimizer()
new.create_route_objects()

for the new system, and

curr=Optimizer(current_system=True)
curr.create_route_objects()

for the current system. Then you will be able to call any of the functions in the "Functions" section by "the name of the object".the_function().

Ex. curr.summary()

You will also be able to get any attributes of the system in the same way (but without the parentheses).

Ex. curr.routes

Any of the functions listed in "Other Functions" can be called without the object name in front of them.

Ex. verify_system(curr.routes)

Examples can be found in example.ipynp