Title: **LAB III.**

Notice: Dr. Bryan Runck
Author: Michael Felzan
Date: April 1, 2021
Project Repository**:** ***https://github.com/fezfelzan/GIS5572.git***

Abstract

This report outlines three different methodologies for solving a vehicle routing problem, specifically involved with generating sets routes which multiple delivery drivers could take to minimize their cumulative time on the road, delivering goods to ten different stops. The first of these methodologies (*Section I.)* describes how to set up a custom network dataset in a Python Jupyter Notebook (using Esri's 'arcpy' package), and the process of inputting that dataset into the Arcpy network analyst "Solve Vehicle Routing Problem" function. The second extract-load-transfer (ETL) process described here uses that same Arcpy "Solve Vehicle Routing Problem" function, but instead uses Esri's proprietary network dataset as the input for the function. The final process outlined in this report demonstrates how the same type of vehicle routing problem may be approached using Esri ArcOnline Notebooks, using the "Plan Routes" function. The parameter inputs of these functions are explained in detail, and include establishing route boundaries and arrival time windows for delivery stops. The results (and validity thereof) of these ETLs are presented and discussed at the end of this document.

Problem Statement

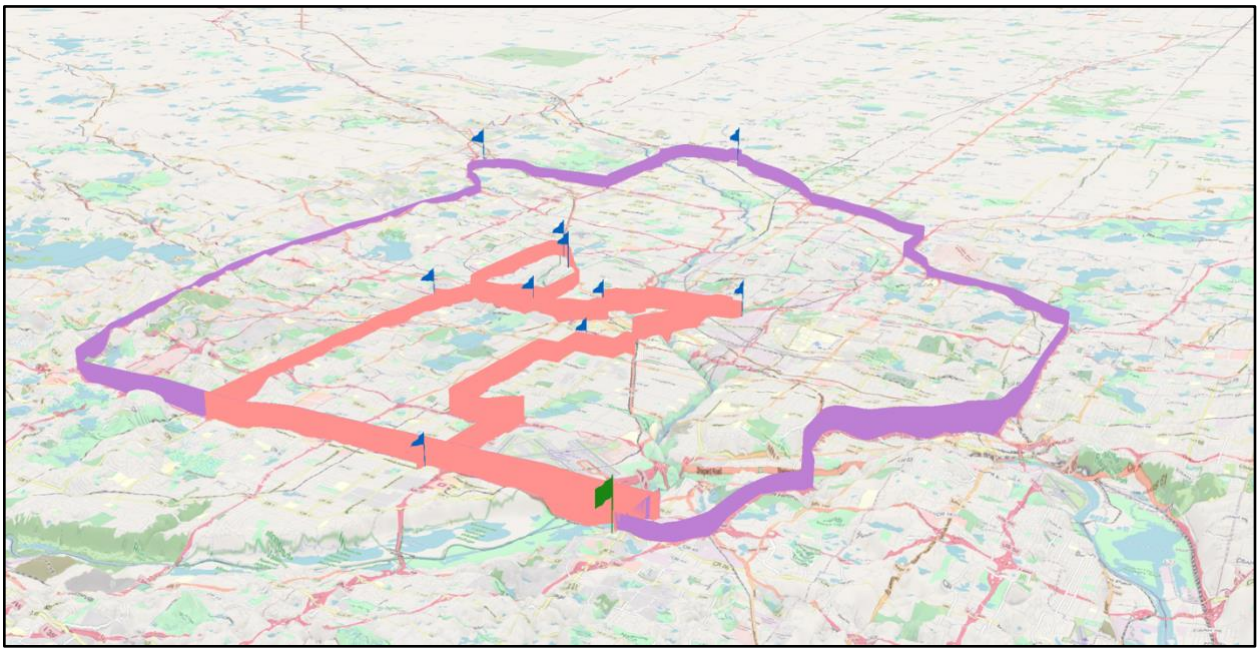


Figure 1: *Illustrative figure of the Lab 3 objective – designing the two optimal (least time consuming) routes two USPS drivers can take to deliver packages to ten places in the Metro Area, and return to their depot location.*

The objective of Lab 3 is to create an ETL which downloads coordinate locations source network data, creates a network dataset (or alternatively, utilizes Esri's proprietary network dataset), and generates the two optimal routes to ten different locations which have the lowest cumulative time-cost compared to any other route permutations. The routes must be generated in using both ArcPro and ArcOnline functionality. In addition to finding the optimal routes to the delivery points, restrictions must be placed on the drivers, so that they:

- - will not cross over certain highways (94 and 35W)
- - will arrive at two locations during a specific time window

The route directions must be printed in standard language form.

Table 1. Defining the Problem

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	Road network	Raw input dataset from MNDOT (county and local roads)	Road geometry	Speed limit, road name	Mn GeoSpatial Commons	Delete 94 and 35W; clip to reasonable extent
2	Stops	Coordinates of each location where the drivers must deliver packages to, made up of latitude and longitude values	Point Geometry		(Generated from routing text addresses into Google Places API)	XY Point to table; projecting to NAD83 UTM Zone15
3	Depot	Coordinates of the location where the drivers must start and end their journey at. Composed of latitude and longitude values	Point Geometry		(Generated from routing text addresses into Google Places API)	XY Point to table; projecting to NAD83 UTM Zone15
4	Network Dataset	Dataset made up of source network data, which contains the rules for the network (eg. costs of travelling on certain areas)	Topology (?)	Costs, Travel Modes, crs	Made up of MN Metro Area centerlines data	Change evaluators so that travel time cost considers speed limit

Input Data

Table 2. Input Data

#	Title	Purpose in Analysis	Link to Source
1	Minnesota Road Centerlines	Determining where drivers may travel to get to delivery locations	Mn GeoSpatial Commons
2	MN Counties Shapefile	Used to clip centerline data to be less large of a file	Mn GeoSpatial Commons
3	Coordinate data of delivery locations	Determine where the drivers need to go on their routes	Google Places API

Methods

Table of Contents:

Section I: ETL for Optimal Routing Problem using ArcPro/Arcpy

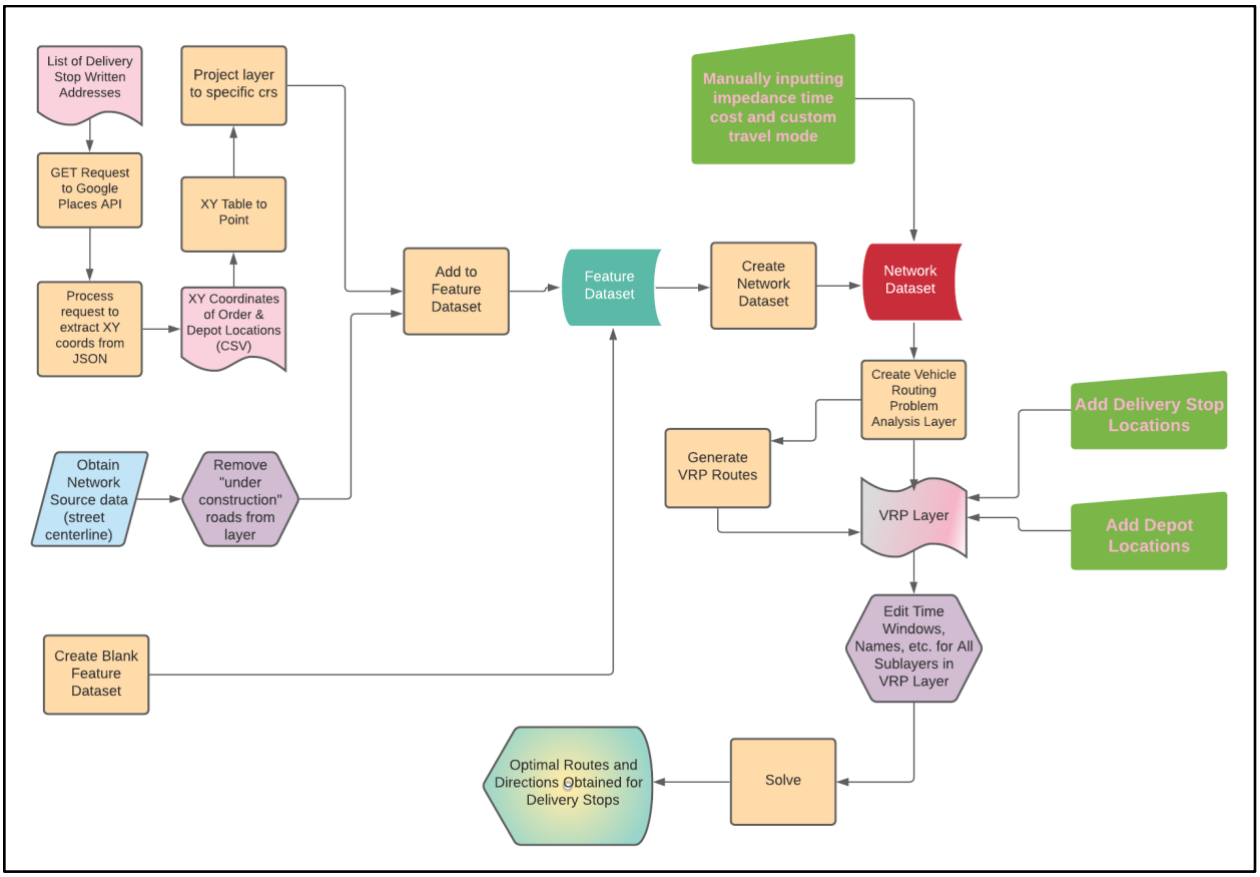


Figure 2. *Flow Diagram depicting the ETL process of retrieving lat/long coordinates for a list of written addresses via Google Places API, converting the coordinates to a point-geometry feature class, creating a network dataset, and setting up a vehicle routing problem layer to solve for the lowest-cost routes.*

1. **Obtaining a list of lat/long coordinates from a list of delivery stop addresses using Google Places API**

The first step in this process is to retrieve lat/long data for list of address points that are on the delivery route. Because we have been provided with written addresses, we may send each of these addresses to the Google Places API, to receive back location geometry data.

```
addresses = [
    "5525 Cedar Lake Rd S St Louis Park MN 55416",
    "225 Thomas Ave N 700 Minneapolis MN 55405",
    "701 N 5th St Minneapolis MN 55401",
    "920 E Lake St 123 Minneapolis MN 55407",
    "783 Harding St NE Minneapolis MN 55413",
    "4165 W Broadway Ave Robbinsdale MN 55422",
    "1321 E 78th St Bloomington MN 55425",
    "12547 Riverdale Blvd Coon Rapids MN 55448",
    "9875 Hospital Dr Maple Grove MN 55369",
    "3300 Oakdale Ave N Robbinsdale MN 55422"
]

startendloc = "1436 Lone Oak Rd St Paul MN 55121"
```

We may first input all of the written addresses into a Python list, as shown in **Figure 3.** We also must retrieve lat/long data for our start/end location (depot), so we will set up a Python variable for the depot as well.

Once we have a list of written addresses, we may iterate over this list and send GET requests to the Google Places API for each of these addresses to obtain their lat/long coordinates.

Figure 3. *Assigning delivery stops & depot location to a Python list*

As the Google Places API requires a key, we will first assign this to a variable. To perform this task in one fell swoop (for loop), I constructed *def* functions which send the addresses to the Google API, parse the returned JSON code, and append each addresses lat/long pair to a list.

```
my_API_key = "AIzaSyDFqX3I0LQmRZ23WJwBnnh7Y5VZZpwYt7o"

goog_places_URL_base = "https://maps.googleapis.com/maps/api/place/findplacefromtext/json"
```

Figure 4. *Assigning personal Google Places API key, URL base for API request to variables.*

Defining function which formats & sends a text address to the Google Places API, to receive back JSON text

```
def GooglePlacesRequester(address):

    formatted_input = address.replace(" ", "%20")

    input_param = "?input=" + formatted_input
    input_type = "&inputtype=textquery"
    fields_param = "&fields=formatted_address,name,geometry"
    key_param = "&key=" + my_API_key

    final_goog_path = goog_places_URL_base+input_param+input_type+fields_param+key_param

    goog_req_obj = requests.get(final_goog_path)
    googtext = goog_req_obj.text

    split_by_newlines = googtext.split("\n")

    return split_by_newlines
```

Figure 5. *Creating function which sends a written address location to the Google Places API, and returns the resulting JSON text object, split by newlines.*

The function in **Figure 5.** shows how each GET request was constructed and formatted to be registered by the Google Places API. The GooglePlacesRequester() function takes one input (a written address), and returns the JSON object that is returned from the request. However, before this object is returned, the function in **Figure 5.** splits the contents by 'newlines' (\n).

Defining a function which extracts the Lat/Long values from the JSON text (received by the post request), returns lat/long pair list

```
def LatLongExtractor(splt_newlines):

    rawlatlng = []
    latcounter = 0
    longcounter = 0

    for item in splt_newlines:
        if "lat" in item:
            if latcounter == 0:
                rawlatlng.append(item)
                latcounter += 1
        if "lng" in item:
            if longcounter == 0:
                rawlatlng.append(item)
                longcounter +=1

    firstsplit = rawlatlng[0].split(': ')[1]
    isolated_lat = firstsplit.split(',')[0]

    isolated_long = rawlatlng[1].split(': ')[1]

    return [isolated_lat, isolated_long]
```

Figure 6. *Creating def function which parses a "split-by-newline" text JSON object for the relevant lat/long coordinates, and returns them in as a pair (list).*

The above function LatLongExtractor() in **Figure 6.** parses the split-by-newline JSON text object to extract the relevant lat/long pair. It takes one argument, which is the split-by-newline JSON text object. The function returns a list, containing the latitude in the first position and the longitude in the second.

Now that our functions are constructed, we may iterate over the addresses in our address list and route each address into the chain of functions, appending to a list (latlong_pairs) each lat/long pair at the end of each for-loop iteration. **Figure 7.** shows the result of this function chain.

Iterating over every stop address, routing each address into GooglePlacesRequester() function, then routing JSON text into LatLongExtractor(), and finally appending "latlong_pairs" list with every addresses lat/long pair

```
for addr in addresses:

    iter_splitbynewlines = GooglePlacesRequester(addr)

    latlngpair = LatLongExtractor(iter_splitbynewlines)

    latlong_pairs.append(latlngpair)
```

```
latlong_pairs
```

```
8]: [['44.9616245', '-93.3502154'],
     ['44.9786616', '-93.312241'],
     ['44.98564', '-93.2814958'],
     ['44.9485075', '-93.26073889999999'],
     ['44.9983662', '-93.22108'],
     ['45.0309112', '-93.3392212'],
     ['44.8610404', '-93.25545529999999'],
     ['45.1985688', '-93.34779549999999'],
     ['45.132742', '-93.48116039999999'],
     ['45.0145559', '-93.323251']]
```

Figure 7. *Using def functions created in Figure 5. and Figure 6. to iterate over list of written addresses, route each address to the Google Places API, and retrieve back a list of lat/long coordinates for each delivery stop.*

2. Exporting list of lat/long pairs into csv/ESRI dbf table, to be converted into point feature classes

Having obtained a list of lat/long pairs for the delivery stops, we may now export this list to a .csv file. The process is as simple as shown in **Figure 8,** where a blank csv is created, its header is established, and each lat/long pair is written as a row.

Writing lat/long coord lists to csv files:

```
with open("addy_points2.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(['lat', 'long'])
    writer.writerows(latlong_pairs)
```

```
with open("start_end_pt.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(['lat', 'long'])
    writer.writerows(startend)
```

Figure 8. *Writing list of lat/long coords created in Figure 7. to .csv file*

Converting csv files to ESRI dbf tables; depositing tables in working file geodatabase

```
arcpy.env.workspace = r"C:\Users\michaelfelzan\Documents\arc ii Lab 03"

default_gdb = os.path.join(arcpy.env.workspace, 'lab_iii_scratch.gdb')
#default_gdb
```

```
inTable = "addy_points2.csv"
outLocation = default_gdb
outTable = "STOPS"

arcpy.TableToTable_conversion(inTable,
                              outLocation,
                              outTable)
```

8]:

Output

C:\Users\michaelfelzan\Documents\arc ii Lab 03\lab_iii_scratch.gdb\STOPS

Because of ESRI's read/write protect on .csv's, we will convert our csv files into ESRI dbf tables using the TableToTable_Conversion(), and output these tables in our scratch file geodatabase (**Figure 9**).

Figure 9. *Converting .csv file to .dbf table using TabletoTable()*

We now must turn our table of XY coordinates into a point feature class. We can accomplish through the arcpy.management.XYTableToPoint() function, taking care to assign the "long" and the "lat" to the correct parameters (**Figure 10**).

Converting tables to XY point feature classes, based on their Long/Lat values.

```
arcpy.management.XYTableToPoint('STOPS', # in table
                                "STOP_POINTS", # out fc
                                "long",  # x field
                                "lat",   # y field
                               )
```

2]:

Output

C:\Users\michaelfelzan\Documents\arc ii Lab 03\lab_iii_scratch.gdb\STOP_POINTS

Figure 10. *Converting table of lat long coords to point feature class using XYTableToPoint()*

3. Projecting the point feature classes into NAD83 UTM Zone 15N

We now have separate point feature classes for both our address stops and depot location (*the XY Point to Feature Class step was not shown above, but it follows the same process as Figure 10).* However, we must ensure that our point feature classes have a projected coordinate system that is consistent with our other data. Because we will be downloading a couple files from the MN DNR FTP server, which are often in NAD83 UTM Zone15N, we will project our point feature classes into this coordinate system.

```
sr = arcpy.SpatialReference(26915)
```

```
sr
```

]:

type	Projected
name	NAD_1983_UTM_Zone_15N
factoryCode	26915
linearUnitName	Meter
GCS.name	GCS_North_American_1983

Projecting point feature classes to NAD83 UTM Zone 15

```
projection_input = "STOP_POINTS"
projection_output = "Projected_Stops"
out_crs = sr

arcpy.Project_management(projection_input,
                         projection_output,
                         out_crs)
```

5]:

Output

C:\Users\michaelfelzan\Documents\arc ii Lab 03\lab_iii_scratch.gdb\Projected_Stops

Figure 11. *Using coordinate system's ESPG spatial reference number as the crs parameter input for the Project tool*

The above **Figure 11.** shows the process of entering in a coordinate system's ESPG spatial reference number into the Project tool to output a feature class w/ that spatial reference.

4. Downloading street centerline data for the MN Metro Area using the MN DNR FTP server

If we were to use Esri's network dataset to construct our optimal route paths, we would not have to construct a custom network dataset. However, because routing via Esri's network dataset costs Esri tokens (money), we will construct our own network dataset in ArcPro, and use it to create the two least-cost driving routes to our delivery stops. Afterwards, we will compare our routes to ones that are generated using Esri's network dataset.

In order to construct a network dataset from scratch, we need network data. Because our network will be a driving route, we may download the MN Metro Area Street Centerlines data from the MN DNR FTP server to use as our source data. **Figure 12.** shows the process of sending a POST request to the DNR API to return/extract a zipped shapefile.

Downloading MN Metro Area Street Centerlines shapefile from MN DNR FTP server:

```
gis_mngov_home = "https://resources.gisdata.mn.gov/pub/gdrs/data/"
roadcenterline_path = "pub/us_mn_state_metrogis/trans_road_centerlines_gac/shp_trans_road_centerlines_gac.zip"
final_roadpath = gis_mngov_home + roadcenterline_path

road_output_path = r"C:\Users\michaelfelzan\Documents\arc ii Lab 03"

roadreq_obj = requests.post(final_roadpath)

zippy = zipfile.ZipFile(io.BytesIO(roadreq_obj.content))
zippy.extractall(road_output_path)
```

Figure 12. *Sending a POST request to the MN DNR API to return/extract a zipped street centerline shapefile*

5. Downloading MN county shapefile (to be used to crop street centerline data) from the MN DNR FTP server

Because the Metro Area street centerlines data is a very large file, we may also download a shapefile of the MN county boundaries, to be used as a tool for clipping our data. There will likely be counties that the delivery drivers will almost certainly not have to go through (determined by eyeballing the map), so we can delete those parts of the street centerline data.

Downloading MN County Boundaries shapefile from MN DNR FTP server:

```
county_bound_path = "pub/us_mn_state_dnr/bdry_counties_in_minnesota/shp_bdry_counties_in_minnesota.zip"
final_cnty_bdr_path = gis_mngov_home + county_bound_path

cnty_bnd_reqobj = requests.post(final_cnty_bdr_path)

zippy2 = zipfile.ZipFile(io.BytesIO(cnty_bnd_reqobj.content))
zippy2.extractall(road_output_path)
```

Figure 13. *Sending a POST request to the MN DNR API to return/extract a zipped shapefile of MN county boundaries*

6. Removing highways 94 and 35W from the street centerline layer, to prevent routing on those roads.

In our specific routing problem, Randy and Reilly may not drive their USPS trucks on highway 94 or 35W, as they are under construction (classic). So, we must establish these roads as barriers in our network dataset so they may not cross over them. There are a few ways to perform this task, but I found the most success by just *removing the roads from the street centerline dataset before the network dataset is constructed*. **Figure 14.** demonstrates how a new feature may be created, using a combination of SelectLayerByAttribute() and CopyFeatures(). We may select only the roads with name '94' and '35W', and then invert the selection**,** to select all roads *except* 94 and 35W. From that selection, we may then create a new feature.

Selecting By Attributes all of the streets with name '94' or '35W', and then INVERTING THE SELECTION -- creating a new street centerline layer which excludes 94 and 35W:

```
roads_besides_badones = arcpy.management.SelectLayerByAttribute('RoadCenterline.shp',
                                                                 "NEW_SELECTION",
                                                                 "ST_NAME = '94' Or ST_NAME = '35W'",
                                                                 "INVERT")
arcpy.CopyFeatures_management(roads_besides_badones,
                              'centerlines_without_closed_rds.shp')
```

6]:

Output

C:\Users\michaelfelzan\Documents\arc ii Lab 03\centerlines_without_closed_rds.shp

Figure 14. *Using Select By Attributes to create new feature from Metro Area street centerline layer which includes every road in the original file except interstates 94 and 35W*

7. Adding data (all projected into NAD83 UTM Zone 15N) into one Feature Dataset

In order to create a network dataset, one must first have all of their source data in a Feature Dataset. **Figure 15.** shows the process of creating a new (blank) feature dataset, defined by a specific coordinate system (NAD83 UTM Zone15N), and exporting feature classes into the network dataset using the FeatureClassToFeatureClass() tool. We are able to do this only because we previously projected all of our source data into the network dataset's coordinate system

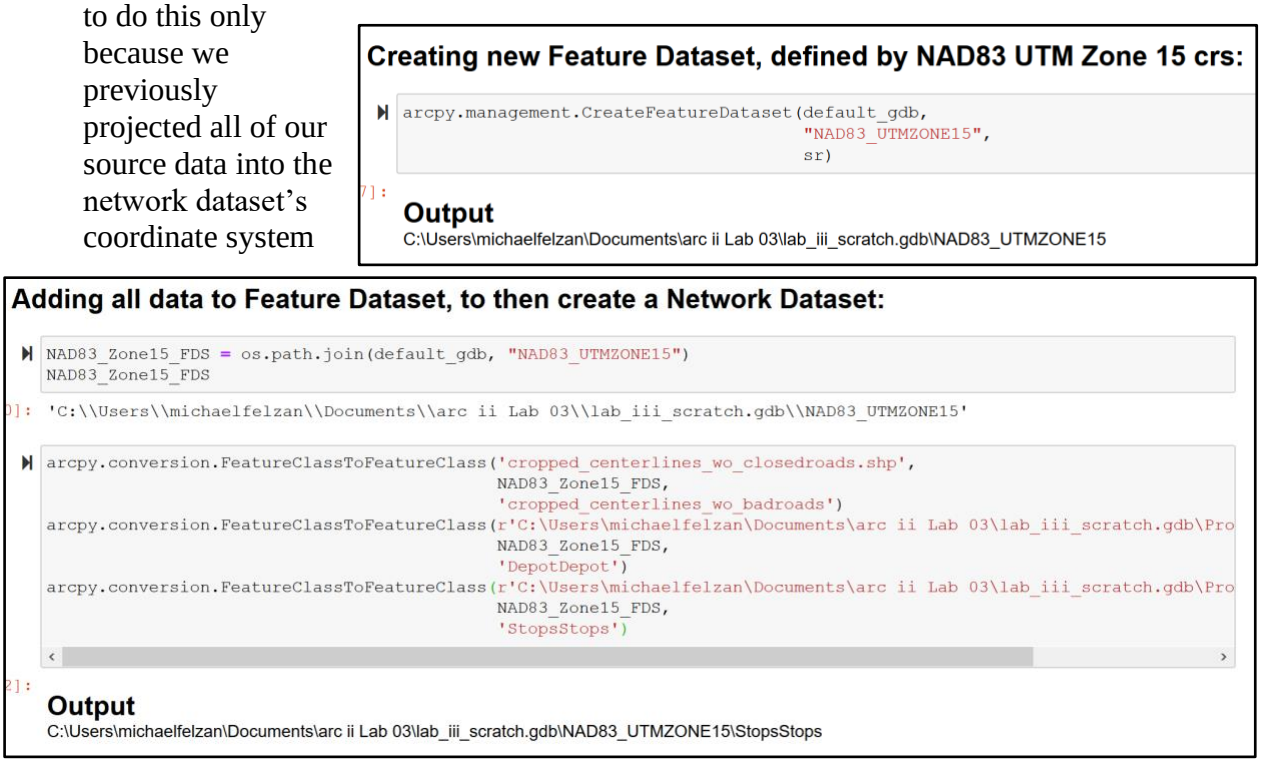


Figure 15. *Creating new feature dataset; adding projected data into feature dataset (in preparation of constructing a network dataset)*

8. Creating a Network Dataset, manually inputting impedance time cost and custom travel mode, building network

Now that all of our source data is in a single Feature Dataset, we can now create a Network Dataset. **Figure 16.** shows how this may be accomplished in arcpy using the na.CreateNetworkDataset() tool. The first argument this tool takes is the feature dataset, followed by the output name for the network, followed by the source data (in our case, street centerlines).

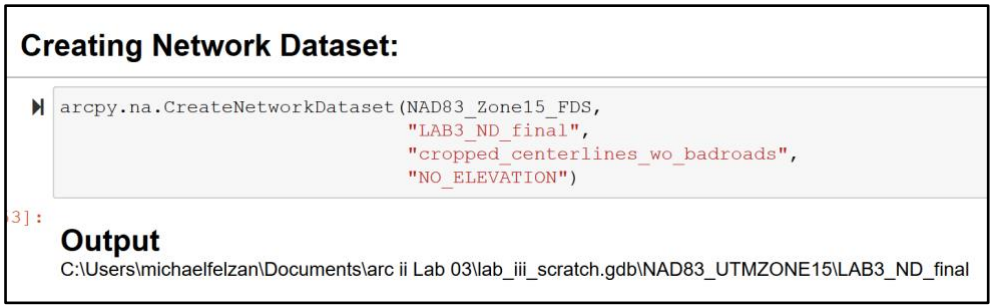


Figure 16. *Creating a network dataset using arcpy*

*** I spent a good amount of time searching the documentation for ways to create a custom **Cost** in a network dataset's attributes, and decided to "' throw in the towel ".. I think it is probably possible, thru modifying the properties of the VehicleRoutingProblemSolverProperties object…*
…..The following section uses the ArcPro GUI to demonstrate the process on constructing an "Impedance Time" cost and a custom Travel Mode in a network dataset.
***

To create a custom "Cost" in our network dataset in the ArcPro GUI, we may right click the network dataset (when it is not present in the map contents), and click Properties → Travel Attributes. Clicking the " ☰ ", we may add a new Time Cost, shown in **Figure 17,** as "ImpedenceTime_". We may also configure this cost's properties, such as the units it uses (we'll use seconds) and its data type. Additionally, we may create an evaluator for the road network which factors in the speed limit of each road (shown as the highlighted **[Shape_Length]/([SPEED_IMP]*0.44704))**
Shape_Length is the length in meters of each road segment, and SPEED_IMP is the time impedance due to the speed limit of each road segment. We multiply by 0.44704 because 1 mph = 0.44704 m/**s**, and NAD83 UTM Zone15N uses meters.

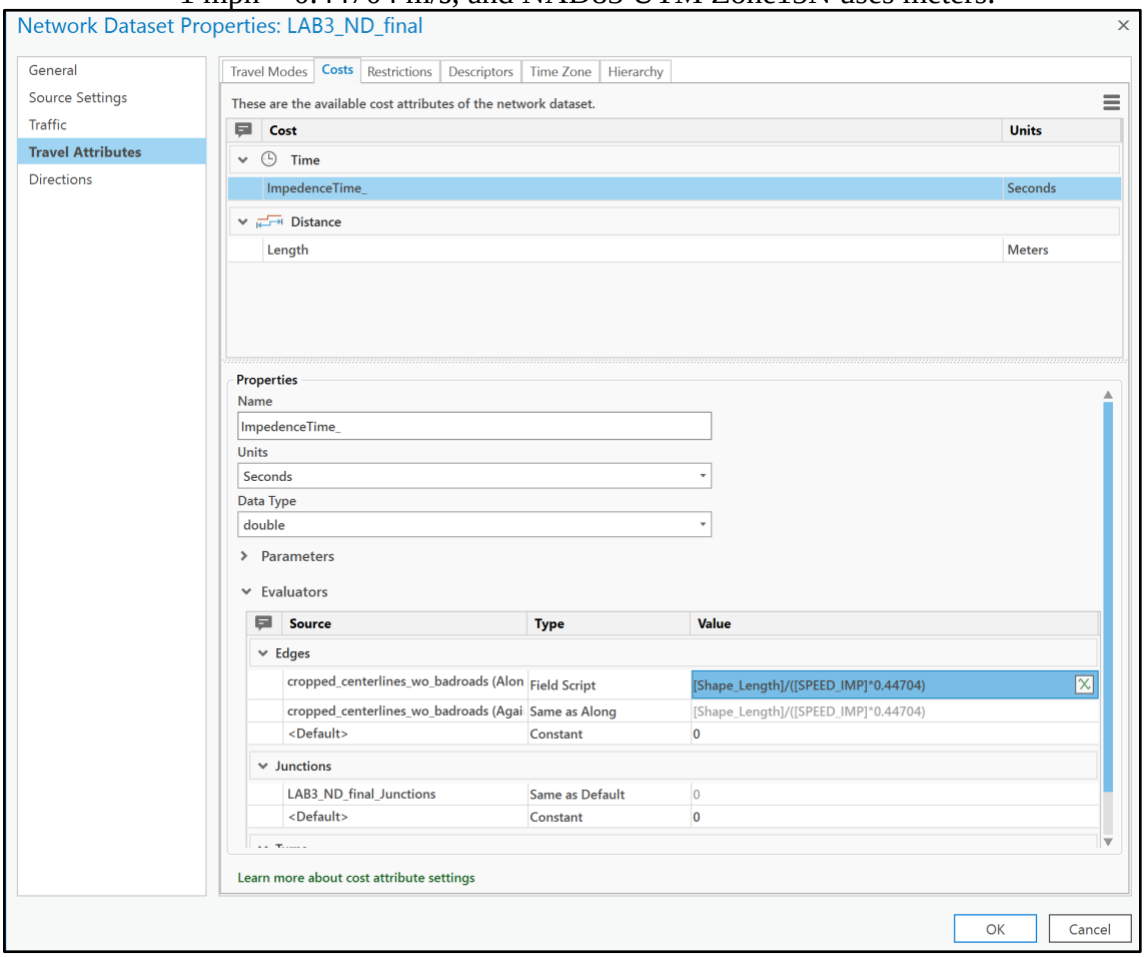


Figure 17. *Creating a custom impedance time "cost" via the Network Dataset Properties window.*

Now that we have configured our custom ImpedenceTime_ cost (realizing now I spelled impedance wrong), we can construct a custom Travel Mode for the network dataset. We may label it Drive_SOV, and set the Impedance to "ImpedenceTime". Turning on "Dead-Ends and Intersections" will allow the drivers to turn around if one of their stops falls on a cul-de-sac.

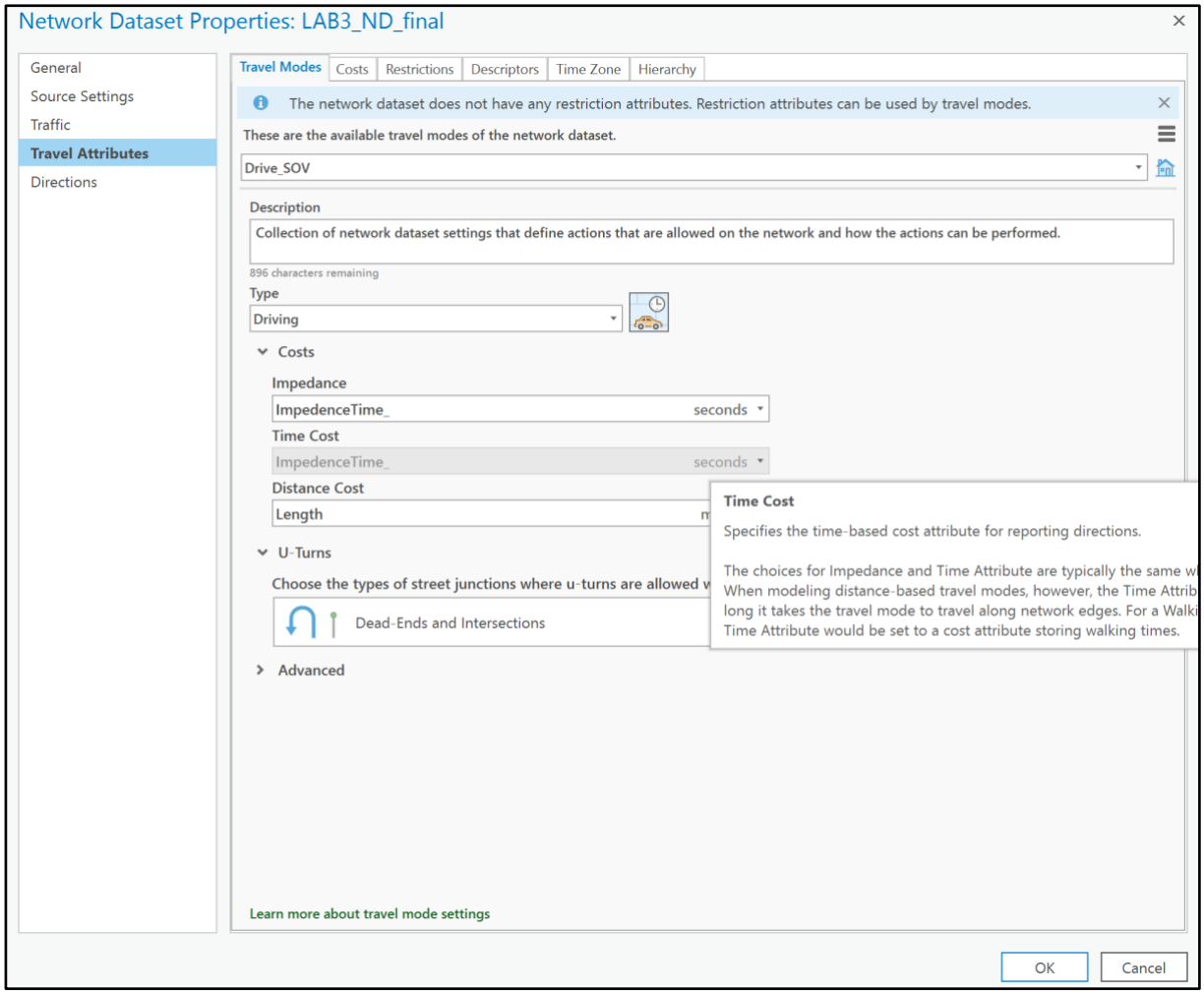


Figure 18. *Creating a custom "Travel Mode" for a network dataset via the Network Dataset Properties window.*

9. Creating a Vehicle Routing Problem Analysis Layer

With our network dataset properly set up, and after having built the network using BuildNetwork(), we may now make a Vehicle Routing Problem Analysis Layer with the arcpy tool of the same name. **Figure 19.** shows how this tool may be utilized: the first argument is the input network dataset, the second is the name for your routing problem layer, the third is the travel mode (which we assign to our custom "Drive_SOV"). We

also need to enter in the units for the calculation, the 'default date,' and importantly, if we want **directions** to be outputted when we hit Solve.

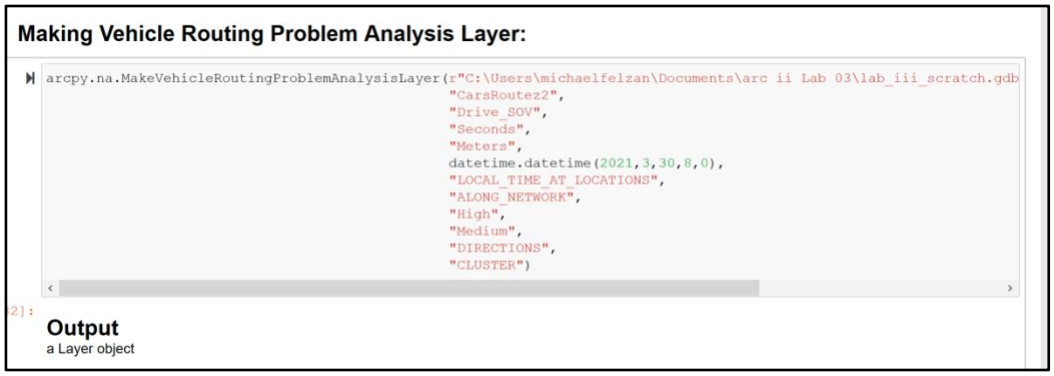
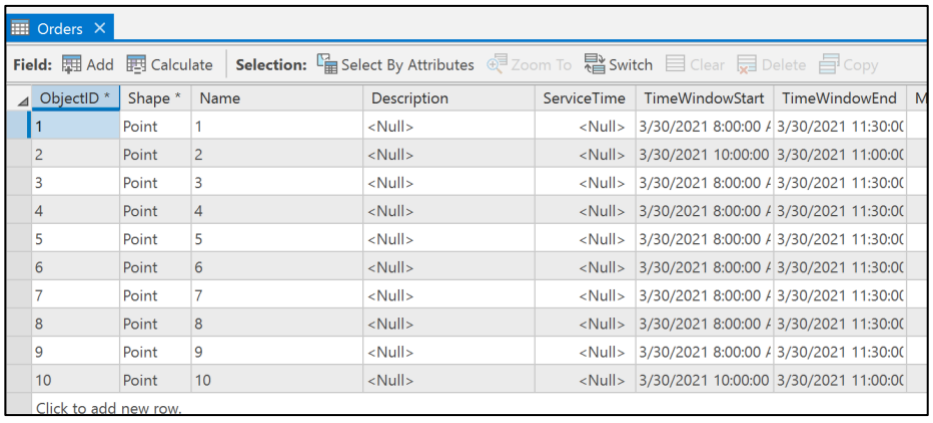


Figure 19. *Demonstration of how the arcpy Make Vehicle Routing Problem Analysis Layer function may be parameterized.*

10. Adding Delivery Stop and Depot Locations to the Vehicle Routing Problem Analysis Layer

Before we can run the Solve() function and receive the two quickest routes two drivers may take to deliver packages to 10 different stops, we must 'add' these locations to the Routing Problem layer. However, before we do this, we need to make some edits to our Stops, Depot, and Routes attribute tables.

***Note……I also could not figure out how to do this using Arcpy. I have a feeling I was on the right track (see ArcPro notebook in repo), but because the vehivle routing problem layer has multiple feature classes within it, the field mapping process seems much more involved than something that could be accomplished using SearchCursor. The following edits to the VRP layer will be demonstrated in the ArcPro GUI.*


Orders

Field: Add Calculate Selection: Select By Attributes Zoom To Switch Clear Delete Copy

ObjectID *	Shape *	Name	Description	ServiceTime	TimeWindowStart	TimeWindowEnd
1	Point	1	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
2	Point	2	<Null>	<Null>	3/30/2021 10:00:00	3/30/2021 11:00:0(
3	Point	3	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
4	Point	4	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
5	Point	5	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
6	Point	6	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
7	Point	7	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
8	Point	8	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
9	Point	9	<Null>	<Null>	3/30/2021 8:00:00 A	3/30/2021 11:30:0(
10	Point	10	<Null>	<Null>	3/30/2021 10:00:00	3/30/2021 11:00:0(

Click to add new row.

Figure 20. *Editing the attribute table for the sublayer "Orders" in the Vehicle Routing Problem (VRP) layer. Each order name must be non-null and a unique value, and the "time window" start and ends must be filled out (and be consistent with the other datetime info in the depots and routes tables)*

Figure 20. shows the attribute table for the "Orders" sublayer of the VRP. In order for the Solve() function to not error out, we must give each delivery stop a valid, unique name. Additionally, we must give each delivery stop a valid TimeWindowStart and TimeWindowEnd in 'datetime' format. I decided to use March 30, 2021 as the date Randy and Reilly hit the road – the date shouldn't matter as long as it is consistent across

the data. In my Stops point feature class, stops "2" and "10" happened to be

225 Thomas Ave N #700, Minneapolis, MN 55405

and

3300 Oakdale Ave N, Robbinsdale, MN 55422, respectively.

Because both of these addresses have a concrete time window the deliveries must be performed during (between 10:00 and 11:00 AM), these points' TimeWindow rows will be updated to reflect this restriction. For the other points' TimeWindows, I used a large window (8:00AM to 11:30PM), as there is no specified end-time cap …. though their boss probably doesn't want to pay them for more than 14 hours of overtime work.

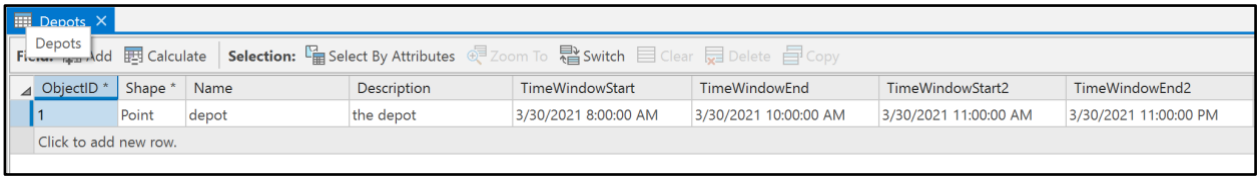


Figure 21. *Editing the VRP "Depots"sublayer, giving the table rows valid time window start/ends*

Figure 21. shows the attribute table for the 'Depots' sublayer in the VRP layer. Like the Orders table, the depot must have a valid name, and non-null TimeWindows. The first time window represents when the drivers should leave by, and the second is when they must return by. I figured that as long as the drivers got to the two locations that need to be hit exactly between 10-11 AM, it doesn't really matter if they get a late start. Also, if they finish early (before 11:00), they can either chill in USPS station parking lot and rack up some more overtime, or clock out early.

(We also will have to ensure our 'Routes' sublayer has valid table inputs, but because the process is identical to the Orders and Depots table manipulations, I will skip discussing this step)

11. Adding Stops and Depots locations to the Vehicle Routing Problem (VRP) Analysis Layer

Having now updated our VRP sublayer tables, we can now input our stops and start/end location to the layer. To do this, we can use the AddLocations() tool in Esri's Network Analysist toolbox. We need to run the tool two separate times – once to input the Orders, and again to input the Depots.

For the parameters of the AddLocations() function, we will input the VRP name, whatever sublayer we're inputting (orders or depots), and the name(s) of the network source data.

ADDING LOCATIONS TO VEHICLE ROUTING PROBLEM ANALYSIS LAYER -- Function must be run two times, to input both Stops & Depot

```
arcpy.na.AddLocations("Vehicle Routing Prob",
                      "Orders",
                      "StopsStops",
                      "Name Name #;Description # #;ServiceTime # #;TimeWindowStart # #;TimeWindowEnd # #;MaxViolatio
                      "5000 Meters",
                      None,
                      "cropped_centerlines_wo_badroads SHAPE;LAB3_ND_final_Junctions NONE",
                      "MATCH_TO_CLOSEST",
                      "APPEND",
                      "NO_SNAP",
                      "5 Meters",
                      "EXCLUDE",
                      "cropped_centerlines_wo_badroads #;LAB3_ND_final_Junctions #")
```

3]:

Output
a Layer object

```
arcpy.na.AddLocations("Vehicle Routing Prob",
                      "Depots",
                      "DepotDepot",
                      "Name Name #;Description # #;TimeWindowStart # #;TimeWindowEnd # #;TimeWindowStart2 # #;TimeWi
                      "5000 Meters",
                      None,
                      "cropped_centerlines_wo_badroads SHAPE;LAB3_ND_final_Junctions NONE",
                      "MATCH_TO_CLOSEST",
                      "APPEND",
                      "NO_SNAP",
                      "5 Meters",
                      "EXCLUDE",
                      "cropped_centerlines_wo_badroads #;LAB3_ND_final_Junctions #")
```

Output
a Layer object

Figure 22. *"Adding locations" to the VRP layer using the AddLocations() network analysist function. The function must be run two times – once to add the order locations, and again to add the depot locations. The routes will be added thru using the AddVehicleRoutingProblemRoutes() function.*

12. Adding Vehicle Routing Problem Routes, and Solving for the optimal routes

We may now finally set up the parameters to generate our optimal route paths, and run the function to calculate the routes. Shown in **Figure 23**, we can use the AddVehicleRoutingProblemRoutes() function to set up the framework for our routes. The second parameter of this function specifies how many routes you want to generate – in our case, we want to generate 2 – one for Randy, one for Reilly. After designating two routes to be created, we may also split the number of deliveries each driver must make in half, by entering in '5' for the max orders parameter (** *as I'm writing this now, I'm realizing that this maybe was not the correct way to go about this problem…as it might have been faster for, say, Randy to tackle 6 delivery spots and Reilly only 4)*. With these parameters entered in, we will now generate the two most optimal routes (routes between two drivers that will have the lowest net time) when each driver is assigned 5 stops.

Running na.Solve() on our Vehicle Routing Problem outputs the two optimal routes, along with the written English 'MapQuest' directions corresponding to each (because we toggled on "output directions during solve" in an earlier function).

Adding Vehicle Routing Problem Routes; running SOLVE

```
arcpy.na.AddVehicleRoutingProblemRoutes("Vehicle Routing Problem 2",
                                        2,
                                        "Route",
                                        "Placemark",
                                        "Placemark",
                                        "8:00:00 AM",
                                        "10:00:00 AM",
                                        5,
                                        None,
                                        None,
                                        "# 1 # # #",
                                        None,
                                        "APPEND")

arcpy.na.Solve("Vehicle Routing Problem 2",
               "HALT",
               "TERMINATE",
               None,
               '')
```

Figure 23. *Using the AddVehicleRoutingProblemRoutes() function to add the row "skeletons" to the VRP, which will be filled in by the true optimal routes after the "solve" function has been run*

~ see github repo for PDF directions for both routes.

Section II: Comparing Our Custom Network Dataset's Route Output to One Generated by Esri's Network Dataset

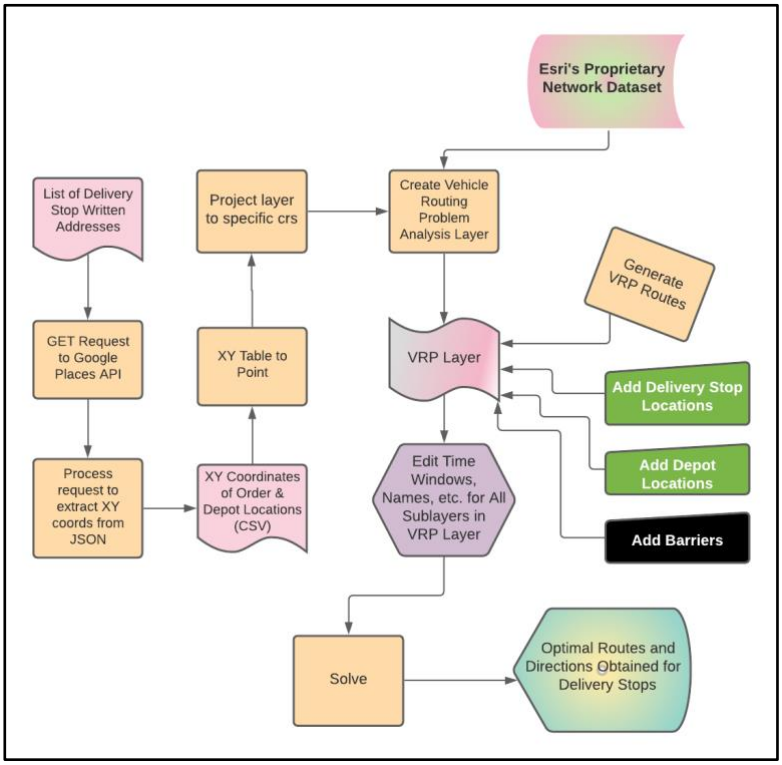


Figure 24. *Flow Diagram depicting the ETL process of setting up a vehicle routing problem layer to solve for the lowest-cost routes, using Esri's proprietary network dataset instead of creating a custom network dataset.*

To generate the two optimal delivery routes using Esri's proprietary network dataset, we can run through the same process of making a vehicle routing problem analysis layer outlined in step I.-9, but instead enter Esri's network dataset into the first parameter. To call this network dataset, one must enter in the string "https://www.arcgis.com/" . After a new VRP layer has been generated using this network dataset, steps 10 through 12 from section I. may (for the most part) be repeated.

Because we set up "barriers" on our custom network dataset by simply deleting highways 94 and 35W from the source road data, and we cannot access the source data for Esri's proprietary dataset, we have to set up line or polygon boundaries in the VRP layer. However, because I was unable to properly construct a barriers layer (see section III.-3 for pity-filled explanation), I did not use barriers for my final output. However, if one was to, say, create a feature class composed of heads-up digitized line barriers at all possible ramps where one could get onto 94 or 35W, this layer could be 'imported' to the Line Barriers subclass that is created whenever a Vehicle Routing Problem Analysis Layer is created.

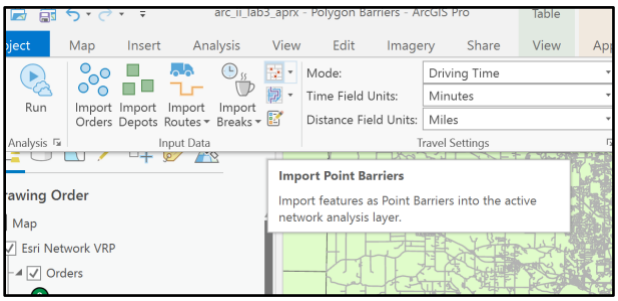


Figure 25. *Hovering over option to import/add barriers to VRP in ArcPro GUI*

I received fairly different results using Esri's network dataset than I did using my own.

***Note: it is hard to adequately compare the outputs, because I changed the parameters when creating the 'Esri Network VRP' to not restrict each driver's number of stops to 5…whereas my final output with the custom network dataset capped the max number of stops by each driver at 5.*

The most noticeable difference I saw was that, when there was no limit to the number of stops that each driver could make, *having one driver perform all of the deliveries himself took the least amount of (cumulative) time*. I printed the directions PDF from this output, saved as *"esri network arcpro one driver.pdf"* in the GitHub repository. I confirmed that one driver making all the deliveries would generate the least time, through adjusting the delivery stop caps on the Esri Network VRP route layers – the second driver only completes as many stops as he's 'allowed' to. If the max number of stops is set to 9, the other driver makes one stop; if the cap is set to 8, the other driver completes two stops.

Section III: ETL for Optimal Routing Problem using ArcOnline

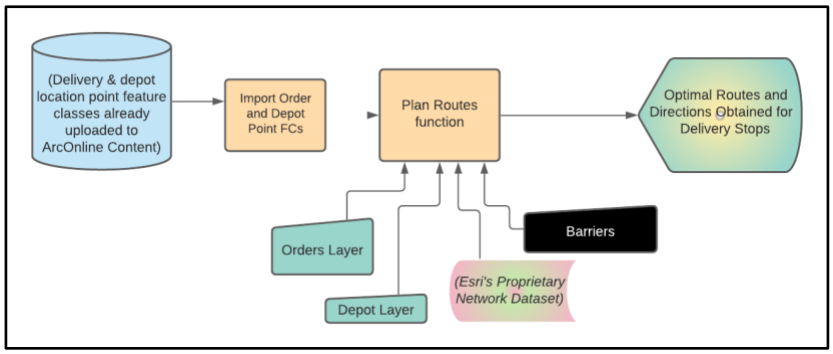


Figure 26. *Flow Diagram depicting the process of using the ArcOnline "Plan Routes" function to solve a vehicle routing problem*

Now that we have demonstrated how to solve a vehicle routing problem in ArcPro using arcpy.na.MakeVehicleRoutingProblemAnalysisLayer() , and explained how the outputs of this function may vary depending on the network dataset inputted, the barriers inputted, the number of stops specified, etc., we will demonstrate how a similar routing problem ETL may be achieved using the ArcOnline notebooks.

~ * ~ During the initial completion of this Lab Report, I was practically convinced that using the plan_routes() function was the only way to complete such a routing task thru ArcOnline. However, having discussed with a couple classmates, I now realize the MakeVehicleRoutingProblemAnalysisLayer() function exists in its own form for ArcOnline, with its own unique and special, slightly different-than-ArcPro ArcOnline syntax. However, I am still under the impression that, if one was truly going to use ArcOnline to perform this task, they would be doing so because they want a simpler, more user friendly workflow…and if they needed* ***accurate *results, they should best do the heavy lifting in ArcPro. So, I've kept the following section intact, which explains the workflow of using ArcOnline's plan_routes() function (which accesses Esri's proprietary network dataset) in order to solve a travelling salesman problem.*

1. **Importing data (delivery stops point FC; depot location point FC)**

   To demonstrate how ArcOnline may be used to solve the same network problem, I will skip the steps of downloading/prepping/projecting the data, as those steps should be similar or identical to coding them through ArcOnline.

   We may start by importing our data (point locations of delivery stops and start/end point), via the ArcOnline method of gis.content.get(<layer ID>). This requires the layers to have already been uploaded to ArcOnline's Content library (public or private).

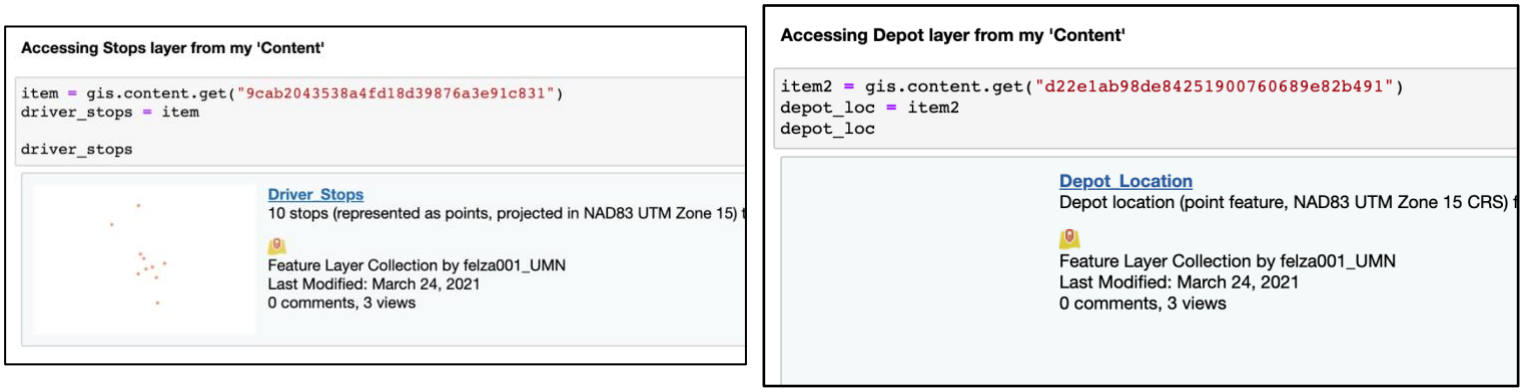


Figure 27. *Using Esri content URL "ids" to assign layers to variables in an ArcOnline notebook; assigning previously uploaded "orders" and "depots" layers to "driver_stops" and "depot_loc" respectively.*

2. **Using the Plan Routes function to create the optimal routes the two drivers can take, where they each are allocated 5 stops**

   Working in an ArcOnline notebook, we have the ability to use the plan_routes() feature. As stated in Esri's documentation for this feature,

   > *"Plan Routes routes many vehicles at once. Furthermore, it determines how the stops should be divvied up among the various routes and the best order in which the routes should visit the stops."* ("Plan Routes", Portal for ArcGIS, Esri)

   The Plan Routes feature is available to any Esri user who has "Network Analysis privileges." Because you are using Esri's proprietary network dataset when you run the tool, each run of the plan_routes() function will consume Esri credits.

   Esri claims that this tool is designed to provide a "simple solution to common uses of fleet routing," though provides a disclaimer that, if the user wants to do more complex routing calculations, like, for example, assigning Time Windows to certain delivery stops, the user should consider using ArcPro's Vehicle Routing Problem Service:

> Note:
> This task is designed to provide a simple solution to the most common uses of fleet routing, or assigning stops to routes, people, or vehicles and finding the best routes between the stops. If you require additional flexibility to solve a more specialized problem, consider using the Vehicle Routing Problem Service instead. It provides many more options, such as honoring time windows at stops; adding barriers to block portions of the road network; setting vehicle-specific limits on maximum travel time, maximum number of stops to visit, and so on.

https://developers.arcgis.com/rest/analysis/api-reference/plan-routes.htm

The "features.use_proximity.plan_routes()" function has a variety of parameters – many which are optional, and some which are self-referential (eg. if you enter in return_to_start as "True", you must set not have an end_layer parameter). **Figure 28.** illustrates how this function's params may be populated in order to generate two routes w/ the lowest cumulative time, given that the drivers make 5 deliveries each, and start their journey at 8:00AM. **Figure 29.** illustrates all of the possible parameters for the plan_routes() function.

Using the Plan Routes function to create the optimal routes the two drivers can take, where they each are allocated 5 stops:

```
route_v2 = features.use_proximity.plan_routes(stops_layer=driver_stops,
                                              route_count=2,
                                              max_stops_per_route=5,
                                              route_start_time=datetime.datetime(2021,3,30,8,0),
                                              start_layer=depot_loc,
                                              return_to_start=True,
                                              end_layer=None,
                                              end_layer_route_id_field=None,
                                              travel_mode='Driving Time',
                                              stop_service_time=5,
                                              include_route_layers=True,
                                              output_name='lab_three_route')
```

```
Network elements with avoid-restrictions are traversed in the output (restriction attribute names: "Through Traffic Prohibited").
```

```
lab3routeitem = gis.content.get("474350a2dcf44b7d8aa755b349133d6f")
lab3_route = lab3routeitem
lab3_route
```

lab_three_route

Feature Layer Collection by felza001_UMN
Last Modified: March 25, 2021
0 comments, 0 views

Figure 28. *Setting up the parameters of the Plan Routes function, calling outputted routes layer to variable*

3. (Setting up road barriers)

As stated above, **Figure 29.** below describes all of the parameter inputs for plan_routes(). Looking at this figure, we can see that this function *does* allow line or polygon barriers to be set up (so that the drivers may not take 94 or 35W on their routes). However, after logging ~3 hours (**Figure 30.**) trying to conjure a polygon barrier in ArcPro for the highways, which has pieces missing where streets cross over/under the highways, I decided to give up… as Esri stated, this tool is more targeted towards simple analysis.

```
Signature:
features.use_proximity.plan_routes(
    stops_layer,
    route_count,
    max_stops_per_route,
    route_start_time,
    start_layer,
    start_layer_route_id_field=None,
    return_to_start=True,
    end_layer=None,
    end_layer_route_id_field=None,
    travel_mode='Driving Time',
    stop_service_time=0,
    max_route_time=525600,
    include_route_layers=False,
    output_name=None,
    context=None,
    gis=None,
    estimate=False,
    point_barrier_layer=None,
    line_barrier_layer=None,
    polygon_barrier_layer=None,
    future=False,
)
Docstring:
.. image:: _static/images/plan_routes/plan_routes.png

.. |balanced| image:: _static/images/plan_routes/balanced.png
.. |partially_balanced| image:: _static/images/plan_routes/partially_balanced.png
.. |unbalanced| image:: _static/images/plan_routes/unbalanced.png

The ``plan_routes`` method determines how to efficiently divide tasks among a mobile workforce.
```

Figure 29. *Listing all possible parameters that may be inputted to the Plan Routes function in an ArcOnline cell output, by entering in a "?" at the end of the function and running the cell.*

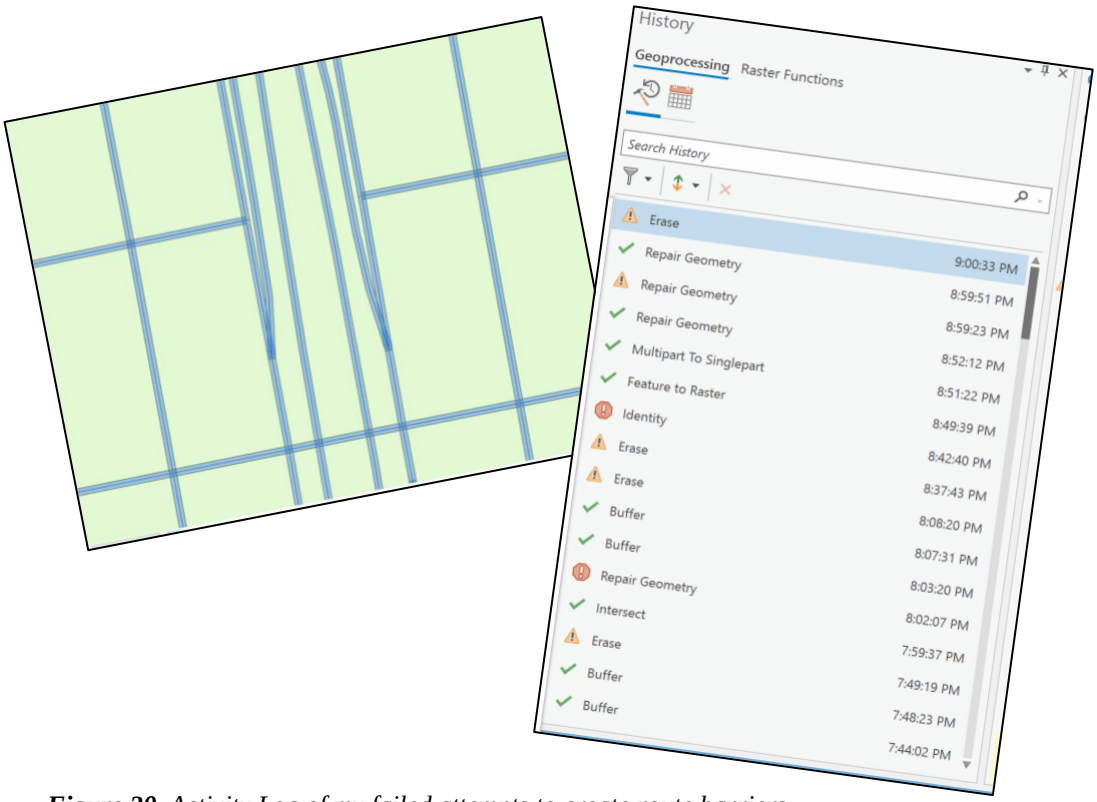


Figure 30. *Activity Log of my failed attempts to create route barriers*

4. Printing directions

After the plan_routes() function has been executed, our two optimal routes layer should appear in our ArcOnline "Content." Opening this layer up Map Viewer, we can view the

written directions for each separate route. I've printed both of the route's MapQuest-style directions as PDFs, and attached them to my GitHub repo.

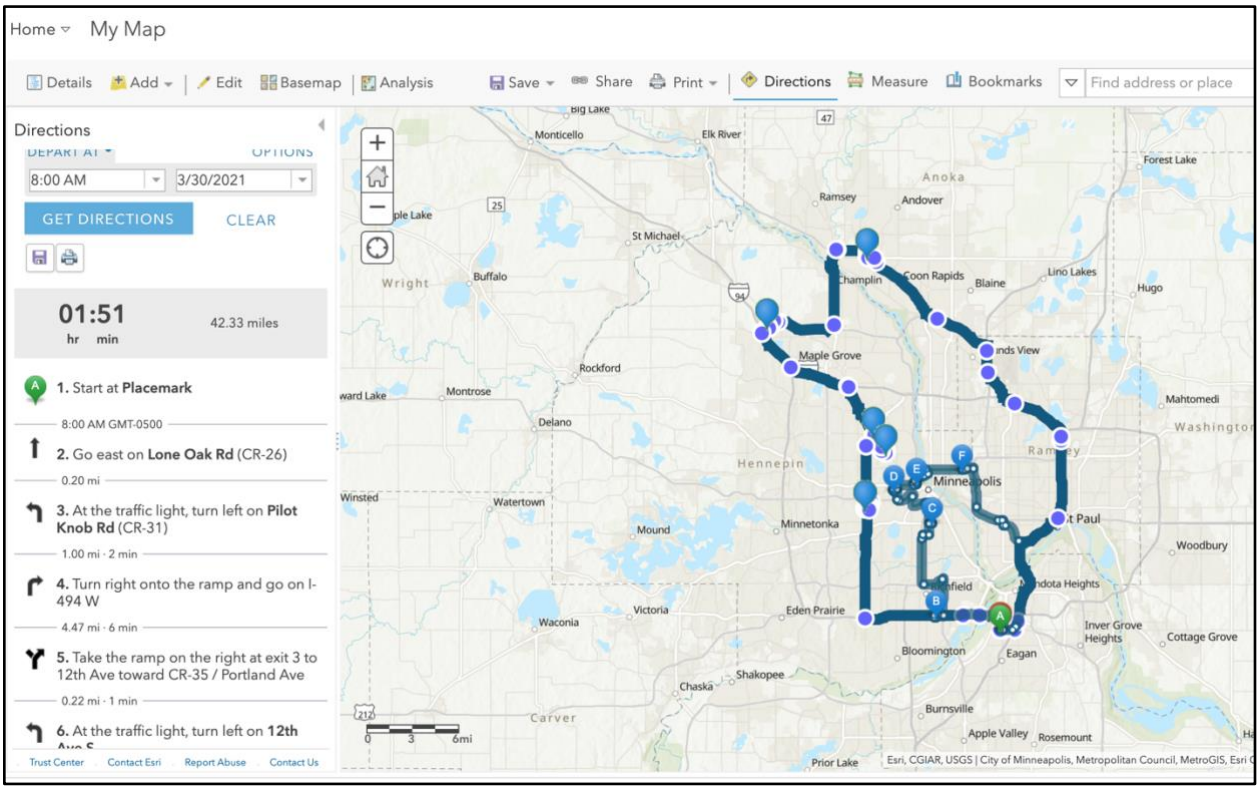
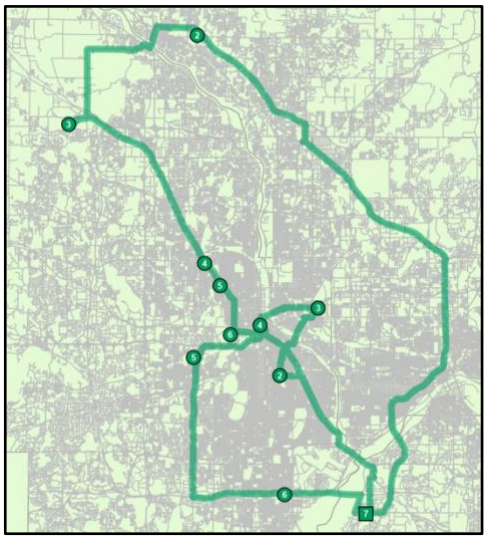
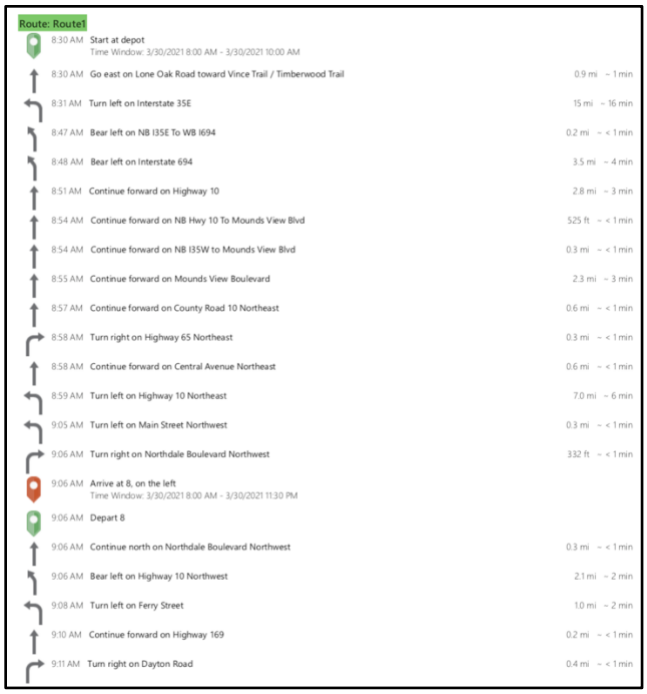


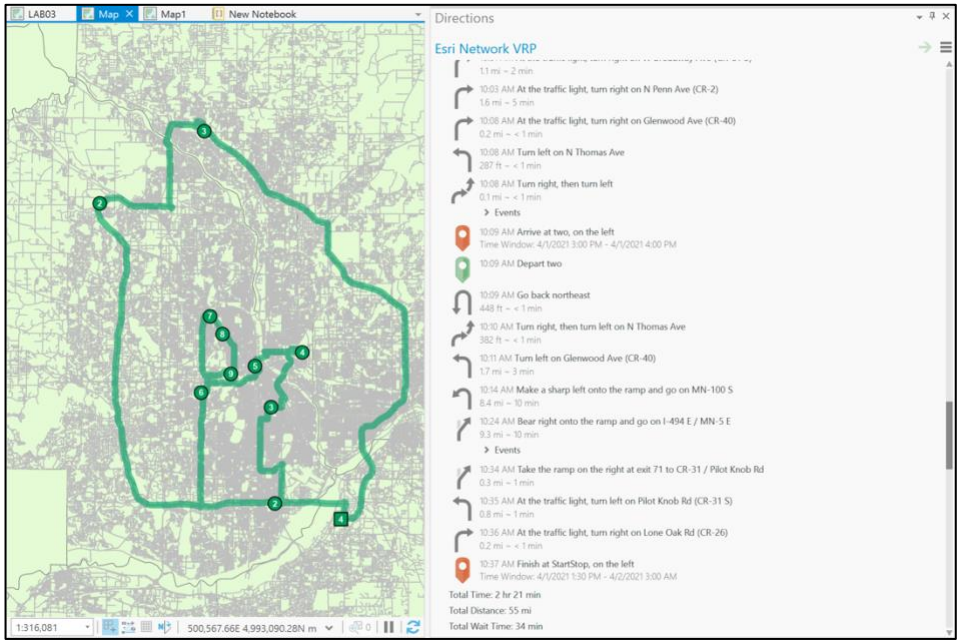
Figure 31. *Opening up the 'routes' layer generated by the Plan Routes function, viewing the written driving directions*

Results / Results Verification

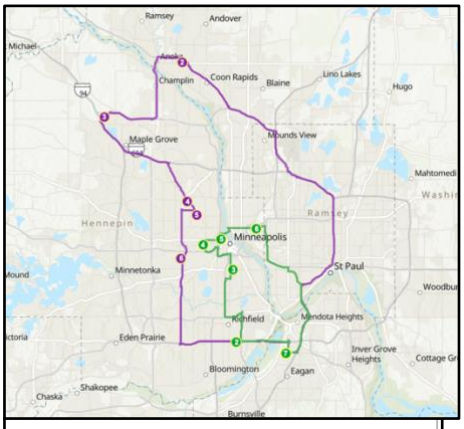
A.

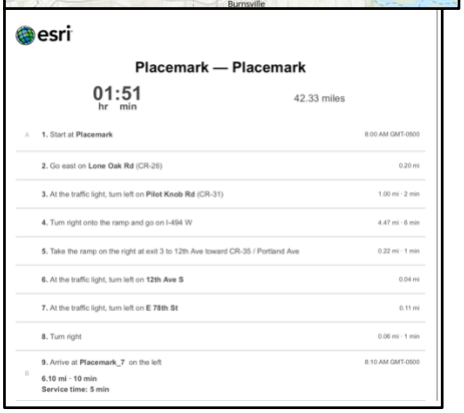






B.





C.

Figure 32. *A.: Output of optimal routes both drivers should take to reach all delivery spots and return in shortest time possible, generated in ArcPro using custom network dataset; B.: Output of vehicle routing problem analysis in ArcPro, using Esri's network dataset instead of custom ND; C.: Output of "Plan Routes" function executed in ArcOnline notebook/environment, & visualization in ArcOnline Map Viewer.*

Figure 32. depicts the final products of the routing ETLs – **A.** via ArcPro (arcpy) inputting a custom network dataset into the Solve Vehicle Routing Problem function, **B.** via ArcPro using Esri's proprietary network dataset and the Solve Vehicle Routing Problem function, and **C.** via the Plan Routes function in an ArcOnline notebook (also using Esri's network dataset). The ETL methodologies outlined in this document were able to successfully generate routes/directions which address this Lab's routing problem. The outputs describe the two routes the USPS drivers can take, from the start location, to the 10 delivery locations, and back home, which minimize their cumulative time on the road.

As confirmed by manually entering location addresses in Google Maps, the point locations of each stop/depot appear to be accurate. The road centerline data for the MN Metro area is assumed to be accurate, as it passed QC from the Minnesota Geospatial Advisory Council (GAC) Road Centerline Data Standard. For the first map generated in ArcPro, using a custom network dataset, the route considers the impedance time due to each road segment's speed limit, and factors that into the total time cost, although average traffic times for each roadway are not considered. Repeating this process in ArcPro with Esri's network dataset, it is hard to say how the dataset is set up on the back-end, though may be assumed that the dataset makes these same impedence considerations, with even more intricate specifications than the custom one created in Section I.

For the custom network generated in ArcPro, Highways 94 and 35W were completely removed from the source road data, ensuring the drivers would not take these roads to complete their deliveries (removing the highways leaves the roads that pass thru the highways intact, because the centerline dataset is made up of multipart line features). I believe the result in ArcPro should be close to accurate – however, I made the choice to divvy up the # of deliveries up between the two drivers evenly, so that each driver makes exactly five stops. In hindsight, I am not sure this restriction would necessarily produce the optimal combination of routes, as it may be faster for one driver to make more than 5 stops, and the other to make less than 5.

Using the Esri's network dataset with the Solve Vehicle Routing Problem function in ArcPro, the optimal "two" routes ended up being, one driver making all of the stops, and the other driver not working. This is quite possibly not an error, as having two drivers on the road could produce more *cumulative* time, considering traffic and the time it takes to go to and from the depot location.

For the route result produced in ArcOnline, using the Plan Routes function and Esri's own network dataset, almost all of the same parameters were used as those in the ArcPro Solve Vehicle Routing Problem function, though there are some differences that could make the ArcOnline routes less accurate than both of the ArcPro routes. One of the major concerns with the ArcOnline result I achieved is that it does not consider the restrictions of travelling on 94 and 35W (though this is not a fault of the function/medium – this was due to my inability to properly construct a barriers layer). Another is that it fails to consider the time 10-11AM time window for those two stops – however, Esri states in one of the documentation pages for the Plan Routes function that the tool does not support that functionality.

I believe that the code workflows/ ETLs provided in this report are reproducible and cohesive, though there are a few instances where I had to fall back on the GUI. I am led to believe the processes I completed in the GUI *are* possible in a Python notebook, at least for the ArcPro section. However, as the Network Dataset functionality is fairly recent to ArcPro (?), the documentation largely focuses on how to configure network datasets in the ArcPro GUI. Considering my current skill level (talkin' brute force Googling stuff like "python arcpy network attribute edit stackoverflow") I believe the ETLs presented in this report adequately represent how a Travelling Salesman problem may be executed in an Arc Notebook, start to finish.

Discussion and Conclusion

This report demonstrates ETL methodologies which operate in ArcPro & ArcOnline Jupyter Notebooks, which include processes such as, downloading coordinate and shapefile data via a variety of APIs, constructing a source network structure, creating a network dataset (or alternatively, utilizing ArcPro's proprietary network dataset), and developing a vehicle routing problem analysis, which describes the two routes two different delivery drivers can take to reach 10 different stops with the lowest cumulative time. The code developed in this report is intended to be reproducible and re-appropriated for similar network problems. The methods presented in this report may be powerful tools in further routing analysis problems, especially as more and more people and vehicles are creating GPS data that is remotely accessible. As these ETLs were developed completely* in Jupyter Notebooks, they may be used as building blocks for automated, real-time path finding for businesses involved in that realm, possibly increasing revenue due to the decreased needed work-time.

References

Add Vehicle Routing Problem Routes (Network Analyst). Esri. https://pro.arcgis.com/en/pro-app/latest/tool-reference/network-analyst/add-vehicle-routing-problem-routes.htm

County Boundaries, Minnesota. Minnesota Geospatial Commons. https://gisdata.mn.gov/dataset/bdry-counties-in-minnesota

EPSG:26915. spatialreference.org. https://spatialreference.org/ref/epsg/nad83-utm-zone-15n/

Find the shortest path and generate directions with Route. Esri. https://pro.arcgis.com/en/pro-app/latest/help/analysis/networks/route-tutorial.htm

Google Places Library. Google. https://developers.google.com/maps/documentation/javascript/places

Make Vehicle Routing Problem Analysis Layer (Network Analyst). Esri. https://pro.arcgis.com/en/pro-app/latest/tool-reference/network-analyst/make-vehicle-routing-problem-analysis-layer.htm

Plan Routes. Esri. https://doc.arcgis.com/en/arcgis-online/analyze/plan-routes.htm#ESRI_SECTION1_543AC2953593434BB26FA5275A4F36FC

Plan Routes. Esri. https://enterprise.arcgis.com/en/portal/10.4/use/plan-routes.htm

Road Centerlines (Geospatial Advisory Council Schema). (03/03/2020). Minnesota Geospatial Commons. https://resources.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_metrogis/trans_road_centerlines_gac/metadata/metadata.html

Vehicle routing problem analysis layer. Esri. https://pro.arcgis.com/en/pro-app/latest/help/analysis/networks/vehicle-routing-problem-analysis-layer.htm

Vehicle Routing Problem service. Esri. https://developers.arcgis.com/rest/network/api-reference/vehicle-routing-problem-service.htm

VehicleRoutingProblemSolverProperties. Esri. Retrieved 2021 from https://pro.arcgis.com/en/pro-app/latest/arcpy/network-analyst/vehicleroutingproblemsolverproperties.htm

Self-score

Category	Description	Points Possible	Score
Structural Elements	All elements of a lab report are included **(2 points each)**: Title, Notice: Dr. Bryan Runck, Author, Project Repository, Date, Abstract, Problem Statement, Input Data w/ tables, Methods w/ Data, Flow Diagrams, Results, Results Verification, Discussion and Conclusion, References in common format, Self-score	28	28
Clarity of Content	Each element above is executed at a professional level so that someone can understand the goal, data, methods, results, and their validity and implications in a 5 minute reading at a cursory-level, and in a 30 minute meeting at a deep level **(12 points)**. There is a clear connection from data to results to discussion and conclusion **(12 points)**.	24	23
Reproducibility	Results are completely reproducible by someone with basic GIS training. There is no ambiguity in data flow or rationale for data operations. Every step is documented and justified.	28	26
Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.	20	18
		100	95