**DSSG 2015 Paratransit Group’s King County Metro Access Bus Rescheduler:**

**Use Cases:**

1. The bus breakdown: the user may input a bus run number (alpha-numeric such as “502R”, “302”, etc.) and an “initial time” representing the time *after* which all passengers that were originally scheduled to be serviced by the requested bus will be placed on to other Access bus schedules. For example, if bus 502R changes from functional to “out of service” at 12:00 PM, there may be any number of passengers who were supposed to be picked up and dropped off by this bus later in the day. The rescheduling application treats these clients as “unhandled requests” and will try to present alternative routing options for each client. These alternative options will include between 0 and 3 buses on to which each client can be rescheduled. Those alternative buses are the ones with the shortest “inconvenience time” relative to their original schedules. The new schedule of the alternative bus with the absolute shortest inconvenience time is presented to the user, that is, it has the unhandled request inserted into the alternative bus’s schedule. The estimated cost of a taxi, along with the estimated costs of dispatching a new Access bus to service all remaining unhandled requests, are also presented.
   1. It is important to note that, currently, any clients who are scheduled to be riding the bus at the “initial time” will not be rescheduled, because we do not have the ability to access any bus’s exact geospatial information continuously in time. Only those clients who have an estimated pick-up time after the initial time will be rerouted.
2. Individual unhandled requests: the user may input the individual Booking ID of any number of clients present in the schedule data. The rescheduler will try to move each Booking ID onto a suitable alternative bus, in the same fashion as the first use case. The user should note that the first use case is actually a subset of the second use case: the user could potentially enter all of the Booking ID’s of the clients with estimated pick-up times after the “initial time” and reschedule the remaining broken run’s rides for the day.

**How busRescheduler.py works:**

1. User input. There are potentially 9 command line arguments that can be fed to the *busRescheduler\_run* function: a demo schedule file name, AWS access key, AWS secret key, broken run ID, path to output directory, initial rescheduling time, individual Booking ID’s, the size of a requested time window (30 minutes default), and a threshold radius.
2. Gathering schedule data. Check if the desired data output directory exists, and if not, make a ‘data’ directory in the current directory. If supplied a demo schedule filename, load the demo file, else, use the access key and secret key to gather schedule data stored in the S3 Paratransit bucket.
   1. Note: we currently receive 15-minute streaming data from KCM in a Dropbox folder on Valentina’s Macbook. The contents of that folder are moved from Valentina’s machine to an S3 bucket. The *s3\_data\_acquire* function asks for streaming data occurring on **today’s date**. If no streaming data has been moved to the bucket on today’s date, the S3 data will not be loaded.
3. Once the schedule data has been loaded as a Python Pandas dataframe, use the *add\_TimeWindowsCapacity* module (which is imported within the *all\_functions.py* file) to add time windows and capacity counts to the schedule data.
   1. Time window construction is as follows: If the client is being picked-up, the time window is plus/minus 15 minutes on either side of the recorded “schtime” column. If a client has a ReqLate time, their drop-off time window runs from 30 minutes (by default) before the specified ReqLate time, and the ReqLate time. If the client is being dropped-off but has no ReqLate time, then the drop-off time is plus/minus an hour on either side of their original ETA.
   2. Both ambulatory and wheelchair client flag columns are appended to the schedule as well.
4. Obtain URID’s (unhandled request ID’s): Check to make sure that either the individual Booking ID’s or the requested bus run ID are present in the schedule data. If a bus run ID has been submitted along with an initial rescheduling time, check to see if there are remaining URID’s on the bus after the initial rescheduling time. URID is a class object with attributes pertaining to each Booking ID that remains to be rescheduled.
5. Begin a loop over the URID’s. For every URID, first we use the *radius\_Elimination* function. This uses the radius threshold argument to eliminate potential bus runs that are further than the radius threshold (default is 3 miles), in terms of straight-line distance, away from the URID throughout the entirety of

**Instructions for use:**

1. For a Windows machine, follow the instructions at <https://github.com/DSSG-paratransit/main_repo/blob/webapp/Python_venv/Windows_instructions.md>
2. Upon navigating to localhost:5000/admin, we begin the process of acquiring scheduling data. The user must supply the application with a day’s full schedule of Access bus routes. There are two options for accomplishing this: the user can use a demo schedule file or obtain semi-real-time streaming data. The demo file should be pre-processed for “useless data”, i.e. for rides with missing locational data or for rides that never leave the garage, etc. To access the streaming data, the user needs to supply the correct access key and secret key to the Amazon Web Services’s S3 bucket that currently receives the streaming data supplied by KCM. Upon supplying the appropriate fields on the /admin page, the user is redirected to the /display page.
3. Use case specification: the user should specify whether they would like to reschedule either a list of individual Booking ID’s or the Run ID of a bus and an initial rescheduling time.
   1. If there are individually requested Booking ID’s that are not present in the acquired schedule data, the user will be redirected to an error page.
   2. If the requested Run ID is not listed in the schedule, the user will be redirected to an error page. Similarly, if there are no clients scheduled to be picked up on the requested Run ID *after* the initial rescheduling time, the user will be redirected to an error page.
4. Upon successfully finding alternative transit options for the various unhandled requests, the user is automatically redirected to the /preferred\_options page. On this page are listed the buses with the lowest inconvenience time for each unhandled request. Each unhandled request’s Booking ID has a link that will send the user to the /alternative\_options page that has the top 3 buses with the lowest inconvenience time, along with some information about the feasibility of placing the unhandled request of interest onto each alternative option. The schedule of the bus with the lowest inconvenience time is also presented on this page.

**Possible improvements to the application:**

* Regarding the second use case, it may be particularly useful for KCM dispatchers to be able to change the pick-up and/or drop-off time for each unhandled request. For example, if a client’s appointment time was changed at the last minute, or if a client is running late, the dispatcher might enter the client’s estimated pick-up and/or drop-off time and subsequently use the rescheduling algorithm to place the client onto a bus more appropriate for the client’s new time restrictions. In particular, it would be helpful to juxtapose the “inconvenience times” of the alternative buses to the cost of placing the client in taxi. Being able to change a client’s new time restrictions would not be extremely difficult to implement and may add large value to the application for KCM users.