**Documentation for Alabama Water Project: Wastewater Layout Generator Code**

This document gives an overview of the wastewater layout generator developed by the Columbia Water Center in 2023. This code was developed to assist municipalities in weighing the costs of decentralized wastewater systems, particularly in rural areas. Given a location and cluster number this code will output a proposed piping layout for a decentralized wastewater collection system as a shapefile, a cost estimate with breakdowns for various components of the system as an excel file, and proper sizing/placement of pumps and pipes. This code can create a layout for any system in the United States. **However, the current cost numbers are for Uniontown, AL. Make sure you update the cost information for you specific location before running this code**. The following documentation consists of an overview on how to use the tool, the assumptions made to develop this tool, and descriptions of each function. If you have any further questions please email [yk2862@columbia.edu](mailto:yk2862@columbia.edu).

**How to Use the Code:**

The workflow for this tool is divided into two python files: main\_functions.py and optimization\_model\_functions.py. The latter file consists of all the optimization models that will size the pipes and pumps for a wastewater system layout. The former will produce the layout, consisting of a treatment plant location, graph network with nodes (buildings producing wastewater or a treatment plant) and edges (pipelines connecting said nodes).

In order to run the model have your directory set to whatever folder optimization\_model\_functions.py is in. Make sure you also have all the necessary packages downloaded. The gurobipy package can be obtained for free if you have a university email, but your license will need to be renewed frequently. The other packages should be able to be downloaded via the anaconda command prompt or pip install without much difficult.

When running the optimization \_model\_functions.py file in a python IDE you will be prompted with a series of questions in the console (see Figure 1). The bounding box is the rectangular area encompassing the locations you would like to build a wastewater treatment layout for. The minimum longitude is the most westward point of your bounding box, the maximum longitudinal coordinate is the most eastward point, the minimum latitude is the southernmost point, and the maximum latitude is the northernmost point. The state refers to one of the 48 contiguous U.S. states. **Note: when writing down the state that your municipality resides in, be sure that you capitalize the first letter and you write the name correctly. If the name is not inputted in this format there will be an area**.

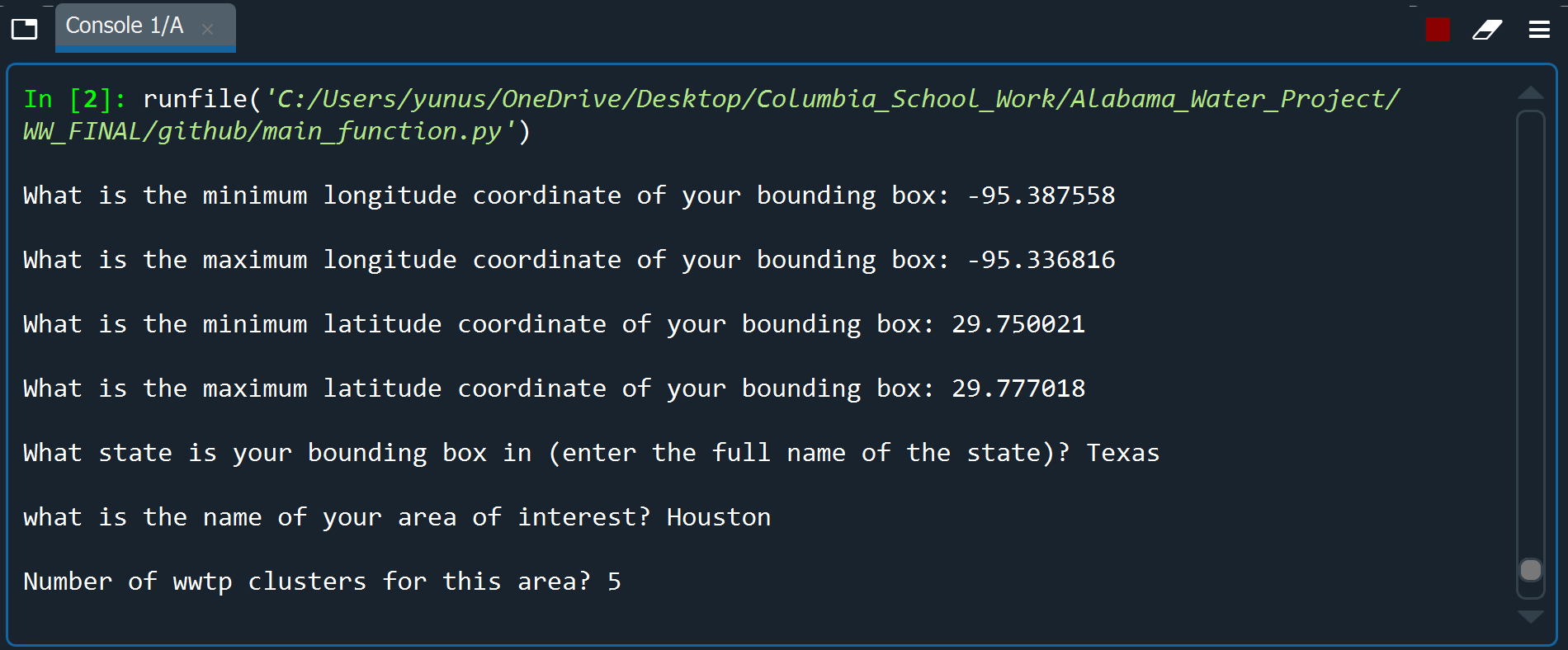


Figure 1: Example of what it looks like when the code is running.

The model creates the layout (see Figure 2 for a diagram of the full process) by taking the min/max coordinates for longitude and latitude, and extracting both the building footprints from the US Microsoft Building Footprints Database and the road network from Open Street Maps (OSM). The building data is directly downloaded from the github page for the US Microsoft Building Footprints project, outputting both the centroid of each building and the area. After this, the road network is used as the skeleton for the wastewater network layout. The code will connect any unconnected segment within the bounding box, and then will simplify the road network for the sake of reducing the number of redundant nodes, loops, and repeated edges. After the network is simplified, we iterate through each node of the network to break up any edge longer than 100 meters into 100 meter segments. Each node is then assigned an elevation. These elevation numbers come from the py3dep package which downloads a tif within the coordinates of the given bounding box. Each road node is assigned the elevation value within their corresponding pixel of a 10x10 meter resolution DEM. After this we finish pre-processing the road layout.

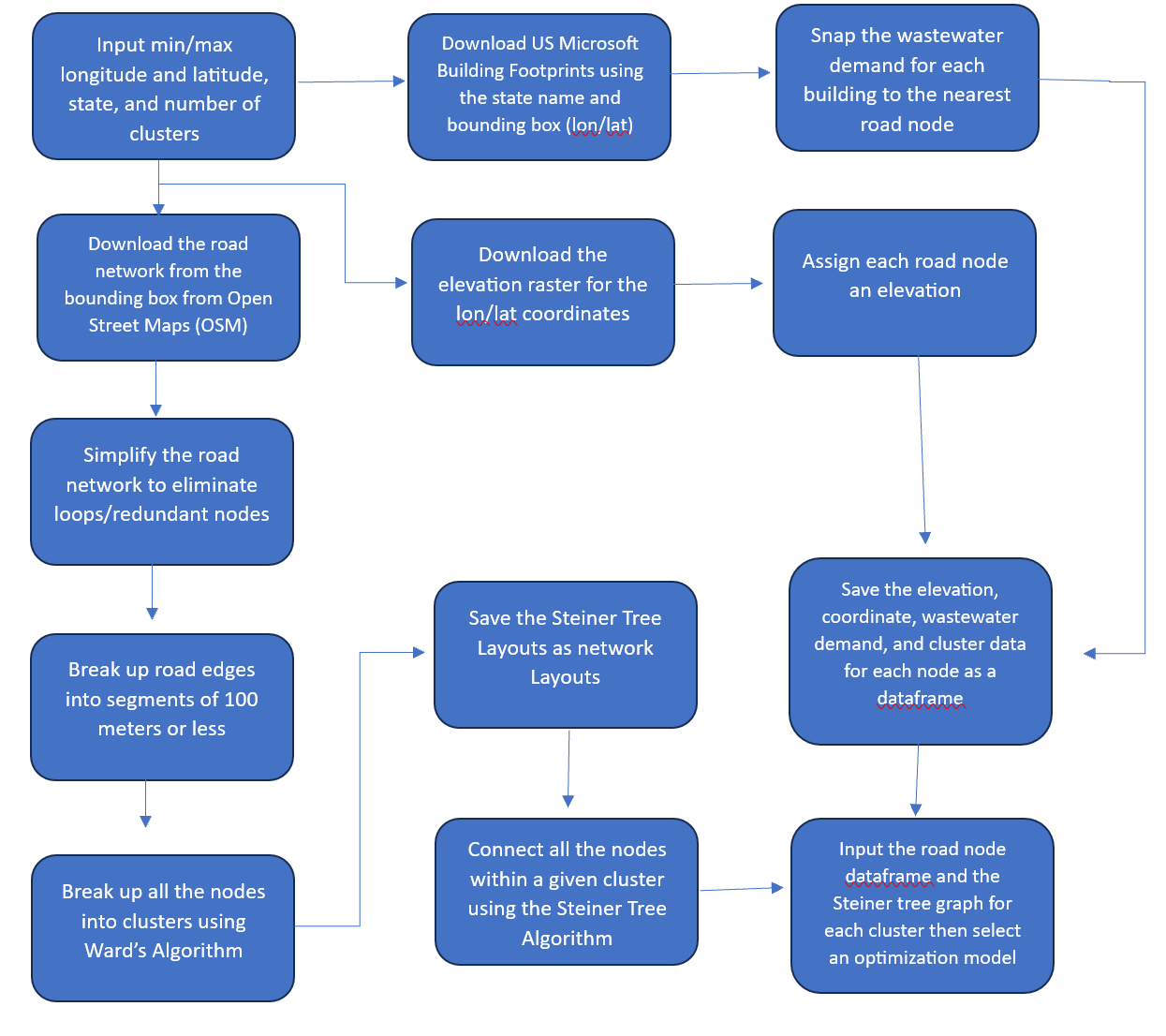


Figure 2: Workflow for the tool. Once the optimization model is selected the output is an excel sheet with the prices for the different sub-components of the wastewater layout.

Once we pre-processing the layout, we snap each building to the nearest road node (based on distance from the centroid). We assume that the wastewater demand for each building is the same (with an assumed node flow), but the building footprint areas could be used for demand per square foot if the user would like to rewrite the code for the wastewater demand. Now all road nodes for the necessary information to divide the network within the bounding box into clusters. Taking the number of nodes inputted by the user, the code uses the complete linkage algorithm (we used ward’s in our paper but unfortunately python did not offer that for the packages we could install) to divide all the road nodes into clusters. Once all the road nodes are broken up into clusters, we use the Steiner Tree algorithm to connect all the clustered road nodes through the pre-processed road network layout. This will output a branched network connecting all the nodes within each cluster. Both this network and a data frame containing the elevation, coordinates (lon/lat and espg 2163 projection), number of buildings snapped (wastewater demand), corresponding cluster, and building area snapped (square meters of buildings snapped to a given road node).

Now we have the graph for each cluster and all the necessary information for each point within the network. You will then be prompted. We input this information as well as the cost numbers into the tool (see Figure 3). Once the tool runs a csv for each cluster with the cost estimates for the capital costs for piping, capital costs for pumping, O&M costs for the piping/pumping, and treatment is output.

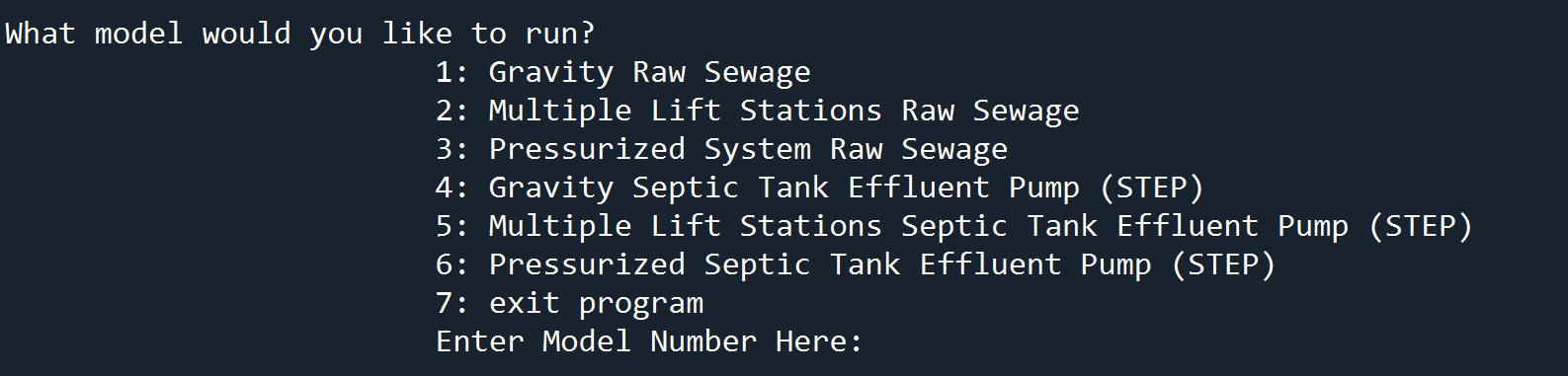


Figure 3: Example of the six optimization model options

**Interpreting Results:**

As mentioned in the section above there will be two outputs from running this tool. A series of shapefiles with the wastewater system layout for each cluster (helping visualize the placement of pumps and pipes, see the items highlighted in yellow in figure 4) and a csv file with the overall system cost (see the item highlighted in green in figure 4). The naming convention for the shapefiles is as follows: “(total number of clusters)\_cluster\_(cluster number)road\_arcs.extension”. Meanwhile the naming convention for the cost document is “(total number of clusters)\_clusters\_(name of area)\_(optimization model name).csv”.

Every cost estimate file will be broken up into four different objectives: Obj1, Obj2, Obj3, Obj4, and Additional cost + objective functions (see Figure 5). Obj1 is the excavation cost, Obj2 is the pump installation costs, Obj3 is the pipe installation costs (sometimes the pipe costs include excavation for a given diameter, in this case Obj1 will default to 0, see Figure 5), Obj4 is treatment costs, and the additional costs added to these four objectives include hometreatment costs and added post\_proc. The reason why these costs are added to the end is because they are a constant cost that comes with the clustered buildings. They do not need to be factored into the tradeoffs weighed by the optimization model.

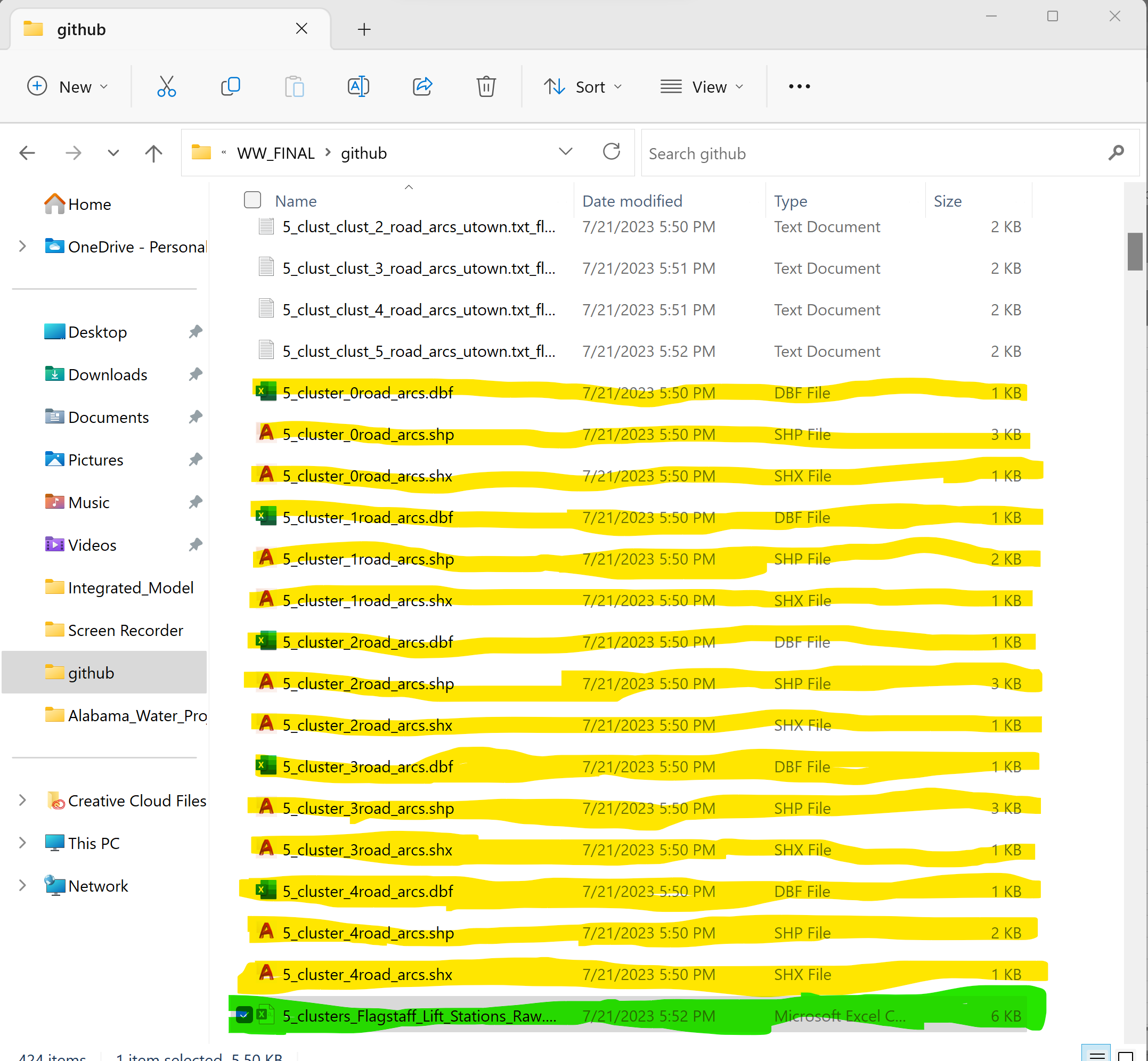


Figure 4: Outputs from the tool. Everything highlighted in yellow is a shapefile showing the layout of the system. The document highlighted in green is a csv containing the location of the pumps as well as the overall cost of the system.

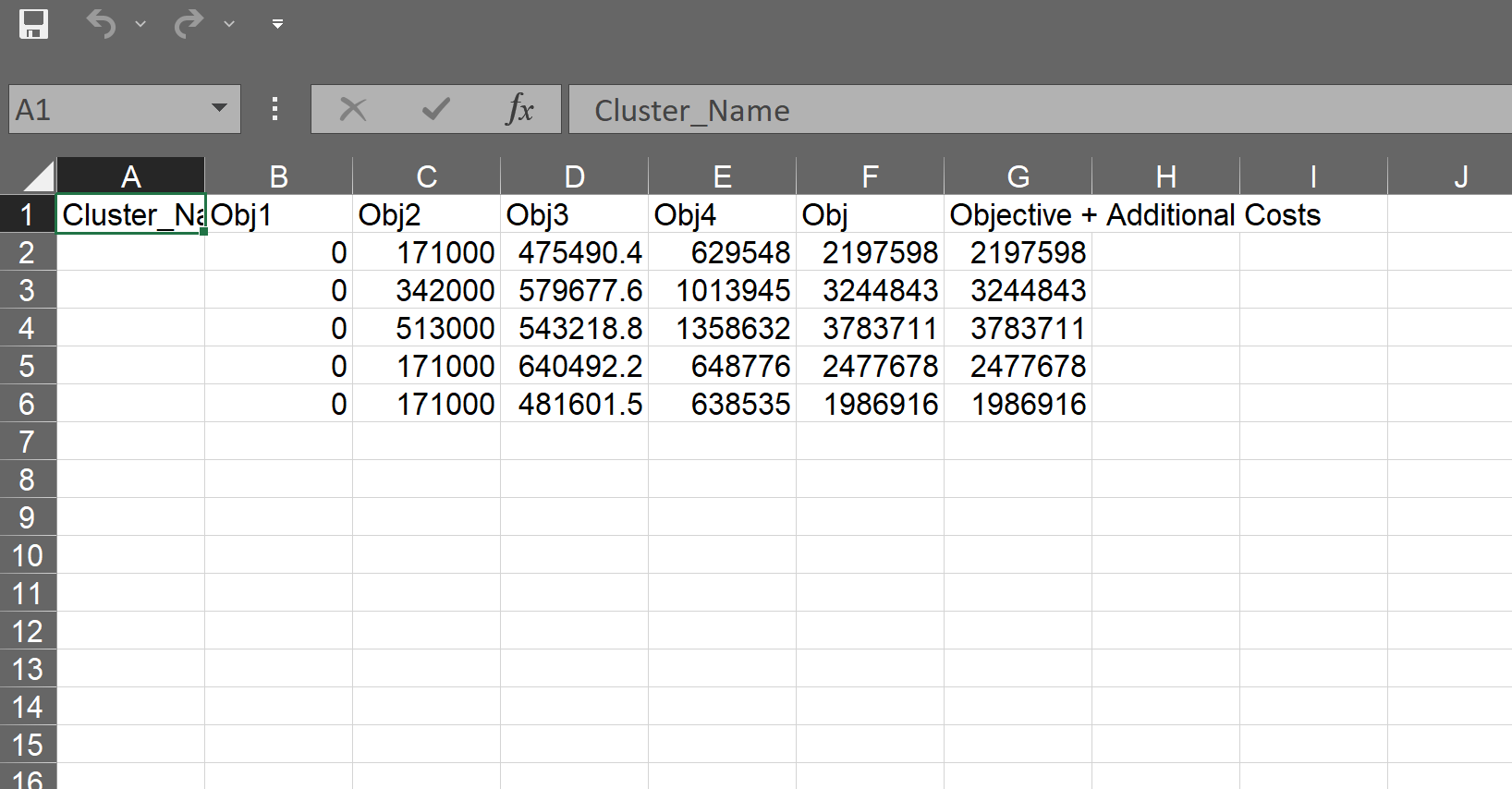
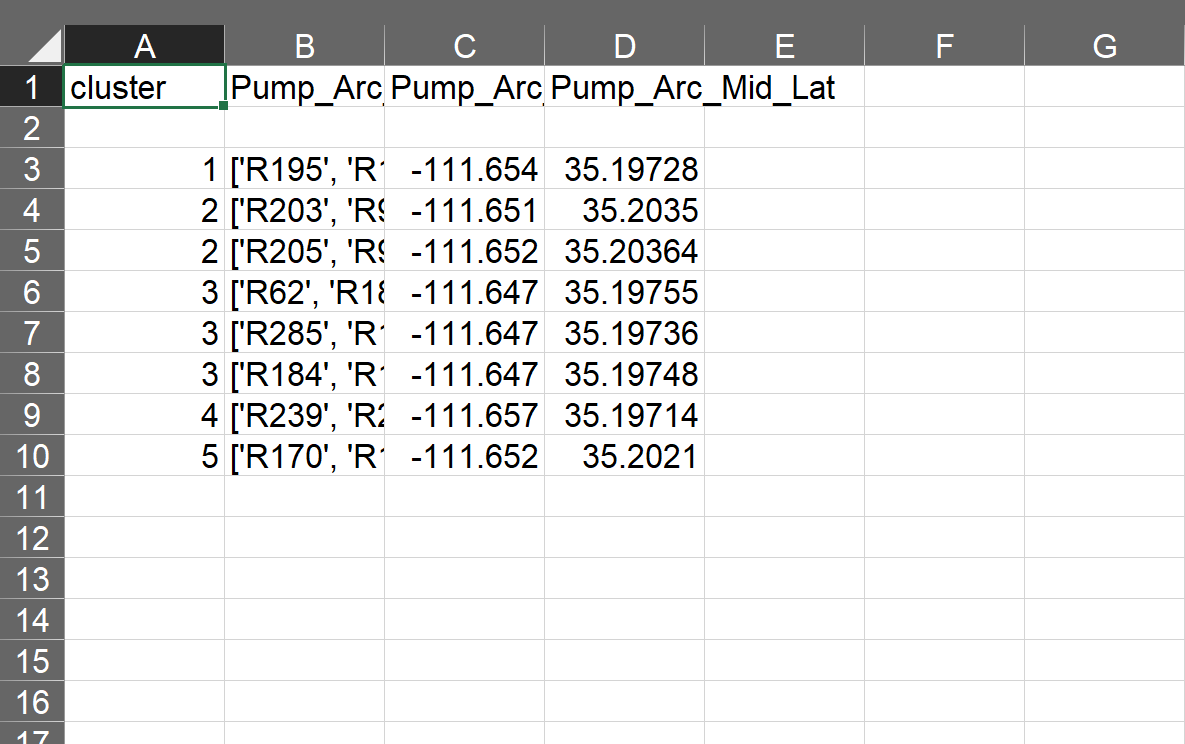
 

Figure 5: Lifecycle cost of a wastewater treatment system utilizing raw sewage lift stations for each cluster in the Flagstaff, Arizona (left) and placement of lift station pumps (right).

**Assumptions:**

**Cost Numbers Assumptions:**

The cost numbers were derived for Uniontown Alabama in 2020 (see Table 1). Cost numbers from local vendors were used to create cost functions to predict the prices of pumps and treatment plants based on flow rate, pipes based on diameter, and connections per household. For further guidance on how these numbers were derived refer to the following paper: “Optimizing Scale for Decentralized Wastewater Treatment: A Tool to Address Failing Wastewater Infrastructure in the United States” (Schwetschenau et. al). At the moment we do not prompt the user for cost numbers, so researchers must make sure to change the cost variable values within the code for their own locations/needs.

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Description** | **Value** |
| Arb\_min\_slope | [float; m/m] minimum allowable slope for a pipe segment | 0.01 |
| Arb\_max\_slope | [float;m/m] maximum allowsable slope for a pipe segment | 0.1 |
| pipesize | [list; m] allowable pipe diameters for a given model | Values within these lists vary. Example: [0.1, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45] |
| nodeflow | [float; gpm] wastewater demand of a given node | Two possible values: Daily Average—250 gpd/(60\*24) or Daily Maximum—1.8 gpm |
| Pipe\_dictionary | [dictionary; $/m] unit cost of each pipe for a given diameter | Two dictionaries: including excavation within the piping cost and excluding excavation (the values will not be listed here because it will take up too much space). |
| Excavation | [float; $/m3] unit cost of excavating for pipe | If we use the pipe\_dictionary including excavation costs excavation is 0. If we use the dictionary excluding those costs it is 25/ |
| Bedding\_cost\_sq\_ft | [float; $/m2] unit cost for 1 ft of bedding on top of the excavated channel | 6 (only applies to gravity flow systems) |
| Capital\_cost\_pump\_station | [int; $/unit] fixed capital cost for building a pump station at a given edge | 171,000 for gravity flow and lift station optimization models, 0 for pressurized models (they use smaller pumps that are incorporated into the home treatment cost). |
| Ps\_flow\_cost | [float; $/gpm] additional capital cost for the size of a pump station based on flow going through an edge | 0.38 for gravity and lift station models, 0 for pressurized models |
| Ps\_OM\_cost | [float; $/unit] net present value across the pump’s lifespan for operation and maintenance cost per unit | For the raw gravity and raw lift station models it is 359317, for the step gravity and step lift stations models it is 175950, for the raw pressurized model it is 10,279, and for the step pressurized model it is 2795. Note: unlike the capital cost the O&M for the pressurized model is not included in the home treatment cost. |
| Treat\_om | [int; $/unit] net present value across a treatment plant’s lifespan for operation and maintenance cost per unit | 237,000 |
| Fixed\_treatment\_cost | [int; $/unit] fixed capital cost of building a treatment plant | 44,000 for all the raw sewer systems, 18,000 for all the systems with STEP pre-treatment |
| Added\_post\_proc | [int; $/gpm] additional capital costs of building a treatment plant per additional unit flow | 8.52 for raw sewer systems, 26 for STEP pre-treatment systems |
| Collection\_om | [int; $/building] operation and maintenance cost for the connection at every individual house | 209 |
| hometreatment | [int; $/building] cost for the pre-treatment before the wastewater enters the collection system. This can include but is not limited to septic tanks and grinder pumps (pressurized system). | For the STEP gravity and STEP lift stations it is 2000, and for the two pressurized systems it is 5500. |

Table 1: All the cost variables considered for the wastewater treatment system.

**Layout Assumptions:**

Every cluster has exactly one wastewater treatment plant.

Whenever a network within a bounding box has several unconnected networks, the largest networks (the one with the most nodes) is selected as the default. The unconnected network that is closest to the default is then joined with an edge that will link the two closest points on the two networks. Once the two networks are connected, they became the new default network, and this process is continued until all the networks are connected.

The edges are broken into 100-meter segments in order to meet min/max slope requirements of 0.1-0.01 m/m, from the Alabama EPA. If edge segments are longer than 100 meters the slope might not be able to meet the minimum slope requirement. However, it should be noted that this requirement for pipe slopes is generally scene throughout the United States as a design requirement to ensure flow remains within a reasonable velocity range.

When clustering the nodes into respective networks we only cluster nodes with flow demand and ignore nodes that do not have buildings snapped to them. These nodes that are part of the network, but do not contribute wastewater demand are given a cluster number -1 in the dataframe to differentiate them from nodes with wastewater demands (which will be assigned values 0, 1, 2, 3… corresponding to the number of clusters).

Within the cluster we select the road node with the lowest elevation as the default site for the treatment plant. This is a heuristic used to save costs on pumping and/or excavation. Within the code we connect this road node to a dummy node that serves as the treatment plant, ensuring that all flow goes through one edge before entering the treatment node. We put the road node for the treatment plant at ground level and have the dummy node located 10 meters away at the same elevation (we have trouble with the slope constraints of the optimization model before we utilized a dummy node, so this was our work around).

Because we assume a branch network and node flow, we have predetermined the flow direction and value for ever edge within the network. We also assume the system has no redundancy.

**Optimization Model Assumptions:**

We assume the same wastewater node flow for each building. For the gravity and lift station models we use the total wastewater production from a four-family residential household over the course of a day, using the average hourly flow as our node demand. For the pressurized systems we also assume a four-family residential household, but we take the maximum flow over the course of a day (1.8 gallons per minute) as our node demand.

**Main\_function.py Functions:**

**Get\_elevation\_raster(bbox, name)**

Bbox: list (ex: [ymaxi, ymini, xmaxi, xmini])

The variable bbox is a list containing the lat/lon coordinates of the bounding with the largest latitude coordinate (Northernmost), then the smallest latitude value (Southernmost), then the largest longitude value (Easternmost), and finally the smallest longitude value (Westernmost).

Name: string (ex: “”)

The name is the name of the area of interest

This function will save a raster file (.tif extension) to your computer). This file encompasses a DEM covering your entire bounding box with a pixel resolution of 10 meters x 10 meters. The raster assumes that the coordinates are in lat/lon coordinates (EPSG 4326) and converts them into a EPSG 3857 projection. It will also output the name of this tif file (name.tif).

**Cut(line, distance)**

Line: Linestring Shapely Object

A line with a length longer than 100 meters

Distance: Float

Distance from the starting point of the line that a cut will be placed (in this case 100 meters away from the start).

This function will take a line and will divide it into two segments, one with a length of 100 and one that is the length of the rest of the line. These two segments are then returned as two separate Shapely LineStrings by the function.

**Split\_line\_with\_points(line, points)**

Line: Linestring shapely object

A line with a length longer than 100 meters

Points: Points shapely object

Points alone the aforementioned line

Will break up the Linestring into multiple Linestring separated by the specified points.

Example:

>>> line = LineString( [(1,2), (8,7), (4,5), (2,4), (4,7), (8,5), (9,18),

... (1,2),(12,7),(4,5),(6,5),(4,9)] )

>>> points = [Point(2,4), Point(9,18), Point(6,5)]

>>> [str(s) for s in split\_line\_with\_points(line, points)]

['LINESTRING (1 2, 8 7, 4 5, 2 4)', 'LINESTRING (2 4, 4 7, 8 5, 9 18)', 'LINESTRING (9 18, 1 2, 12 7, 4 5, 6 5)', 'LINESTRING (6 5, 4 9)']

**Get\_buildings(name, state, xmini, xmaxi, ymini, ymaxi)**

Name: String

Name of the area of interest

State: String

Name of the state the bounding box is inside

Xmini: float

Easternmost longitudinal coordinate of the area of interest

Xmaxi: float

Westernmost longitudinal coordinate of the area of interest

Ymini: float

Southernmost latitudinal coordinate of the area of interest

Ymaxi: float

Northernmost latitudinal coordinate of the area of interest

This function will save a txt file to your computer. This file will contain all the buildings within the specified bbox, their centroid coordinates, a designated name for each building (“B1, B2, etc.”), and the building area expressed as a float. The txt file is organized such that all this information for one building is on one line separated by comas (see figure 6 below). The function will also return the name of this txt file (“Centralized\_elevcluster\_name.txt”).

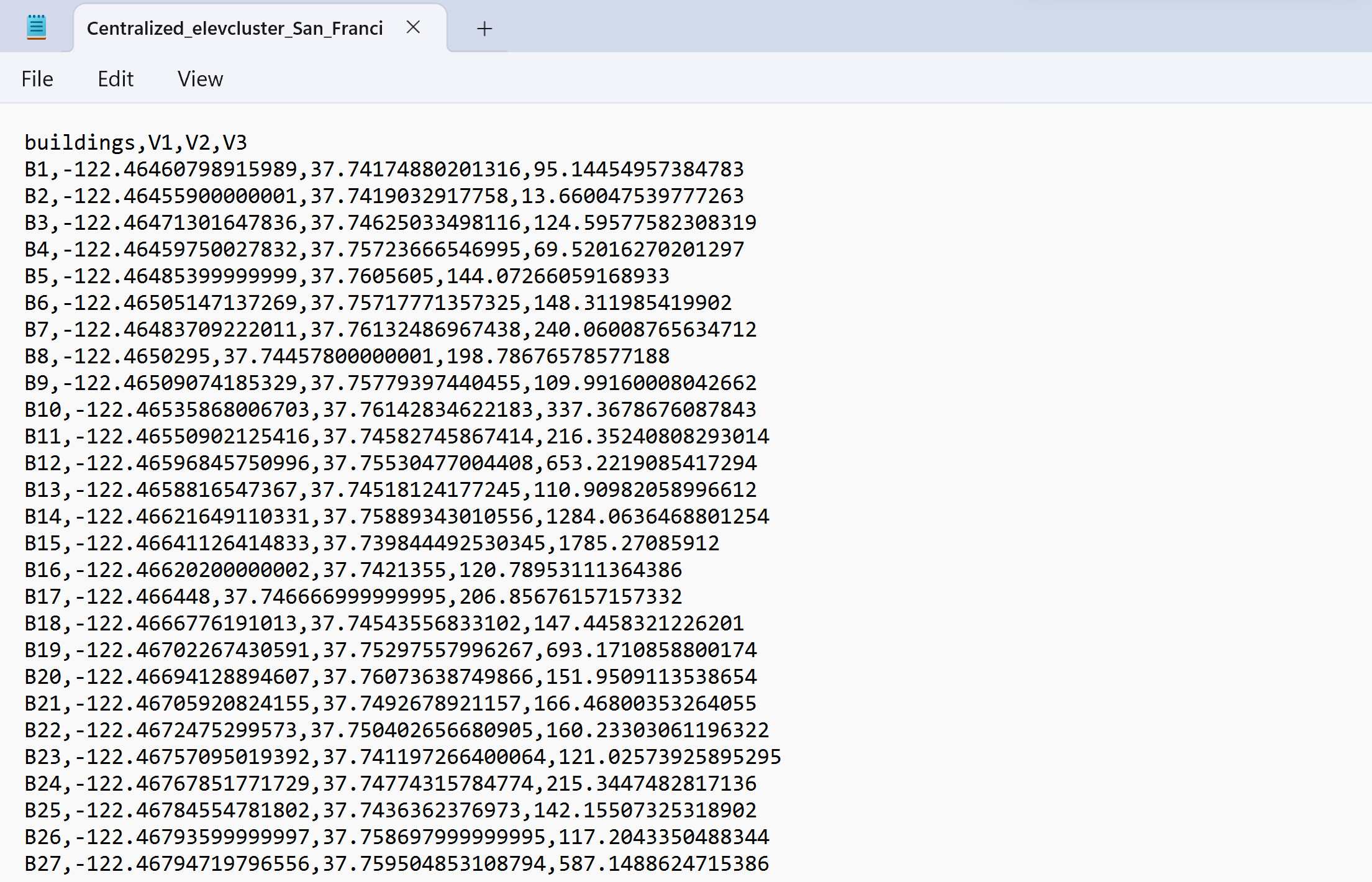


Figure 6: Example of txt file with building information for city of San Francisco

**Connect\_Graphs(G0, nodes, edges):**

G0: networkx graph

Graph of the road network with multiple unconnected segments

Nodes: geodataframe

Dataframe consisting all the node information within the G0 graph

Edges: geodataframe

Dataframe consisting all the node information within the G0 graph

This function takes an unconnected networkx graph, as well as the corresponding nodes and edges dataframes (these can be derived from osmnx graph\_to\_gdf function). The output of this function is a connected graph for all the road nodes.

**Arcs\_visualization\_many(ngroup, edges\_out\_li, nodes\_df0)**

Ngroup: int

Number of clusters for your area of interest

Edges\_out\_li: list

List for all the edges within a given network

Nodes\_df0:

Dataframe containing the node coordinates and the cluster they belong to

Saves shapefiles to the computer for the layout of ever cluster. This visualization function is used when using the get\_arcs\_many\_nodes function, which is meant to be used for road networks with over 5000 road nodes.

**Arcs\_visualization\_few(ngroup, mst\_list, nodes\_out)**

Ngroup: int

Number of clusters for your area of interest

Mst\_list: list

List containing networkx graphs of the branched wastewater network for each cluster

Nodes\_out:

Dataframe containing the node coordinates and the cluster they belong to

Saves shapefiles to the computer for the layout of ever cluster. This visualization function is used when using the get\_arcs\_less\_nodes function, which is meant to be used for road networks with less than 5000 road nodes. **Note: The way the code is written now only utilizes the get\_arcs\_less\_nodes function (even when dealing with over 5000 road nodes), due to the fact the get\_arcs\_many\_nodes function was creating unconnected road networks from connected road node networks.**

**get\_arcs\_many\_nodes (build\_shp, nodes, edges, dem\_name, G0, ngroup, city)**

Build\_shp: Dataframe

Data frame containing information on the building locations (centroid) and area

Nodes: Dataframe

Geodataframe for the road nodes

Edges: Dataframe

Geodataframe for the road edges

Dem\_name: string

Tif file for the area of interest

G0: networkx graph

Networkx graph for the road network

Ngroup: int

Number of clusters

City: String

Name of area of interest

This function takes all this information and creates ngroup number of networkx clustered graphs using the steiner tree algorithm. The steiner tree algorithm will create a branched networkx graph for each cluster. It also creates txt files which contain the edges within these graphs and a csv dataframe with the wastewater demand (n\_demand), coordinates (lat/lon as well as projected), elevation (meters), and cluster number for every road node. The csv and txts are saved to the users computer. The function outputs the name of the txt files for each cluster as a list. **Note: This function is not working. For some reason when it divides the graph into 100 meter segments it creates an unconnected graph. As a result we only use the Get\_arcs\_less\_nodes function. This function was meant to speed up the processing time for larger road networks by simplifying the network, clustering it, creating the branched mst networks, then dividing the edges within those branched networks into 100 meter segments. The Get\_arcs\_less\_nodes function is much slower because it creates its 100 meter segments before clustering and creating the branched networks. However, it has sufficed for most test cases, so we will use it for now.**

**get\_arcs\_less\_nodes (build\_shp, nodes, edges, dem\_name, G0, ngroup, city)**

Build\_shp: Dataframe

Data frame containing information on the building locations (centroid) and area

Nodes: Dataframe

Geodataframe for the road nodes

Edges: Dataframe

Geodataframe for the road edges

Dem\_name: string

Tif file for the area of interest

G0: networkx graph

Networkx graph for the road network

Ngroup: int

Number of clusters

City: String

Name of area of interest

This function takes all this information and creates ngroup number of networkx clustered graphs using the steiner tree algorithm. The steiner tree algorithm will create a branched networkx graph for each cluster. It also creates txt files which contain the edges within these graphs and a csv dataframe with the wastewater demand (n\_demand), coordinates (lat/lon as well as projected), elevation (meters), and cluster number for every road node. The csv and txts are saved to the users computer. The function outputs the name of the txt files for each cluster as a list. **Note: We use the Get\_arcs\_less\_nodes function for all road networks now. The Get\_arcs\_many\_nodes function was meant to speed up the processing time for larger road networks by simplifying the network, clustering it, creating the branched mst networks, then dividing the edges within those branched networks into 100 meter segments. The Get\_arcs\_less\_nodes function is much slower because it creates its 100 meter segments before clustering and creating the branched networks. However, it has sufficed for most test cases, so we will use it with all cases for now.**

**Get\_arcs(building\_txt, xmini, xmaxi, ymini, ymaxi, ngroup, city)**

Building\_txt: string

Name of the txt file with all the building locations (centroids) and areas

Ngroup: int

Number of clusters within the area of interest

Xmini: float

Easternmost longitudinal coordinate of the area of interest

Xmaxi: float

Westernmost longitudinal coordinate of the area of interest

Ymini: float

Southernmost latitudinal coordinate of the area of interest

Ymaxi: float

Northernmost latitudinal coordinate of the area of interest

City: string

Name of the area of interest

This function takes all the information above and it returns the names of ngroup number of txt files and the name of the dataframe (a csv file) consisting of all the necessary road node information (coordinates, n\_demand or wastewater demand, elevation, and cluster) for the optimization models to run.

**Main()**

This function runs the model so that the user is asked a series of questions (see Figure 1 and 3), and the model runs the get\_arcs, get\_buildings, get\_elevation\_raster, and get\_arcs\_less\_nodes functions (as well as the graph\_connect, cut, and split\_line\_with\_points functions if necessary) to get the layout. Then it runs one of the 6 optimization model functions to get a size the various wastewater system components and get a cost estimate.

**Optimization\_Model\_Functions.py Functions:**

**haversinedist(lat1, lon1, lat2, lon2):**

lat1: float

Latitudinal coordinate of point 1

lon1: float

Longitudinal coordinate of point 1

Lat2: float

Latitudinal coordinate of point 2

Lon2: float

Longitudinal coordinate of point 2

This function takes the lat and lon coordinates of two points and outputs the distance in meters

**readArcs(fileID):**

fileID: String

The name of the txt file with all the edges of a branched network for a cluster

Reads the txt file and outputs the edge pairs as a Nx2 numpy array

**findDistances(tree, dataframe):**

tree: numpy array

2xN numpy array consisting of all the edges within the cluster branched network

Dataframe: dataframe

The dataframe containing the coordinates of every road node within the area of interest

Outputs a dictionary where the keys are the edge names and the values are the distance inbetween them (using the haversinedist function)

**flip(arcList, out):**

arcList: list

A given edge

Out: String

The target node

This function takes an edge in the form of a list of two nodes and arranges the list such that the target node is the second value and the source node is the first value. The target node value is specified by the out variable inputted into the function.

**correctFlow2(arcsR, outletR):**

arcsR: Nx2 numpy array

This is the numpy array with all the edge pairs for a cluster’s branched wastewater layout

outletR: String

This is the road node with the lowest elevation within the cluster. It is the location of the treatment plant

This function goes through all the edges in the graph starting with the edges containing the outletR node. It ensures that all edges are draining into the outletR node such that the first value is the “from” node and the second value is the “to” node. Any edges that are not in this orientation will be flipped to ensure that they are facing this direction.

**gravity\_Raw(arcFlow, arcs, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num):**

arcFlow: Dictionary

Dictionary containing all the flow in each edge in m^3/s. This is found because we already know the direction of the flow (because we know the outlet in a branched network) and we know each node has the exact same wastewater demand. This only applies to a single cluster within your area of interest

Arcs: numpy array

2xN list of all edges

Nodes2: list

List of single element tuples, containing the name of every road node within a given cluster

Df: Dataframe

Dataframe with all the road node information for every in node all ngroup clusters.

Pipesize: list

Available pipesizes in meters in a list

Outlet\_node: string

Name of node where the treatment plant will be placed

arcDistances: dictionary

Distances between all edges across every cluster

Inflow: dictionary

Dictionary containing the wastewater production from each node in a given cluster

Bedding\_cost\_sq\_ft: float

Refer to Table 1

Excavation: float

Refer to Table 1

capital\_cost\_pump\_station: float

Refer to Table 1

Pipecost: dictionary

Every possibe pipe diameter is represented as a string and the value is the per meter cost of a given pipe diameter

collection\_om: float

Refer to Table 1

ps\_OM\_cost: float

Refer to Table 1

treat\_om: float

Refer to Table 1

fixed\_treatment\_cost: float

Refer to Table 1

added\_post\_proc: float

Refer to Table 1

Hometreatment: float

Refer to Table 1

Sheet: Workbook Object

Excel workbook sheet where cost information will be written down on (refer to xlwt Workbook package for more info)

Cluster: int

Cluster number within ngroups

Pumps: Workbook Object

Excel workbook sheet where pump placement and location information will be written down on (refer to xlwt Workbook package for more info)

Pumpcounter: int

Number of pumps that have been already placed

building\_num: int

Number of buildings within a given cluster

This function will return an updated pump counter value to accommodate any additional pumps added to the network, as well as write a row in the inputted excel spreadsheet for the cost numbers when designing a raw gravity collection system for a given cluster.

**multi\_LS\_Raw(arcFlow, arcs, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num)**

Exact same notes as the gravity\_Raw function above except this function designs a wastewater collection system with multiple lift stations for raw sewage.

**pres\_Raw(arcFlow, arcs, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num)**

Exact same notes as the gravity\_Raw function, except this function designs a wastewater collection system with pressurized flow for raw sewage.

**gravity\_STEP(arcFlow, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num)**

Exact same notes as the gravity\_Raw function, except this function designs a wastewater collection system with gravity flow for sewage that has undergone septic tank effluent pump pretreatment.

**multi\_LS\_STEP(arcFlow, arcs, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num)**

Exact same notes as the gravity\_Raw function, except this function designs a wastewater collection system with multiple lift stations for sewage that has undergone septic tank effluent pump pretreatment.

**pres\_STEP(arcFlow, nodes2, df, pipesize, outlet\_node, arcDistances, inflow, bedding\_cost\_sq\_ft, excavation, capital\_cost\_pump\_station, pipecost, collection\_om, ps\_OM\_cost, treat\_om, fixed\_treatment\_cost, added\_post\_proc, hometreatment, Sheet, cluster, Pumps, pumpcounter, building\_num)**

Exact same notes as the gravity\_Raw function, except this function designs a wastewater collection system with pressurized flow for sewage that has undergone septic tank effluent pump pretreatment.

**def get\_Results(model\_name,** **pipe\_dictionary, arb\_min\_slope, arb\_max\_slope, node\_flow, pipesize, exc, bed, capex\_PS, ps\_flow, ps\_OM, treat\_o, h\_treat, fixed\_treat, added\_post, collect\_o, xmini, xmaxi, ymini, ymaxi, aquifer\_file, ngroups, node\_df, name, arc\_files):**

Model\_name: int

Number that refers to the wastewater collection method (see Figure 3)

pipe\_dictionary, arb\_min\_slope, arb\_max\_slope, node\_flow, pipesize, exc (excavation), bed (bedding\_cost\_sq\_ft), capex\_PS (capital\_cost\_pump\_station), ps\_flow (ps\_flow\_cost), ps\_OM (ps\_OM\_cost), treat\_o (treat\_om), h\_treat (home\_treatment), fixed\_treat (fixed\_treatment), added\_post (added\_post\_proc), collect\_o (collection\_om)

You can look up all these parameters in Table 1 (note: if there is a parentheses next to a parameter you can find it in Table 1 under the name in parantheses)

Xmini: float

Easternmost longitudinal coordinate of the area of interest

Xmaxi: float

Westernmost longitudinal coordinate of the area of interest

Ymini: float

Southernmost latitudinal coordinate of the area of interest

Ymaxi: float

Northernmost latitudinal coordinate of the area of interest

Aquifer\_file: String

Name of shapefile with all aquifers in the United States (helps ensure the treatment plant location is able to discharge into ground water)

Ngroups: int

Number of total clusters

Node\_df: dataframe

Information about all the road nodes in an area of interest (elevation, if they can be selected as a treatment plant, coordinates, wastewater\_demand, etc.)

Name: string

Name of area of interest

Arc\_files: list

Names of txt files containing the edges of each clustered network within an area of interest

This functions takes all the information above, runs it through one of the optimization models, and then returns multiple shapefiles with the wastewater network layouts for each cluster within an area of interest and csv file with all the cost numbers and pump locations.

**Additional Considerations/Future Directions:**

This tool was tested in Uniontown, Alabama. The results of this case study are available in this paper: “Optimizing Scale for Decentralized Wastewater Treatment: A Tool to Address Failing Wastewater Infrastructure in the United States” (Schwetschenau et. al.). This will provide the formulation for the six optimization models as well as the efficacy of this tool in helping this municipality weight the costs and benefits of decentralized wastewater treatment utilizing technologies like RO, MBR, etc. If this tool is to be effectively utilized by operators, we need to connect the cost numbers to a database with regularly updates cost numbers from a nearby municipality or state. The option for users to input cost numbers or revert to a default for the state they are in could be an option to consider. Additionally incorporate water reuse into these decentralized wastewater systems could increase savings, and demonstrate that these technologies that can treat wastewater to reusable levels can be cost competitive with centralized wastewater collection systems.

**Acknowledgements:**

Thank you for Dr. Lall and Dr. Schwetschenau for their guidance and patience on this project. All optimization models and cost numbers are from “Optimizing Scale for Decentralized Wastewater Treatment: A Tool to Address Failing Wastewater Infrastructure in the United States” by Schwetschenau et. al. Additional gratitude towards Columbia World Projects and the Columbia Water Center for sponsoring this work.

Flow Diagram:

Snap the wastewater demand for each building to the nearest road node

Download US Microsoft Building Footprints using the state name and bounding box (lon/lat)

Input min/max longitude and latitude, state, and number of clusters

Assign each road node an elevation

Download the elevation raster for the lon/lat coordinates

Download the road network from the bounding box from Open Street Maps (OSM)

Simplify the road network to eliminate loops/redundant nodes

Save the Steiner Tree Layouts as network Layouts

Save the elevation, coordinate, wastewater demand, and cluster data for each node as a dataframe

Break up road edges into segments of 100 meters or less

Input the road node dataframe and the Steiner tree graph for each cluster then select an optimization model

Connect all the nodes within a given cluster using the Steiner Tree Algorithm

Break up all the nodes into clusters using the complete linkage Algorithm