Automatic Vehicle Location and General Transit Feed Standard Linear Referencing Project

Cole Anderson and Ying Song

Github Repository: https://github.com/and04671/AVL_RESEARCH_SP_22
Materials > Report: Project documentation report
Scripts: Python files for each major script section
Data: AVL and GTFS original and modified data files

Objective and Background

The first main objective of this project was to match AVL message pings with the corresponding plotted GTFS route by spatiotemporal factors, then linear reference the AVL messages along that matching GTFS route. Ideally, differences between the scheduled GTFS route stops/times and the linear referenced AVL route stops/times could be used to estimate delay locations for Metro Transit. The other primary objective for this project was to move the ArcGIS function steps required into a more universal Python code script using some sort of linear referencing software module (ArcPy or Open Source). Method based on the first half of *'Modeling spatial-temporal patterns of bus delays at and between stops using AVL and APC data and semi-Markov techniques'* (Song 2018). The time and data restraints only allowed for linear referencing the AVL data.

AVL Data

AVL stands for "Automatic Vehicle Location"

The AVL data is a set of 'ping' messages from bus transmitters that relay the location, direction, time, and service information for the bus. This dataset dates from 10/1/16 to 10/8/16 for the A-Line Metro. The messages were sorted and grouped extensively in both Pandas dataframes and Excel .csv files

The current A-Line diagram and schedule: https://www.metrotransit.org/route/aline

Important Fields:

Column	Description	Nullable?	Data Type
TRANSMITTED_MESSAGE_ID	Unique message identifier	No	Bigint
CALENDAR_ID	Service day identifier, format 1YYYYMMDD	Yes	Int
MESSAGE TYPE ID	Message type identifier	No	Smallint
LATITUDE	PLGR latitude as long integer * 10,000,000	Yes	Int
LONGITUDE	PLGR longitude as long integer * 10,000,000	Yes	Int
ODOMETER	Mileage in 1/100th miles, reset at startup/day	Yes	Int
SOURCE_HOST	Transmitter/bus ID number	Yes	Smallint
DIRECTION	Cardinal direction; 1 = southbound, 4 = northbound	Yes	Tinyint
MESSAGE_TIMESTAMP	Message time in UTC Standard time	Yes	Datetime
LOCAL_TIMESTAMP	Message time in local time	Yes	Datetime
SERVICE_ABBR	Service type: SAT, SUN, WK	Yes	Varchar

GTFS Data

GTFS stands for "General Transit Feed Specification"

The GTFS data is used to create a baseline scheduled movement to compare to the AVL message's spatiotemporal information. Dates are generalized; GTFS route days are generic within defined date ranges. The transit schedule/route data is split into component files defined below.

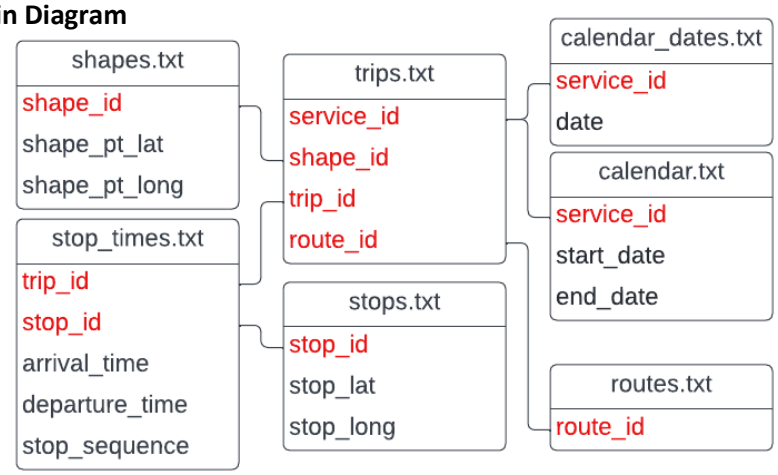
Each component file contains part of the full information, and multiple component files can be joined to each other on unique fields to define relationships between the contained data.

The GTFS dataset was downloaded from: https://transitfeeds.com/p/metro-transit/1284

File Description:

Component File	Description
calendar.txt	Defines the weekly service schedule with start and stop dates
calendar_dates.txt	Defines the exceptions to calendar.txt schedule
routes.txt	Gives the transit route information; name, number, etc (A-LINE)
shapes.txt	Defines the points of each trip's directional polyline
stop_times.txt	Defines the arrival and departure times at each stop on all trips
stops.txt	Defines the stop locations for pickup and drop off
trips.txt	Defines the sequence of stops, or trips, along a route

Entity Join Diagram



Geospatial Advisory Council (GAC) Schema Centerlines Data

GAC Schema Centerlines is a polyline dataset of roads in the Twin Cities metro, used to compare to create routed shapes. This dataset is in Universal Transverse Mercator NAD83, with units in meters. It was published on 03/03/2020 by the Metropolitan Council.

This dataset was collected from the Minnesota Geospatial Commons website:
https://gisdata.mn.gov/dataset/us-mn-state-metrogis-trans-road-centerlines-gac

Brief Process Overview

GTFS Comparison and Function Exploration (Project Meeting 01 - 03)

- Testing ArcGIS Pro's GTFSShapesToFeatures tool
- Comparing GTFS files between days to see if trip shape changes majorly from day to next.
- See scripts **1 Arcpy ShapesToFeatures** and **2 Comparing GTFS Files**

GTFS Joining (Project Meeting 04)

- Joining component GTFS files and querying down to only required rows.
- See script **3 GTFS Joining**

GTFS and AVL Sequencing and Summaries (Project Meeting 05 - 11)

- Trying to assign nesting sequence numbers to AVL messages in several different way. Ultimately, data issues made this non-sensical.
- Summarizing GTFS and AVL data by various grouping and sorting
- See scripts **4 GTFS and AVL Summaries** and **(OLD) GTFS and AVL Summary Tables**

Comparison and Linear Referencing (Project Meeting 12 - 14)

- Final weeks attempt to turn 1 trip into linear reference due to sequencing data issues.
- See script **5 Linear Referencing**

[2022-01-19] Project Meeting 00

Tasks:

- Set up Google Folders for the projects
    - (https://drive.google.com/drive/folders/1EGvZJIzOUNa9PDYcl7ZE0UJvOkh1ZED7?usp=sharing)
- Weekly meeting formats: progress report and task discussion
- Upload and organize the data into the Drive folder: https://drive.google.com/drive/folders/1l5UIM2H-9j_xTJ2WvoogaUpix8P1qGM3?usp=sharing
    - Road network data (attributes): (OSM-GIS or road centerline-DOT)
- Create a Jupyter project and read AVL data into pandas
    - Fields and field types?
    - What fields are potentially useful?

Reading fields in AVL A-Line Messages table

- Transmitted_Message_ID - Keep
- Calendar_ID: bigint, format 1YYYYMMDD, ie 120161001 - Keep
    - Datatype: bigint
- Message_Type_ID: - Keep
- Latitude: latitude in long integer, ie (lat*10,000,000) - Keep
    - Datatype: int
    - Importance: gives message/bus location
- Longitude: longitude in long integer, ie (long*10,000,000) - Keep
    - Datatype: int
    - Importance: gives message/bus location
- Adherence: difference between scheduled and actual, minutes - Keep
    - negative values are behind schedule
    - Datatype: smallint
    - Importance: indicates delay, paired with odometer
- Odometer: integer * 100, given in the 100th/mile. Restarts daily - Keep
    - Datatype: int
    - Importance: total distance travelled so far in the day
- Validity: - Drop
- Source_Class - Drop
- Source_Host: vehicle transmitter ID- Keep
    - Importance: gives the vehicle id
- Route_Version - Drop
- Messages_Version - Drop

- Route _Offset - Drop
- Direction: 1 is Southbound, 4 is Northbound - Keep
- TIME_POINT_OFFSET - Maybe
- STOP_OFFSET - Drop
- FLAG32 - Drop
- EFFECTIVE_SERVICE - Drop
- MDT_TIMESTAMP- Keep
- MESSAGE_TIMESTAMP- Keep
- PASSENGER_COUNT_ON: passenger boarding count - Keep
- PASSENGER_COUNT_OFF: passenger disembark count - Keep
- ST_MDT_VERSION - Maybe
- MSG_GROUP - Drop
- CAT_1- Drop
- CAT_2- Drop
- CAT_3- Drop
- CAT_4- Drop
- CAT_5- Drop
- CAT_6- Drop
- CAT_7- Drop
- CAT_8- Drop
- CAT_9- Drop
- CAT_10- Drop
- LOWER32- Drop
- UPPER32- Drop
- CURRENT_DRIVER
- FREE_TEXT_MSG - Maybe
- LOCAL_TIMESTAMP - Maybe
- MDT_BLOCK_ID - Maybe
- SIGNAL_STRENGTH- Drop
- BLOCK_ABBR - Drop
- ROUTE_ABBR - Keep
- PROPERTY_TAG - Maybe
- SERVICE_ABBR: service type (Weekday, Sun, Sat) - Keep
- NODE_ABBR: 4-letter code standing for - Keep
- FOM: GPS accuracy, 2 is high - Maybe
- dGPS - Maybe
- ValidOdo: Boolean - Maybe
- ValidAdh: Boolean - Maybe
- ValidLoc: Boolean - Maybe
- MESSAGE_TYPE_TEXT: ‘Vehicle Location’ or ‘Verbose Pass Count’ – Maybe

[2022-01-24] Project Meeting 01

Questions and Clarifications

- Questions: What does ‘overloaded’ mean? These rows have TPC/APC value filled
- Questions: Clarification on fields 17, 18, 42, 46, 49, 55

Task

- Create a key value for future “join” back to the original data

- Old scripts for fields reference (https://drive.google.com/drive/folders/1CTg1zKGXidH9AhExV9bDFKGYF8qn5g7h?usp=sharing)
- GTFS routes to shapefile/geometry (network geometries); AVL data-GPS (x, y, t) <linear reference to the routes>
- Reconstruct the geometries of routes/trips
    - Python packages (GTFS - trips)
        - gtfs-kit: https://pypi.org/project/gtfs-kit/
        - gtfs_functions: https://pypi.org/project/gtfs-functions/
        - grfsk: https://mrcagney.github.io/gtfstk_docs/
        - py_gtfs: https://pygtfs.readthedocs.io/en/latest/
        - Peartree: https://pypi.org/project/peartree/
        - partridge: https://pypi.org/project/partridge/
        - gtflib: https://pypi.org/project/gtfslib/
- Are the GTFS files the same for multiple dates? Compare using script.
    - Oct1, Oct4, Oct5 are the same
    - Oct5 not same as Oct6
- We have decided to just use Oct 1st for all days

[2022-01-31] Project Meeting 02

Task:

- Create route shapefiles from GTFS data, which will be used as the base for linear referencing.
- Explore ArcPy functions
    - Try GenerateShapesFeaturesFromGTFS_conversion
    - Does this tool produce direct links connecting two points? Or actual routes along roads?
- Check how to integrate GTFS with OSM (ultimately not used): https://wiki.openstreetmap.org/wiki/General_Transit_Feed_Specification

Check ArcPy functions

- Tool 1:
    1. Input: GTFS folder without shapes.txt file
    2. Tool: GenerateShapesFeaturesFromGTFS
    3. Tool: FeaturesToGTFSShapes
    4. Output: new shapes.txt file
    - GenerateShapesFeaturesFromGTFS_conversion - if there is no existing shapes.txt file in the GTFS dataset, and you want to estimate shapes based on the stop, route, and schedule information from an existing GTFS dataset, then creating new shapes.txt file with FeaturesToGTFSShapes tool
    - This tool also requires spatial data for a road network
    - You seem to be able to generate straight lines OR along a network with this tool (although without the network there are some straight and bent). Also creates a transit stop layout.
    - FeaturesToGTFSShapes_conversion - Creates a shapes.txt file for a GTFS public transit dataset based on route line representations created by the GenerateShapesFeaturesFromGTFS tool. This also creates a new stop_times.txt file
- Tool 2:
    1. Input: GTFS with existing shapes.txt
    2. Tool: GTFSShapesToFeatures

3. Output: New feature class
   - GTFSShapesToFeatures_conversion – use if GTFS dataset contains existing shapes.txt and you want to convert the shapes to feature class
   - This seems to create a network of existing shapes

[2022-02-07] Project Meeting 03

Use Python scripts to generate transit route and stop shapefiles

- Create supporting selection queries for future use, then generate the transit route shapefiles and stop location shapefiles

Continuing some tasks from previous week:

- Query
  1. Select all trips and shapes for a given route
  2. Select all stops and stops_times
- Break down AVL data into trips; use the southbound and northbound information to generate trips from AVL data; add new sequence number (ID) for each trip
  - Generate a new table with the route ID, sequence number, first GPS time and location, last GPS time and location;
    - AVL (OID, datetime, log, lat, direction, trip_SEQ, device_id?) OR (OID, datetime, log, lat, direction, date_seq, trip_seq, device_id?)
    - Summary Table of AVL
      - (date_seq, trip_seq, device_id?, direction, first_datetime. first_log, first_lat, last_datetime, last_log, last_lat)
    - GTFS (date, route, trip, <vehicle?>); {stop_time} for each trip, shape
    - Summary Table of GTFS
      - (date, trip_seq,vehicle_id?, direction, first_stop_time, first_st_log, first_st_lat, last_stop_time, last_st_log, last_st_lat)
  - Use start time, end time, and direction of a GTFS trip; use first GPS time, last GPS time, and direction from AVL data, then get the shape of the trip for a given set of GPS data.

Join process brainstorming: Query Steps V1 and V2

	Only want specific routes	*Only want specific times/days*
1	Query routes.txt by route_short_name to give route ID	Query calendar_dates.txt by date to give service ID
2	Join trips.txt on route.txt(tgt) by route_ID to get shape_ID	Join trips.txt on calendar_dates.txt by service_ID to give shape_ID
3	Join shapes.txt on trips_query.txt by shape_ID to get revised shapes	Join shapes.txt on trips_query.txt by shape_ID to get revised shapes
4	Put shapes.txt into GTFSShapesToFeatures	Put shapes.txt into GTFSShapesToFeatures

Only want specific routes	*Only want specific times/days*	
	• Pare calendar_dates.txt down by date	
• Pare routes.txt down by route ID • Select and save as routes_new.txt • Join trips.txt on routes_new by routeID • Select as trips_new.txt • Remove Join	• Join calendar.txt on calendar_dates.txt • Select as calendar_new.txt • Remove Join • Join trips.txt on calendar_new by date_ID • Select and save as trips_new.txt	
	• Join stop_times.txt on trips_new • Select and save as stop_times_new.txt • Remove Join	
• Join shapes.txt on trips_new • Select and save as shapes_new.txt	• Join shapes.txt on trips_new • Select and save as shapes_new.txt	

[2022-02-14] Project Meeting 04

Continue some tasks from previous week:

- Query
    1. Select all trips and shapes for a given route
    2. Select all stops and stops_times
- Break down AVL data into trips; use the southbound and northbound information to generate trips from AVL data; add new sequence number (ID) for each trip
    - Sequence each for northbound and southbound? Or just one?
- Cut down the .txt files based on query (GTFS Joining.py)
- Put new the shapes, stops, stop times files, etc. in a new folder (Folder makes up GTFS dataset)
- Generate feature classes of shapes and stops (GTFS Shapes to Features): this will not work if the format of new files isn't the same..
- Geopandas will not work in the ArcGIS python environment (and ArcPy won't in Anaconda)

[2022-02-21] Project Meeting 05

Generate New Tables

- **AVL Table**
    - Create an intermediate AVL table (OID, datetime, log, lat, direction, date_seq, trip_seq, device_id?)
    - Trip sequence starts over each day, which might end at midnight, might not; D1-T1, D1-T2, D2-T1, etc.
    - Create AVL Summary Table:
        - (date_seq, trip_seq, device_id?,direction, first_datetime, first_long, first_lat, last_datetime, last_long, last_lat)
- **GTFS Table**
    - Create an intermediate GTFS table (date, route, trip, <vehicle?>); {stop_time} for each trip and shapes; probably will need some joins here
    - Create GTFS Summary Table

- (date, trip_seq, vehicle_id?, direction, first_stop_time, first_st_log, first_st_lat, last_stop_time, last_st_log, last_st_lat)

Matching GTFS to AVL:

- Use start time, end time, and direction of a GTFS trip; first GPS time, last GPS time, and direction from AVL data to get the shape of the trip for a given set of GPS data
- Solution 1: Join based on two sets of keys (blue and orange)
  - Calculate the distance between the first/last GPS location and the first/last stop for the matched trip (same bound, similar duration)
  - Calculate the time difference between the first/last GPS location and the first/last stop_time along the matched trip
  - The final output is a joint table (trip ID for GTFS; route_ID + sequence for AVL)
- Solution 2: Join based on the (log, lat, time, direction, vehicle) of AVL and GTFS summaries
  - The final result is a linked/joint table by these keys (blue and orange) <(date_seq, trip_seq, device_id), (date, trip_seq, vehicle_id?)>

Questions and clarifications

- Message timestamp: time in UTC; sometimes this doesn't align with calendar time day
- Local timestamp: time in local time
- Calendar_ID indicates schedule day, not time day
- There are 664 rows with no direction value: these seem to be when the bus is parked at a garage, as the odometer isn't going up
- The sequence codes for the bus should probably be the actual Source Host number and not a sequential value so the same number is always the same bus
- Nesting
  - SEQ1 = Day
  - SEQ2 = Day's Bus
  - SEQ3 = Bus's Trip
  - SEQ4 = Trip's Message
- Does the GTFS table need a trip sequence or trip ID?
- Calendar.txt & trips.txt table: Do I only want the trips with service ID's running through Oct 1-8?

Code for iterative numbering of trips inside bus inside days (See also: another version later on)

```
if: row's day# <> previous row's day#,
        bus# resets to 1 and tripcount# resets to 1
else (day# is same):
        if: row bus# <> previous row's bus#,
                row's tripcount# reset to one
        else (bus# is same):
                if: row dir# <> previous row's dir# ,
                        row's tripcount# = previous row's tripcount# + 1
                else(no change in dir#):
                        If: tripcount# <> previous row tripcount#,
                                row's ping resets to 1
                        Else (trip# is same):
                                row's ping = previous row's ping+1
```

- What about the pings with NO direction? NaN != NaN, so it is labeling each NaN as its own trip, creating lots of "1st" pings in a row: This messes the trip count up A LOT

[2022-02-28] Project Meeting 06

Clarifications

- Use the message timestamp: time in UTC vs. local timestamp

AVL DataSet Work

- AVL summary table: summarize by (Day, Bus), try the groupby function)
    - For each service day, what are the service periods for each bus?
    - Interested in knowing (a) total range, start and end time; (b) any time gaps?
    - We can use this as a reference for future discussion (a) order of service for each bus, (b) across different days? (c) help with next step of joining
- Will each AVL weekday calendar_id get the same whole set of GTFS trips with the single weekday service_id code?
- Early attempt at sorted AVL data:
    - DayBusSortAVL.csv: calendar_id, trans message id, message_time(UTC), source_host, odometer, route_abbr, direction, longitude, latitude, day_id (1-8), source_id (1-?)....then ordered by time
- Summarize by Day: (is just the precursor to above?)
    - How many buses are serving that day? and <required> service_id?
    - Are the set of bus numbers the same across different days?

GTFS DataSet Work

- GTFS summary table
    - GTFS (date, route, trip, <vehicle?>); {stop_time} for each trip; shapes
        - all_trip_shapes.csv: route_id, trip_id, service_id, trip_headsign, shape_id, shape_pt_lat, shape_pt_lon, shape_pt_sequence, direction
        - finalresultGTFS: trip_id, service_id, trip_headsign, start[arrival_time, departure_time, stop_seq, stop_id, stop_name, stop_desc, stop_lat, stop_lon], stop[arrival_time, departure_time, stop_seq, stop_id, stop_name, stop_desc, stop_lat, stop_lon]
- Does the GTFS table need a trip seq or trip ID?
    - Keep in mind that GTFS will provide the SHAPE for each trip, and schedule times at each stop along that trip.
    - We need to map each GTFS trip to the corresponding AVL trip.
- GTFS will provide the SHAPE for each trip; and schedule time at each stop along the trip. And we need to map each trip to the corresponding AVL trip.

[2022-03-07] Project Meeting 07

AVL DataSet Work - AVL tables created

- Will each AVL weekday calendar_id get the same whole set of GTFS trips with the single weekday service_id code?
- Early attempt at sorted AVL data:
    - DayBusSortAVL.csv: calendar_id, trans message id, message_time(UTC), source_host, odometer, route_abbr, direction, longitude, latitude, day_id (1-8), source_id (1-?)....then ordered by time

- Summarize by Day: (is just the precursor to above?)
    - How many buses are serving that day? and <required> service_id?
    - Are the set of bus numbers the same across different days?
- AVL Code: for each bus/device, sort by messenger time, break it into trips, create a new tripID

```
Df Sortby (SOURCE_HOST, message_time)
Current Host = none
Current Direction = none
Current Trip = 0
For Row, Index in sortdf.iterrows():
    Row Direction = Row['DIRECTION'],
    Row Host = Row['Source_HOST']
    If Current Host == none:
        if Row Direction exists:
                Current Host = Row Host
                Current Direction = Row Direction
                Current Trip = 0
            Sortdf.at[index, 'TripID'] = str(Current Host) + '_' + str(Current Trip)
    Else-If Current Host !=  Row Host:
        If Row Direction exists:
            Current Host = Row Host
            Current Direction = Row Direction
            Current Trip = 0
    Else:
        If Current Direction != Row Direction:
                Current Trip + 1
                        Etc….
```

- Summarize AVL trip information

```
new Numpy.array
for group in (Groupby_TripID):
                (sort by messenger time)
                (first row) First messenger time; first long/lat
                (last row) First
                 (route_abb) 'unique'  # 'good'
                 (direction) 'unique'    # ''
                 (total number of GPS data)
                append('TripID, start_time, start_log, start_lat, end_time, end_log, end_lat,
                route_abb, direction', 'startWSS', 'endWSS',
                'cntPts',  'Label')
```

[2022-03-14 to 03-28] Project Meetings 08 - 10

GTFS DataSet Work

- SumtableGTFS – group by service_date, sort based on departure_time
    - For each [service id (as in like the BUS-WK-01), direction], get:
        - First departure_time, last departure_time from the first stop
        - First arrival_time, last_arrival_time from the last stop
        - Total number of trips per serviceID and direction
- Make sure to check the original data so that the last record of service is 26:xx:xx is not in the middle of the day (is it a duplicated record?, maybe reorder by departure time?)

- Check original GTFS data for start and end stop information for north bound? If not, set start as the end of south bound and set the end as the start of the south bound (There were some rows with no endpoint location)
- sumtableGTFS: trip_id, service_id, trip_headsign, stop_seq (1), stop_lat, stop_lon; arrival_time, departure_time, stop_id, stop_name, stop_desc, stop_seq (-1), stop_lat, stop_lon; arrival_time, departure_time, stop_id, stop_name, stop_desc,

AVL DataSet Work

- Double-check the scripts and data
  - The count of AVL records per trip is 150? 200+? In some cases
  - Why the last few records had inconsistent times?
  - Whether the "service date" make sense
  - Convert UTC datetime to local times for future use
  - Double check the date
    - via the total number of records
  - Create DOW (weekday/saturday/sunday)
    - Use this field to join to <service ID> (later discover this will not work- data error)
- SumtableAVL: 'TripID, start_time, start_log, start_lat, end_time, end_log, end_lat, route_abb, direction', 'startWSS', 'endWSS', 'cntPts', 'Label')

Both

- Buses running over midnight for a particular service ID, so can't split on 00:00
- Based on calendar_date, (ie, service date, not literal) get the corresponding service id (ie, the BUS-WK-01 code from GTFS)
- Then sort AVL messages by direction, then trip start time, so that it matches the GTFS results
  - I think the message counts are correct. They add to the message total?
- For each AVL [service_id, service date, direction], left-join the GTFS [service_id, direction], and take difference between
  - Service_ID is just the BUS-WK code
  - Find first departure time difference (AVL-GTFS), last departure time difference (AVL-GTFS), First arrival time difference(AVL-GTFS),
- AVL data: Run sorted_values(['service_date', 'direction', 'trip start time']) so that it matches the GTFS results

[2022-04-04] Project Meeting 11

AVL DataSet Work - Group by ordering

- Group by Source_Host then SortBy local timestamp
- Local timestamp is not as specific as message timestamp; several messages in a row are given the same time.

2016-10-01T13:31:13.320Z	2016-10-01T08:28:09.817Z
2016-10-01T13:33:13.317Z	2016-10-01T08:28:09.817Z
2016-10-01T13:34:13.313Z	2016-10-01T08:28:09.817Z
2016-10-01T13:35:13.327Z	2016-10-01T08:28:09.817Z

  - But if we remove duplicates, there is just one of each local timestamp time
  - Here are the breaks for Bus #2639, which attempt to show where the service day changes (we later find these incorrect)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	CALENDAF	MESSAGE_	LOCAL_TIN	SOURCE_H	ROUTE_AE	DIRECTION	LONGITUD	LATITUDE	TIME	PREVDAY	NEXTDAY		
2	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.49E+08	10/2/16 6:24	TRUE	FALSE	End of day 1	
3	1.2E+08	2016-10-0	2016-10-0	2639	921	4	-9.3E+08	4.49E+08	10/2/16 13:47	FALSE	TRUE	Start of day 2	
4	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.49E+08	10/3/16 6:29	TRUE	FALSE	End of day 2	
5	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.5E+08	10/3/16 11:57	FALSE	TRUE	Start of day 3	
6	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.49E+08	10/3/16 23:18	TRUE	FALSE	End of day 3	
7	1.2E+08	2016-10-0	2016-10-0	2639	921	4	-9.3E+08	4.49E+08	10/4/16 10:45	FALSE	TRUE	Start of day 4	
8	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.49E+08	10/5/16 0:47	TRUE	FALSE	End of day 4	
9	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.5E+08	10/6/16 11:21	FALSE	TRUE	Start of day 5	
10	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.49E+08	10/6/16 23:23	TRUE	FALSE	End of day 5	
11	1.2E+08	2016-10-0	2016-10-0	2639	921	1	-9.3E+08	4.5E+08	10/7/16 10:36	FALSE	TRUE	Start of day 6	

1	MESSAGE_TIMESTAMP	LOCAL_TIMESTAMP	SOURCE_HOST	DIRECTION	TIME
2	2016-10-01T14:00:50.890Z	2016-10-01T09:00:50.890Z	2639	1	10/1/16 9:00 AM
18	2016-10-02T18:47:00.203Z	2016-10-02T06:24:37.180Z	2639	1	10/2/16 6:24 AM
19	2016-10-02T18:47:45.710Z	2016-10-02T13:47:45.710Z	2639	4	10/2/16 1:47 PM
36	2016-10-03T16:24:44.873Z	2016-10-03T06:29:49.803Z	2639	1	10/3/16 6:29 AM
37	2016-10-03T16:57:21Z	2016-10-03T11:57:17.210Z	2639	1	10/3/16 11:57 AM
31	2016-10-04T15:45:24.243Z	2016-10-03T23:18:53.140Z	2639	1	10/3/16 11:18 PM
32	2016-10-04T15:45:27.037Z	2016-10-04T10:45:27.037Z	2639	4	10/4/16 10:45 AM
89	2016-10-07T15:35:22.800Z	2016-10-06T23:23:53.180Z	2639	1	10/6/16 11:23 PM
90	2016-10-07T16:13:29.960Z	2016-10-07T10:36:14.420Z	2639	1	10/7/16 10:36 AM
08	2016-10-06T16:21:44.907Z	2016-10-05T00:47:16.070Z	2639	1	10/5/16 12:47 AM
09	2016-10-06T16:52:40.880Z	2016-10-06T11:21:58.260Z	2639	1	10/6/16 11:21 AM

AVL DataSet Work - GroupBy

- Order by source_host (bus), then time
  - We still need to add 'service_type' marker
- Take the first and last time for each service day (for that one bus)
- Get all AVL messages sorted (by service_type, direction, trip start time)
  - something like give value until >3 hr time gap, then add one?

GTFS DataSet Work - GroupBy

- Order by service day
- Then by direction
- Assign trip sequence numbers (+1 when direction change, reset at day change)
- Then take the start/stop times each trip

[2022-04-18] Project Meeting 12

Use Python to realize the Linear Refencing process

- Tested in Arc previously
- Create route_shape using GTFS toolset (trip_shape), output will be the (trip_shapefile)
- Use the projected coordinate system for inputs (UTM 15N or StatePlane: meters)
- Input of the function (GPS_series, trip_shapefile)
- Output (LRDis_along_routes; log, lat_along_routes; timestamps; sequence)
- Further deal with cases when LRDis is not ascending

[2022-04-25] Project Meeting 13

Process for Linear Referencing

- Tested in both ArcGIS GUI and Python scripting

- Take 1 trip and line up to a GTFS shape_id
- Take S and N route from GTFS shapes file, took 1 S and 1 N trip worth of AVL messages.
- Create N and S routes layer from GTFS/AVL.
- Use Locate Features Along Route to reference
- Then use Make Route Event Layer to create display layer

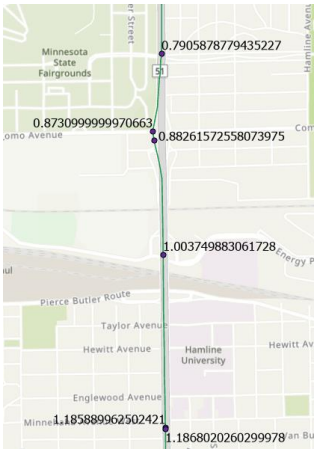
[2022-05-02] Project Meeting 14

- Report writing and code documentation cleaning

Results

The images on the left show the linear reference distances on scaled to 0-3. Note that the N and S route start ends are opposite, hence small numbers on the southbound and larger on the northbound. The images on the right shows all event locations for the same route.

Southbound Route with AVL Events:



Northbound Routes with AVL Events:

Discussion and Conclusion

Data and Challenge Discussion: Python scripting was a major, if not main, component of this project. Objectives were accomplished through ArcPy and Pandas only, rather than the ArcGIS Pro interface. This makes rerunning the same settings repeatedly far less time consuming and less error-prone. The AVL data was a large enough dataset where the whole frame could not be analyzed at once, and this made spotting errors without automatic error catchers more difficult. Data wrangling and cleaning took up 80% of the allotted project time (1 Semester). The GTFS data was straight forward after learning the basics of GTFS joins, and had no major problems. Trying to write code for the AVL messages to summarize/sort by day, bus, and time (especially time) took a frustrating amount of time and proved a major code roadblock. One ordering would be fixed but then another error was created elsewhere. It is likely that there is some error in the AVL dataset, as the Serviceday_ID, Calendar_ID, and Message_Timestamp would have strange overlaps at service day turnover. The times from Message_Timestamp had large time breaks in different places than the GTFS route data denoted. Another code challenge was figuring out how to add nested sequence numbers using a script. A series of nested if-then statements seemed to be the correct method.

Skills Learned *(less formal):* I learned a lot about real life data during this project. The disorder of 'actual' data means holes, duplicates, and errors throughout. The data processing took much longer than I figured it would; scope must consider data quality and size. My Python coding improved considerably; creating/linking multiple scripts and trying to keep them as soft coded as possible proved challenging. The project took about 16 hours per week. This felt like both too much and too little at different points throughout the semester, but my time management and organization skills around research improved regardless.

References

*Many of these papers were used to learn processes and are not cited in text.

Curtin, K. M., Arifin, R. R., & Nicoara, G. (2007). A Comprehensive Process for Linear Referencing. *Journal of Urban and Regional Information Systems Association*, *19*(2), 41–50.

ESRI. (2022). *An overview of the linear referencing toolbox*. ArcGIS Pro Documentation. Retrieved May 8, 2022, from https://pro.arcgis.com/en/pro-app/2.8/tool-reference/linear-referencing/.

Metropolitan Council. (2020). *Road Centerlines (Geospatial Advisory Council Schema).* Minnesota Geospatial Commons [distributor]. https://gisdata.mn.gov/dataset/us-mn-state-metrogis-trans-road-centerlines-gac

Metro Transit. (2016). Metro Transit GTFS. Metro Transit GTFS - OpenMobilityData. Retrieved May 8, 2022, from https://transitfeeds.com/p/metro-transit/1284

Metro Transit. (2022). A Line Roseville- St Paul-Minneapolis. Retrieved May 8, 2022, from https://www.metrotransit.org/Route/921

Song, Y. (2018). Modeling spatial-temporal patterns of bus delays at and between stops using AVL and APC data and Semi-Markov Techniques. *CICTP 2018*. https://doi.org/10.1061/9780784481523.067

Song, Y. (2020). *Geog 8292 Workshop 02 Network Accessibility – GTFS. University of Minnesota GEOG 8292 Urban GIS*.

Song, Y (2022). *Aline-vehicle-messages-Oct1_Oct8-2016.* University of Minnesota [university]