OSAT: Open Source Accessibility Toolset

Accessibility Analysis Process Overview

David Greczner, Noblis

**Necessary Files:**





Shapefiles (.shp) for the boundaries of the Traffic Analysis Zones in a region you are looking at OpenStreetMap (.osm) file for the region you are looking at. Should be large enough such that it contains the full area of each TAZ.

 General Transit Feed Specification file set for each public transit agency which provides service in the area

TIGER/Line files (.RT1 and .RT2) for the area you are analyzing.





Census Data with regards to population, housing, and employment are needed for each TAZ in order for the accessibility equations to be used.

**Useful Software/Plug-ins:**





Quantum GIS – Used for calculating centroids

MapWindow GIS + Shape2Earth plug-in – Used for adjusting centroids and converting the ESRI Shapefile into a KML file

Osmosis – Used to obtain and merge OSM files. Google Earth – Helpful in visualizing KML files

Eclipse – Setting up OpenTripPlanner and using other tools

The tools provided by this project - Calculate accessibility and all other necessary files









**Obtaining Necessary Files:**

*Shapefiles -* Shapefiles of the Traffic Analysis Zones are typically provided by the local metropolitan planning organization for the area. For example, the Shapefiles for King County were obtained via the Puget Sound Regional Council website at [http://psrc.org/data/gis/shapefiles.](http://psrc.org/data/gis/shapefiles) However in some cases, when such data is not readily available, other resources may be of more use. The Shapefiles for the District of Columbia were obtained via [http://dcatlas.dcgis.dc.gov/metadata/TAZPly.html.](http://dcatlas.dcgis.dc.gov/metadata/TAZPly.html)

*OpenStreetMap Files* – Depending on the size of the region that you are looking at, two methods can be used to obtain the proper OSM file. If the region is small, it can be found using the online tool provided by OpenStreetMap, [http://www.openstreetmap.org/.](http://www.openstreetmap.org/) Under the export tab, you can zoom in on the region that you want and form a bounding box which can then be downloaded to your local system. If the region is too large, the site will not let you download the file. Instead you will have to use the site, [http://downloads.cloudmade.com/north\_america/united\_states/, w](http://downloads.cloudmade.com/north_america/united_states/)hich allows you to download the map file for an entire state. In the event you aren’t actually calculating the accessibility for the entire state, you can cut out the unnecessary regions using an open source tool called osmosis, which can be downloaded at<http://dev.openstreetmap.org/~bretth/osmosis-build/>

Using the command line, the following two commands are useful in manipulating large OSM files:

**Merging Files**

osmosis --rx firstFile.osm --rx secondFile.osm --merge --wx merged.osm

**Extract a Portion of OSM File**

osmosis --rx bigFile.osm --bounding-box left=-122.506729 bottom=47.1553345 right=-

121.78244 top=47.9323654 --wx smallFile.osm

*General Transit Feed Specification Files* – GTFS files can typically be obtained via the particular agency’s website. However, agencies often require some form of authentication or other actions before allowing access to their data, so it may be best to just download the files from [http://www.gtfs-data-exchange.com/.](http://www.gtfs-data-exchange.com/) The downside to this is that you have to count on someone else to update the files on the site, and you may not always have the most up to date version of the schedules.

*TIGER/Line Files* – These files can easily be obtained from the census bureau at [http://www.census.gov/geo/www/tiger/tiger2006se/tgr2006se.html.](http://www.census.gov/geo/www/tiger/tiger2006se/tgr2006se.html) The files are organized by county, and you may have to use a lookup table to locate the county or region that you would like to download.

*Census Data* – If you cannot locate the proper files via the local metropolitan planning organization, the data can be aggregated from block level data from the following sources: [http://factfinder.census.gov/servlet/DownloadDatasetServlet?\_lang=en &](http://factfinder.census.gov/servlet/DownloadDatasetServlet?_lang=en) <http://lehd.did.census.gov/led/>

**Outline of Process**

**1.** Find the centroids of each TAZ

**2.** Convert Shapefiles to KML

**3.** Download OpenTripPlanner

**4.** Build a Graph using OTP

**5.** Calculate Public Transit Trip Times

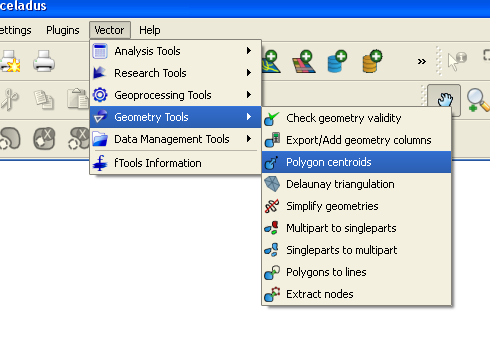
**6.** Calculate Automotive Trip Times

**7.** Append Census Data

**8.** Visualize Data

**1. Find the Centroids of Each TAZ**

Quantum GIS is by far the easiest way to calculate the centroids of each TAZ. After opening QGIS and loading the Shapefile (add Vector Layer), which contains the shapes for each of the TAZs, there is a feature built into the program that will allow you to output the centroids as a Shapefile.



**2. Convert Shapefiles to KML**

Now that you have a Shapefile containing the centroids for each TAZ, you’re going to want to open the newly created file in MapWindowGIS (View->Add Layer). In order to convert the Shapefiles to KML, I used the Shape2Earth plug-in for MapWindowGIS found at [http://shape2earth.com/shp2e.aspx.](http://shape2earth.com/shp2e.aspx) For my purposes I got away with using the trial version which only allows 500 shapes/points to be outputted as KML. So for King County where there were 530 TAZs, I just outputted the first 500, then the final 30, and merged the two together. If this becomes cumbersome you can either buy the full version or look for other free means of converting the data into KML.



Once the plug-in is installed, you’ll want to use it to

output the centroids as well as the TAZ boundaries as KML files. Then, open up the files in Google Earth so that you can see

where the centroids overlap with the actual streets and whether they should be moved. Using MapWindowGIS you can then move the centroids around as necessary to where they should be.

You can then continue to output the data as KML to see if the points are where you want them to be.

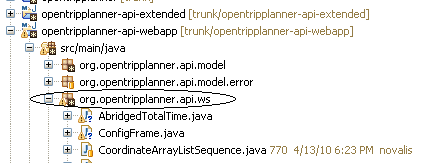
**NOTE:** In order for other tools to be able to use the centroid data, make sure that you specify that the TAZ data field needs to be included as the name field in the KML. If you don’t do this, the program won’t know which point goes to which TAZ!

Another option to consider would be to use the KML output of the centroids and then programmatically set the points to your liking. This way a particular point within a TAZ from which accessibility is going to be calculated can be more clearly defined, as opposed to placing it somewhere that looks correct.

**3. Download and Setup for OpenTripPlanner**

As useful as OpenTripPlanner may be, it is also

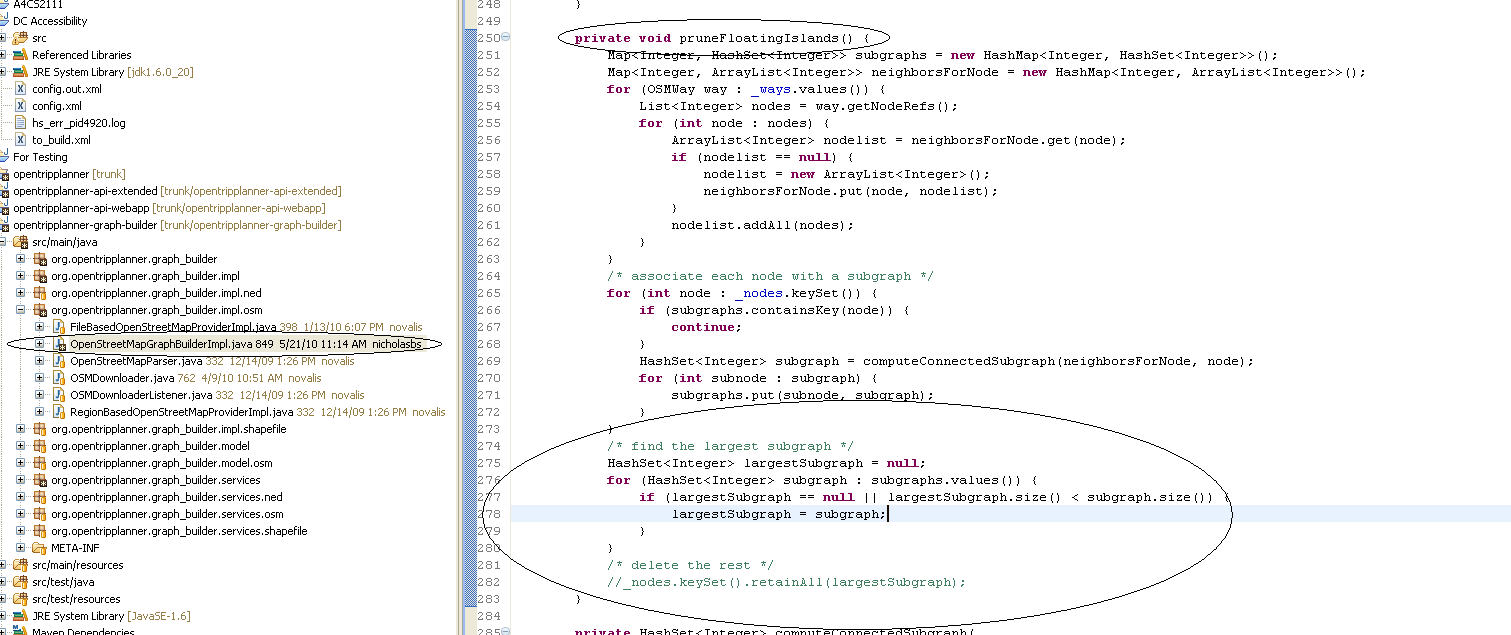
the most likely to cause the greatest amount annoyance and confusion. OTP is implemented in Java and, more



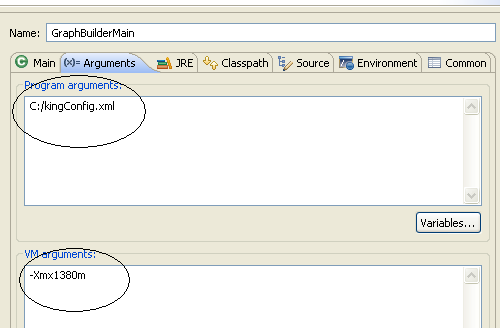
specifically, requires the use of the Eclipse IDE. Eclipse can be downloaded from [http://www.eclipse.org/downloads/packages/eclipse-classic-360/heliosr.](http://www.eclipse.org/downloads/packages/eclipse-classic-360/heliosr) Full details on how to get the project up and running can be found at [http://opentripplanner.org/wiki/Install.](http://opentripplanner.org/wiki/Install) Originally I had a number of issues setting up the project, however most of them were due to missing small details found in the installation notes from the site. Once the project is set up in Eclipse, you’re going to want to include the files in the opentripplanner source folder. These are the files that you will need in order to calculate the public transportation trip times. The most important files to make sure to include are ConfigFrame.java, RouteFind.java, TAZ.java, and XMLFilter.java.

**4. Build a Graph Using OpenTripPlanner**

The directions to create the Graph.obj file needed to use OpenTripPlanner can also be found on the website noted above. However before you do so, you may want to consider whether or not your map data has islands which aren’t apart of the primary network of roads, as was the case with King County which had Vashon Island. Under the original implementation of OTP, it took the largest connected component within the map and deleted any other smaller components. I submitted the bug report regarding this issue and the conflict with some maps with similar scenarios, though at this point the issue has not been addressed. In order to allow for islands to be incorporated into the graph you’ll need to alter OpenStreetMapGraphBuilderImpl.java.



Once you have your configuration correct and you have access to the GTFS files that you would like to incorporate into your graph, you need to write a configuration file. An example of such a configuration file is included in the files provided in the folder OTP Config



for GraphBuilder. To build the Graph object, you need to pass the path to the configuration file as an argument to the main class GraphBuilderMain.java. You’ll also want to alter the Java Virtual Machine arguments to increase the amount of memory you want to allocate towards the program. The OSM files are often fairly large and the program consumes quite a bit of memory in the process of building the Graph object. If not enough memory is allocated, the program will throw an exception and exit execution most likely as it is trying to write the object. Depending on the size of the files, the execution of the program can take anywhere from a

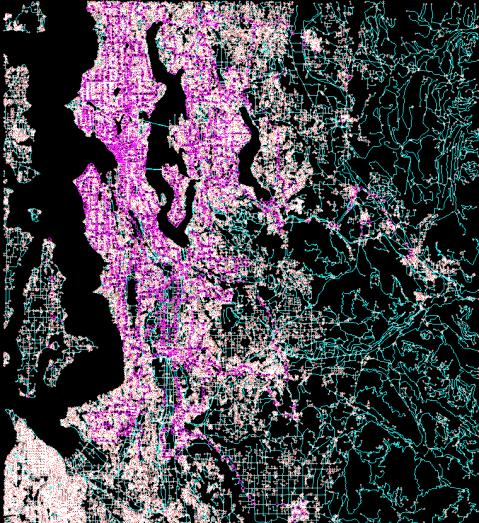
minute or so to several minutes.

If you would like to use the web application that comes packaged with OpenTripPlanner, you can read the directions provided by the link above. You can also pass the path to the Graph object to VizGui as an argument, which will then show the network of

roads and transit stops. I recommend using one of the two ways of

visualizing the graph to ensure that the results are what you want and that the proper files were included. The picture on the left shows the output of VizGui, where you’ll notice that the transit stops are pictured as small purple rings.

**5. Calculate Public Transit Trip Times**



If you have the source files set up in Eclipse, the file you’ll want to use to calculate public transit trip times is RouteFind.java. Once you run the program, a box will pop up asking you to select the proper directories to run from. You’ll also have the option of selecting to load the information stored in the configuration file, config.xml. A sample configuration file is included in the files provided to show the proper formatting necessary for the program to work. After you submit either your own settings, or the setting specified in the configuration file, the program will proceed to read through the centroid KML file and find the route between every single point in the file.

The current settings the program uses specifies that the user must *arrive-by* the given argument and if you want to change this or any other settings, you’ll want to alter the settings in the initRequest() method in RouteFind. Once your settings are correct and the program runs, it will incrementally output each TAZ to the specified directory. If the execution is interrupted or needs to be stopped for any reason, you can add a

simple for-loop in the source to get you back to the point you were previously at and continue execution. If

an exception is thrown during the course of execution, the most likely cause of the problem is that the point needs to be closer to either a road or one of the transit stops. If the first TAZ outputs with no issues, then the rest of the routes should be calculated without issue as well considering if they can both reach the same point, then they can obviously reach each other.

**6. Calculate Automotive Trip Times**

If you are running the source files for the automotive trip calculator, the main class is Interface.java. Once the program is run, it will ask for input and output directories as well as other configurations and has the option to just load the configuration stored in config.xml. As the program begins to run, if an alert message pops up saying that one of the two points may need to be moved, then the duration of the trip was considered to be abnormally short and such trips are often associated with points being misplaced or not properly apart of the network of roads. If the two points are actually just very close to one another, you can just ignore the message and continue execution of the code.

As the program runs, it will append the driving time data to the previously generated XML files created by OpenTripPlanner. Due to this dependency, transit trip times need to be calculated first before automotive trip times can be calculated. However, if necessary, the source files can easily be altered to create a default XML file and append the data as you’d like.

**7. Append Census Data**

Assuming that you have access to census data for each of the TAZs they can be added to the XML data files using the file DataAppender. In order to use this program you need to have an Excel file (.xls) with the TAZ in the first column and the data which you want to append in the second column. If the data you wish to alter requires alterations for all of the destinations from a particular TAZ, then the TAZ’s in the first column correlate to the destinations of the selected TAZ which will be altered. For other pieces of data, such as population data, the TAZ’s in the first column will be associated with individual XML files and each one listed will be altered and updated. As a default setting, any census data added will be listed as from 2010 in the XML file, of course this can be altered to change values for varying years. When adding census data, please note that it is not necessary to list all TAZ’s in the XML and that the program will only alter those that are specified in the Excel file.

**8. Visualize Data**

In order for the program to accurately visualize the data and incorporate the accessibility equations, make sure that the XML files that you plan on having it load are complete and accurate. If you run the program via the source code, the main file is Runner.java. Once the program is run, a window will open and ask for the path to the KML file of the boundaries of each TAZ as well as the directory which contains all of the XML data files. All the shapes in the KML file get drawn so if not all of the XML files match up, they will just be colored black and left out of the equation. So if you load the program and all of the shapes are black, then somewhere along the lines the zone number and the XML file are not matching up. If one of the metrics shows up as either all green or all red, there may be an outlier in your data, or the data itself may be missing and basing the calculations off a default value of 0. Currently when you choose to extrude the current model there is a placeholder for a future metric that is currently just called name. If it is chosen it will just extrude by the color of the current model, though the file ModelCalculator.java could easily be altered to add a new metric to extrude by. If you plan on altering any of the metrics or adding a metric, you’ll need to add a button for it in ControlPanel and then have the program do the actual calculations in ModelCalculator.

When you click around on the map of the GUI and the map seems to distort itself where it looks like it is only half there and it is at the bottom of the screen, or really any other painting error, it just seems like a glitch in Java’s graphics or the way it is interpreting the current values. Just zoom in and then out and the screen will be back to its normal viewing state. Another little glitch with the graphics is at the start of the program when the splash screen is displayed, sometimes it will look like the frame never closes. However, what is really happening seems to be some sort of repainting issue with the current frame and despite the frame actually being closed, it’s image gets painted on the other frame. If you do something to that frame

such as scrolling up, which forces a repaint, you’ll notice that the image goes away.

**Overview of Classes:**

**DC Accessibility**

*-calculations*

- Exporter

Takes the current model being visualized and using JDOM creates a KML file which is then outputted to the

specified directory. Used in both the main GUI tool as well as KMLBuilder to export the models in bulk when necessary.

- KMLDescriptor

The primary purpose of this class is to provide a means of describing what model is being selected to be outputted as KML. Essentially just uses whether each button is selected in the control panel and stores that information so that it can be used later for output.

- KMLReader

This class opens a KML file which contains the coordinates of the boundaries for each TAZ and parses the file in order to create a mapping between each zone and the Shape for the particular zone.

- KMLWriter

Primarily used with KMLBuilder, originally created in order to convert an XML file containing information about KML files and goes ahead and builds the specified files. Later altered to work with KMLBuilder to read to\_build.xml and outputs the KML files as listed in the configuration file.

- ModelCalculator

Using which buttons are selected in the control panel as input, determines which metric or model is being selected and using stored TAZ information calculates the metric and normalizes the value and stores the information as a color and maps it to the particular TAZ.

*- graphics*

- ConfigFrame

Creates the frame which pops up at the start of execution of the GUI tool and passes the data, whether it is the data loaded from the configuration file or the data selected manually, along to GUIMain.

- ControlPanel

Creates the panel that appears on the right hand side of the GUI and allows the user to select which model should be calculated. Calls on the ModelCalculator to calculate the model and passes that information along to the map to be painted. If adding a metric to be calculated, you’ll want to add a button in this class.

- DataPanel

Creates the panel on the top right hand side of the GUI which displays the information for the selected TAZ

- ExportFrame

Creates the frame that pops up when the user selects to export the current visualization to KML. Using the model that has been calculated and what the user selects to extrude the metric by, uses the Exporter class to output the data to the specified path.

- InitFrame

Creates the splash screen which is displayed while the XML files are loading and properly increments the progress bar as each file is loaded.

- InitPane

Reads in the Noblis logo as a BufferedImage and draws the image on the panel which will be added to

InitFrame and displayed while the XML files are being loaded.

- KeyPanel

The only purpose of this class is to create a panel which draws the range of colors which will be used in the heat maps so that the user has an understanding of what is a high and what is a low value.

- MainFrame

This class contains all of the code which is used to take all of the elements of the GUI and put them together and display them. If you want to change the arrangement of the GUI, you’ll most likely want to start in this class.

- MapPanel

Creates a panel which is used to draw each of the shapes for the TAZs and using the color mappings provided

by the ModelCalculator fills the shapes accordingly. Also contains code for how the screen coordinates are transformed into world coordinates.

- Viewport

Takes all of the shapes for each TAZ and using the union function of each shape, determines the most northern, eastern, western, and southern points to use as a boundary for setting the viewport and drawing the shapes to the screen.

*-main*

- COMvEmployment

Used to develop data which could be copy and pasted into Excel and analyzed, prints out a comparison of

accessibility and cumulative opportunity values for both public transit and automotive modes of transit for each TAZ.

- DataAppender

Helper program to allow the user to upload values from an Excel file and append the values to particular XML

data files.

- ErrorChecker

Sample class used to show how points gathered from WMATA’s online trip planner were compared against

actual values. Would need to change path for the actual file to read and compare.

- KMLBuilder

Helper program which allows the user to add configurations to a list and ultimately output, in bulk, particular

KML files to work with.

- PopulationHousing

Not part of any of the programs, merely serves as an example of how to possibly aggregate block data and then append that data to the specified directory of XML files.

- RegionAccessibilityCalc

Not a part of the main visualization, just a standalone program which reads the XML data files and calculates the average percentage of jobs reachable from each TAZ within a region.

- GUIMain

The main class which runs the GUI visualization program. Asks for configuration via a ConfigFrame which then initializes data via InitFrame and finally opens the MainFrame to display the data stored in the specified directory of XML files.

- StopChecker

Serves as a standalone example program which takes in acreage data for each TAZ as well as loads the list of transit stops and determines the average number of stops per acre for each TAZ. Could be used to determine which TAZs to use in an analysis.

**opentripplanner-api-webapp/src/main/java/org/opentripplanner/api/ws/**

- ConfigFrame

Essentially the same as the ConfigFrame listed above, although it is altered to account for the change in

required input from the user.

- DataManager

Not essential to calculating transit trip times, however it can be used to load an XML dataset and just prints out

the information stored for transit trips in a frame graphically.

- FileChecker

Again, not an essential program, though can be used to load a directory and it’ll check whether any transit

routes require a walking distance of over one mile.

- RouteFind

The main file for calculating public transportation trip times. Loads the Graph.obj and then uses the files contained in OpenTripPlanner to find the fastest route and stores the data in TAZ objects and also outputs all of the information to a specified directory.

- TAZ

Primary file for storing all transit route data. Anytime you need the ability to store more information, retrieve information, or alter the way that the information is outputted as XML, you’ll want to do it in this file.

- XMLFilter

A file filter used when reading a directory and seeking out files whose extension is .xml.

**Automotive Accessibility Calculator**

-*main*

- ConfigFrame

Very similar to the above listed configuration frames, though has been altered to only allow for input of the

necessary files and directories.

- GeoEncoder

Not used in the automotive calculator, and is a standalone program. Used to read in a KML file and a

TIGER/Line directory and output a text file which associates a road segment with each TAZ which could then be used in the ie.py script to check data against an online trip planner.

- Interface

Main file which is used to run the automotive trip calculator and append the data to the specified XML files. Loads the database of roads using TIGER/Line files and then parses the KML file to create a mapping between each zone and its centroid. It then takes those mappings and finds the fastest route between each of them and alters the appropriate XML file.

- RTFilter

File filter used to filter a directory down to only files ending with .RT1 and .RT2

-*mapping*

- Compass

Calculates the direction of a RoadSegment

- Edge

Class used to model the edge of a graph which consists of RoadSegments, the object contains “way” data which could be used to model one way roads.

- Node

Models a vertice in a graph which tracks which edges it connects to and also stores the incoming edge which is used for Dijkstra’s algorithm. If you want to limit which points can be connected by edges, you’ll want to use this class as a possible means.

- Point

Converts a point from TIGER files into latitude and longitude coordinates

- RoadSegment

Stores information about a particular RoadSegment including starting and ending points and the name of the actual road.

- ShapePoints

Very similar to the Point class, however does not divide the input latitude and longitude points and is used for storing the shapes of the TAZs.

-*routing*

- Database

Interface used to define the methods for the DefaultDatabase.

- DatabaseFilter

Interface used to define the methods for DefaultDatabaseFilter.

- DefaultDatabase

An implementation of Database consisting of road segments from the TIGER/Line database from the US Census. The TIGER/Line data consists of a separate set of files for each county in the US. In the file of Type

1, each record represents a segment of a one-dimensional feature such as a road, river, or political boundary. It is a road if its 3-character Census Feature Class Code (CFCC) begins with 'A'.

- DefaultDatabaseFilter

Simple filter for querying the database of RoadSegments for a particular street.

- DefaultRouteStep

Aggregates a number of RoadSegments of the same name together into a single path. This is typically used after

the result from Dijkstra is obtained and you then want to see the whole route without looking at individual small segments. Calculates the estimated time and total distance for the given step.

- Dijkstra

Implementation of Dijkstra’s algorithm for finding the shortest path between two points in a graph.

- GeoDatabase

Another Implementation of a Database which is used to add in shape points for each road from RT2.

- RouteFailureException

Signals when a route cannot be found between the two specified points in the graph.

- RoutePlanner

Primary class for calculating the fastest route between two points. This class is the one that initializes the node pool for Dijkstra to use, calls Dijkstra, and aggregates the output into a route. This is the file you’d want to change if you want to alter the speed limits for particular roads.

- RouteStep

Interface which defines the methods for a RouteStep and also implements the turn penalty and rules for turning within the graph. If you want to alter the turn penalty, you’ll have to do it here.

Python Script:

ie.py – This is just an example of a file I used to gather data from WMATA’s online trip planner and then later compare against the results of OpenTripPlanner. You could write a similar script for other sites, however this particular script will only work with WMATA at the current time.

**Configuration Required for Main Classes:**

**Interface –** A configuration file is optional to run this program, however you will need to know the path for the input directory, which is the directory of the XML files to be appended. You’ll also need to specify an output directory, which can either be the same as the input directory, or different depending on where you want the data to end up. You’ll also need to specify a KML file which contains the coordinates for the centroids and the directory which has the TIGER/Line files. If you are using a configuration file, make sure that it is in the same folder as the src folder of the project so it becomes visible to the program.

**GeoEncoder –** Configuration file not necessary, though you will be prompted for the TIGER/Line directory as well as the path to the KML file with the centroids.

**RouteFind-** Configuration file is not necessary to run this program, though make sure you have already created Graph.obj and have a KML file with the coordinates of each centroid.

**DataManager –** Because this is a standalone program only meant to take a peek at the data outputted, it currently takes not configuration and if you want to see a different XML directory loaded, you’d have to manually change the file name as noted in the code.

**FileChecker –** Another standalone program which does not require a configuration file, though you would have to change the directory in the code to load the proper directory that you want.

**COMvEmployment –** This file requires the use of the configuration file, and loads whatever files are specified in the document.

**DataAppender –** Does not use the configuration file, though will prompt the user for a directory to load the XML data files, as well as an Excel file which contains the data to be added. Please note that the Excel file should be .xls and not .xlsx because the latter will often cause an error at runtime. Also note that the first column should be the TAZ number to be altered and the second column should be the value that will actually be added.

**GUIMain –** The use of the configuration file is optional, though it will require access to the XML data files, a

KML file containing the boundaries of each TAZ and a path for the logo to be displayed as a splash screen.

**KMLBuilder –** This file requires the use of the configuration file and whatever is specified as the current paths/directories will be used.

**PopulationHousing-** For this program to work, you need to alter the paths in the source code to the actual files that you are using for the conversion between TAZs and blocks and the actual block data to be aggregated.

**RegionAccessibilityCalc –** Strictly uses the configuration file, so make sure to specify the fields properly.

**StopChecker –** Standalone program which does not use the configuration file. Though in the source code you will need to specify the path to the KML file of the boundaries of each shape as well as the stops.txt file from a particular transit agencies GTFS file.

**GraphBuilderMain –** To build a graph, you must pass this program a path to a configuration file for what exactly to build the graph using. Using the sample provided, you can alter the file to include your own particular GTFS files and OSM data.

**VizGui-** Requires you to specify the location of the graph object as an argument passed to the program.

**Common Questions**

**When I attempt to run one of the programs, it runs for a while, and then I get an error that it has run out of heap space?**

This error means that you need to allocate more memory to the Java Virtual Machine, so you’ll have to alter the Run Configurations of the program and set the value to something higher, such as “-Xmx1300” which will allocate 1300 bytes of RAM to the program.

**In the GUI visualization tool, all of the TAZs are black?**

This means that somewhere along the lines the name of the TAZ is getting lost in translation. You’ll want to make sure you are using the proper inputs and if so, add some output statements to make sure that they are hashing to the same values.

**I clicked on the screen and the TAZ went black?**

This means that you probably right clicked on the zone, and under the current implementation, when a zone is right clicked it is extruded from accessibility calculations and the map is re-colored accordingly.

**A pop-up window told me that there was a file with the configuration file?**

If this is at the start of the execution of the program, you may want to double check that the files you are using are correct and that all necessary opening and closing brackets were used in the XML file.

**How do I change the speed limits for each road?**

As noted you can make this change in RoutePlanner.java which contains the implementation for speed-limits.

**How do I alter the turn-based penalties?**

You can access the method which determines the penalties in RouteStep.java.

**I’m getting an error during execution of either of the route calculators?**

Chances are the point you are currently using is not close enough to a road or transit stop and if you move it slightly towards the nearest road, will more than likely solve your issue.