# Private Transport Route Choice Implementation in SimMobility

In real life scenarios where a traveller is to travel from an Origin to Destination, there exists a number of paths that the traveller can choose from. There are a variety of parameters that a traveller considers before choosing the path which is right to the best of his knowledge. These parameters include but not limited to shortest path in terms of time and distance, congestion, ease of travel, cost etc. The simulated environment is not very much different. Simulators try to cover as many parameters as possible to replicate the real life scenarios. This writing aims to cover the the following steps required to find the best path connecting ODs. SimMobility implements path related tasks in its PathSet Manager module. SimMobility has a separate *configuration file* in XML format dedicated to PathSet Manager settings. The file name usually is *data/pathset_config.xml* but you can always find the valid file name in <*path-set-config-file*> tag in SimMobility's main XML configuration file. You can enable/disable pathset module in <*pathset*> tag, in the beginning of PathSet Manager configuration file. The rest of this document will be divided into the following sections:

A. path set generation
B. path set storage
C. path set retrieval
D. path selection

## A.Path Set Generation

*Path Set* is referred to the group of paths that connect a single origin to a single destination. In this section the various aspects of generating a pathset are described.

**Online-Offline Generation:** Pathset generation can be performed *online* or *offline* mode. Pathset generation mode can be configured in <*pathset*> tag, in the beginning of pathset configuration file. For online pathset generation, leave the *mode* attribute as *"normal"*. For offline generation use *"generation"* keyword.

In *offline* generation, SimMobility acquires the travel demand, and then generates pathset for each distinct OD. The result of this generation is a CSV file which conforms to the corresponding 'path' table structure in SimMobility's database. Therefore, this file can be manually bulk inserted into 'path' table in SimMobility's database. Simulation exits after completion of pathset generation without initiating the supply simulator.

In online generation, SimMobility starts its normal operation and when it needs a path for any of its ODs, it will consult the Pathset Manager. The Pathset Manager will search in its cache and database looking for a set of paths for the given OD. If the pathset is not find, Pathset Manager will generate the pathset immediately- online generation.

*Developer's Note:*

*Developer's Note:*

online pathset generation uses *sim_mob::PathSetManager::generateAllPathChoices* and offline pathset generation uses *sim_mob::PathSetManager::bulkPathSetGenerator* for their operation.

Generating a set of paths containing, almost, all sensible paths connecting an *Origin* to *Destination* requires implementing the following methods altogether:

- shortest path
- K-Shortest Path
- link elimination
- random perturbation

***Shortest Path*** Calculation is the process of finding the minimum accumulated weight of all small segments which are able to connect the requested Origin to Destination. SimMobility uses *A_Star* algorithm to find a shortest path from the graph of Road Network. To do that, simmobility maintains a graph with Weight factor W of the edges to be the *distance* of each road segment(aka edge). By definition, this graph has the capability to provide shortest path from all Origin nodes in a graph (O) to a Destination node(D).
*Developer's Note:*

- Shorest path is calculated through the following method:
  *sim_mob::PathSetManager::findShortestDrivingPath*
- The street directory implementation class for shortest distance path is :
  *sim_mob::A_StarShortestPathImpl*

**K-Shortest Path** Calculation is the process of calculating 1st, 2nd,...,kth shortest path connecting the requested Origin to Destination. SimMobility has implemented Yen's algorithm for- a configurable- K value. This implementation also includes a proposed enhancement to discard duplicate paths.
*Developer's Note:*

- K-Shortest path is calculated through the following method:
  *sim_mob::PathSetManager::genK_ShortestPath*
- The following implements k-shortest path:
  *sim_mob::K_ShortestPathImpl*

***Link Elimination*** extends Shortest path by providing alternate shortest path when part of a path is eliminated. This approach is suitable to ensure alternate path(s) when part of the network is blocked. To implement this feature, Simmobility sets Weight of the target link (to be eliminated) to a very large value. It then repeats the shortest path Algorithm again. This method has two more variations.

1. First variation is based on travel time(rather than travel distance),

2. Second variation (third version) is also based on travel time but considers more weightage to Highways.

Each of the above three methods will generate a number of paths to contribute to a richer path set. The paths generated by these variations may have some duplications which will be discarded.

*Developer's Note:*

- The following methods handle calculation of the above three types of paths:
  *sim_mob::PathSetManager::genSDLE* (Shortest Distance Link Elimination)
  *sim_mob::PathSetManager::genSTTLE* (Shortest Travel Time Link Elimination)
  *sim_mob::PathSetManager::genSTTHBLE* (Shortest Travel Time Highway Biased)
  *sim_mob::PathSetManager::generateShortestTravelTimePath*
- Generation of paths in each of these methods is performd by black-listing a link (indirectly through roadsegments) and running the corrsponding shortest path implementation again. This is considered as a heavy task in *Online Pathset Generation* and will drastically porlong the execution of each simulation step. Therefore, each iteration will be assigned a worker thread to run the routines using the corresponding street directory class implementation. The worker thread runs by executing the following:
  *sim_mob::PathSetWorkerThread::run*

*TODO:*

The above mentioned variations of path generation methods are basic realization of an approach called *labelling approach*. Labelling approach is more mature in public route choice design. There are more travel time graphs being generated at the beginning of the simulation. Examples include combinations of peak hour and off-peak based graphs. These graphs are left unused. Invoking these graphs in pathset generation can contribute to reacher path sets.

***Random Perturbation*** method attempts to add to the number of paths in a set using the travel time graph by multiplying the weight(travel time) of each edge in the graph by a random number and generating shortest travel time path again. The entire process of multiplying weights by a random number is repeated multiple times at the beginning of simulation. Therefore, there are several number of graphs initially produced by simmobility which contain random weight edges. The number of iterations(number of random graphs) and the range of the random number are configurable.

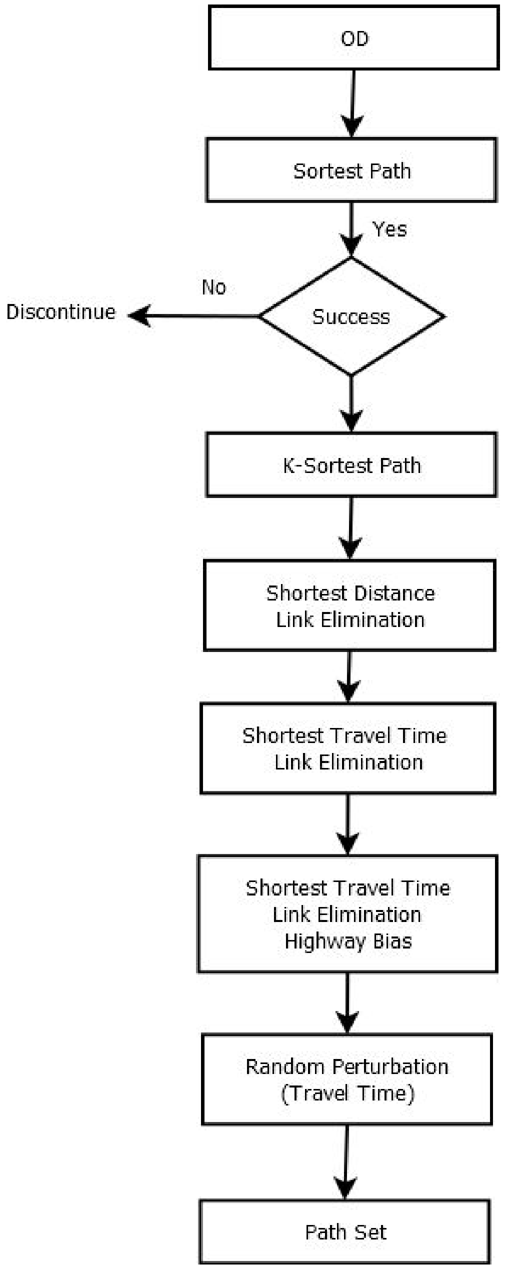
TODO:

Consider generating random perturbation graphs based on distance. The placeholder for this purpose is kept in <*random_perturbation*> tag under <*path_generators*>

In summary, the *current Implementation* of pathset generation has the following steps:

1. **Shortest Path**
2. **K-Shortest Path**
3. **Shortest Distance Link Elimination**
4. **Shortest Travel Time Link Elimination**
5. **Shortest Travel Time Link Elimination with Highway Bias**
6. **Random Perturbation**

The generation if performed in the following steps:


OD
Sortest Path
Yes
No
Discontinue
Success
K-Sortest Path
Shortest Distance Link Elimination
Shortest Travel Time Link Elimination
Shortest Travel Time Link Elimination Highway Bias
Random Perturbation (Travel Time)
Path Set

As the above diagram suggests, the entire process is dependent on initial generation of shortest path algorithm. If shortest path calculation fails, the path set generation routine for that specific OD will discontinue.

**Post pathset generation process:**

Once pathset generation is complete, some initial processing will be performed before the pathset is ready for storage. This includes finding pathsize, finding different labelling tags, and partial utility (utility without its time dependent parameters).

**Recursive pathset generation:**
pathset generation can be recursive. It means that, when a pathset for an initially given Origin-Destination (OD) is completed, simmobility will examine all the paths in the pathset and finds all the intersections visited by those paths. Each of these intersections will become a new Origin 'Oi' which, together with the initial Destination 'D' will form a new Origin-Destination (OiD). The pathset generation will be repeated for all these intermediated OiD's. Even when any of these intermediate nodes completed its pathset generation, the same process will be again applied to its further-intermediate ODs, and this recursive process continues until end. As one can see, this is a very heavy process as compared to one OD pathset generation. It will theoretically cover generation of pathset from 'all' network intersections(as Origins) to the initial Destination. Care is taken to avoid generating path sets which are already generated or being generated. To enable/disable recursive pathset generation, use <*recursive_pathset_generation*> tag in *pathset_config.xml*.

*Developer's Note:*
There is no method dedicated to recursive pathset generation. The logic is implemented at the end of the main path set generation method:
*sim_mob::PathSetManager::generateAllPathChoices*

*TODO:*
use configurable methods for pathset generation. For example, if there are multiple variations of link eliminations, choosing which one to use in pathset generation should be configurable. The placeholder for this configuration is available in <*path_generators*> tag.

**Network Graph:**
Generation of paths between ODs on any method requires a customized graph to be generated before hand. SimMobility's Street Directory module uses boost's adjacency list to generate graphs for drivers, pedestrians, buses etc. It uses distance and travel time in different forms as the weights of edges in its graphs.

*TODO*:
Street directory graph generation and initialization needs cleanup and usage improvement. For instance, generation of several graphs can be configurable. Moreover, the generation can be postponed to when pathset generation is really required. If paths had already been generated and used during simulation, generation of many graphs can be skipped. Moreover, only when, during simulation, a path is requested for an OD whose pathset has not been generated before, then online path set generation can trigger graph generation to subsequently generate a pathset.

*Developer's Note:*

- A Path Set is represented by a class called sim_mob::*PathSet*. Each path is represented by sim_mob::SinglePath class. The member variable sim_mob::PathSet::pathChoices keeps the record of paths(*SinglePath* objects) in a *PathSet* object .

- The corresponding methods in SimMobility for path set generation is :
  *sim_mob::PathSetManager::generateAllPathChoices.used in both online and offline.*
  *sim_mob::PathSetManager::onGeneratePathSet*
  *sim_mob::PathSetManager::setPathSetTags*
  *sim_mob::generatePathSize*
  *sim_mob::PathSetManager::generatePartialUtility*

- The current database stored procedure for pathset retrieval is usually *get_path_set(OD)* and can be configured in <*functions pathset="get_path_set"*> tag attribute. It returns database entries from the database's path table. Database path table name can also be configured through <*tables singlepath_table="path"*>.
- Column "id" in the path table is a csv string of road segments IDs that form a path. Column "pathset_id" contains Origin and Destination IDs separated by a comma.
- Kindly note that the steps after shortest path generation can and will generate duplicate paths. Care has been taken to discard and cleanup the duplicate paths to avoid logical and technical inconsistencies. This is especially important as it is suboptimal (or practically impossible) to use column 'id' in 'path' table in the database.
  - *TODO*: The 'scenario' column in the 'path' table currently holds a short code to indicate the pathset generation method which created that specific path. with a little care, this column('scenario') and column 'pathset_id' together can form an optimal primary key.

*[REF] Benchmark Route Choice Model::PathChoiceSetGeneration. Huang He*

## B.Pathset Storage

As mentioned above, paths are generated when there is no record of a path set for a given OD. Once the paths are ready, SimMobility will ensure that the expensive process of pathset generation will not be repeated for that specific OD within simulation or in subsequent simulation runs. To do that, simmobility will store the path set in an external data store. The current external source is chosen to be our postgresql database. As mentioned before, it is possible to specify the pathsets' table using <*tables singlepath_table="path"*> in the pathset configuration file.
Paths are stored in the form of a comma separated string containing Road Segment IDs.
Along with the paths in database some other invariant information are calculated and stored. These include:

1. Partial utility: calculates utility of the path with all necessary parameters except those which are related to travel time.
2. PathSize: Degree of overlapping for each path within its path set.
3. Number of signals along the path
4. Number of right turns along the path
5. Scenario: currently is a code indicating the generation method and its possible iteration number.

6. length of the path
7. length of the highway part of the path
8. Does this path have the minimum distance within its pathset?
9. Does this path have faces minimum number of Traffic Signals as compared to other paths in the pathset?
10. Does this path require minimum number of right turns as compared to other paths in the path set.
11. Does this path travel maximum highway distance as compared to other paths in its set?
12. A placeholder to indicate if a path is valid or invalid
13. Is this the shortest path(generated by the shortest distance graph)

*TODO:*

Columns MIN_DISTANCE and SHORTEST PATH should be practically same. Put proper checks, if at all necessary, to confirm that the paths outputted by SimMobility graphs as the shortest distance path is really the shortest path in the pathset. Then discard the MIN_DISTANCE column.

| **Column Name** | **Description** |
|---|---|
| ID | CSV string of path's Road Segment IDs |
| PATHSET_ID | OD |
| PARTIAL_UTILITY | time-independent part of utility Utility |
| PATHSIZE | Calculated Path Size |
| SIGNAL_NUMBER | Number of signal encountered in this path |
| RIGHT_TURN_NUMBER | Number of Right Turns Encountered in this path |
| SCENARIO | Currently stores generation method and iteration index |
| LENGTH | Length of the path in meter |
| HIGHWAY_DISTANCE | Distance of the Highway portion of the path |
| MIN_DISTANCE | Does this path have the shortest distance in the set |
| MIN_SIGNAL | Does this path have minimum number of traffic signals |
| MIN_RIGHT_TURN | Does this path have minimum number of Right turns |
| MAX_HIGHWAY_USAGE | Does this path the maximum highway usage in the set |
| VALID_PATH | is this a valid path |
| SHORTEST_PATH | Is this the shortest path |

| | |
|---|---|
| SERIAL_ID | unique serial Id |

Developer's Note:
The current functions responsible to carry out Path Set Storage are:

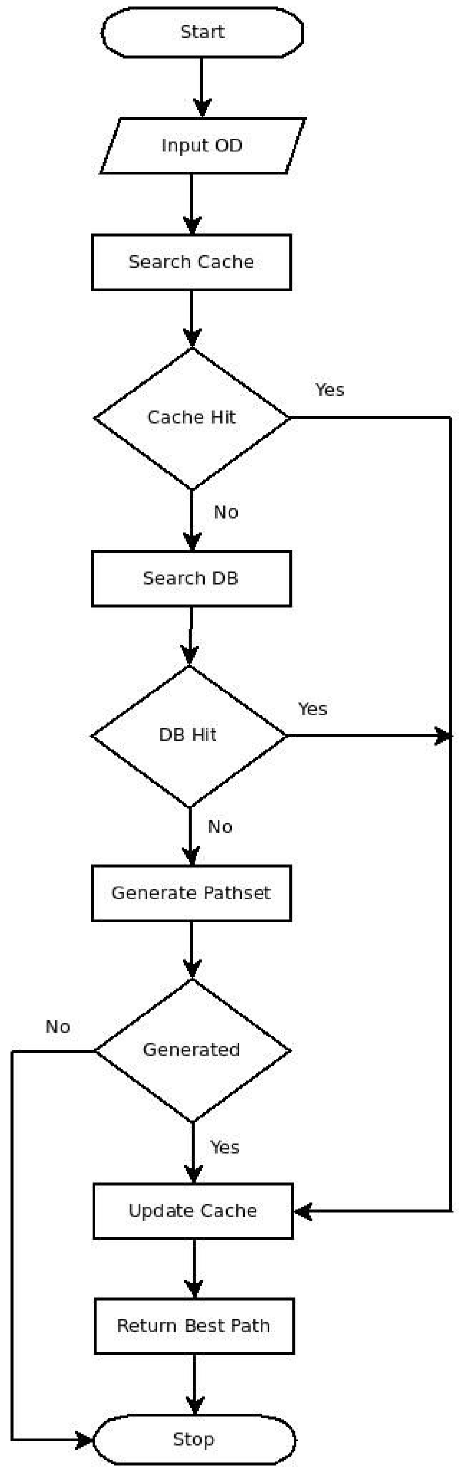
- *sim_mob::PathSetParam::storeSinglePath*
- *sim_mob::aimsun::Loader::storeSinglePath*
- *DatabaseLoader::InsertSinglePath2DB*

*TODO:*

- The functions involved (in *DatabaseLoader*) will eventually execute half hard-coded Sql queries. Porting the sql query to stored procedures can be considered.
- With emergence of new feature called CBD, an additional column can indicate if the target path is crossing any CBD area

## C. Path Set Retrieval

Path Set storage mentioned in the previous section is the external source of storing the paths. Simmobility also maintains a cache system for fast retrieval of the paths already been retrieved or generated  during the current simulation run. The flow is as follows:



As the flowchart presents, when a request for path for an OD is issued, the program searches in the cache. If not found, the database is searched. If that step failed too, the program will attempt to generate a pathset. Upon Success at any level the best path is chosen(described in the next section) and cache is updated accordingly. The cache system is a SimMobility's Utility which follows LRU (Least Recently Used) policy. The Cache container has a limited size that can hold only few thousands of path sets in itself. Upon a cache hit, the corresponding record is shifted forward in the cache ordering system to be marked as the latest used pathset. If a pathset is to be added to the already-full cache, the least recently used path set will be removed from the cache in order to make room for the incoming path set entry. The removed cache can be alway found in the database.

*TODO:*
cache size is hardcoded now(in PathSetManager's constructor) Make the cache size configurable in pathset configuration file. For example, high cache size for a large simulation on a low profile computer can run the system out of memory. Further, make SimMobility intelligently decide the cache size by first analyzing the hardware configuration of its host. More RAM naturally can accommodate higher cache size. This is also true for setting PathSetmanager's threadpool size which is currently configurable.

**Utility Calculation:**
After Pathset retrieval, some more data processing is required to prepare for final path selection. This process mainly calculates each ath's utility and the path set's logsum. Calculating utility includes includes finding *travel time* and *travel cost* and combining it with the partial utility which had been previously calculated for each path of the pathset during path set generation. Some of the factors affecting the utility like path length never change over time. These factors calculate *partial utility*. They are stored into the database along with the paths. But other contributing factors like *travel time and travel cost* depend on the *time* of day and the previous user *experiences*. Therefore they have to be combined with partial utility as and when the final utility is really required. This separation of utility factors will also improve the simulation performance as it avoid expensive recalculation of the partial utility. In short, *Travel Time* is the time experienced by users in the previous simulation runs. The travel Cost on the other hand is the cost incurred to the traveller for travelling through a path. It currently involves the ERP Gantry tolls along paths. As mentioned above, the path selection process uses these two parameters and the *partial utility* to calculate the *final Utility*. Before calculating the final utility, checks are performed to ensure that partial Utility had been previously calculated and is available.
*TODO:*

- Travel Cost only includes ERP. At least, it should also include petrol cost. Keep in mind to keep petrol cost configurable to be able to observe changes in drivers' decision when petrol cost changes.
- The coefficient for travel cost is currently zero. It means travel cost has no impact on travellers' path selection. Modellers are advised to come up with a sensible solution.

*Developer's Note:*
The following methods are involved in pathset retrieval:
*sim_mob::PathSetManager::getBestPath*
*sim_mob::PathSetManager::findCachedPathSet*
*sim_mob::PathSetManager::findCachedPathSet_LRU*

*sim_mob::aimsun::Loader::loadSinglePathFromDB*
*sim_mob::PathSetManager::onPathSetRetrieval*
*sim_mob::PathSetManager::getPathTravelTime*
*sim_mob::getPathTravelCost*
*sim_mob::PathSetManager::generateUtility*
*sim_mob::PathSetManager::generatePartialUtility*

## D. Path Selection

Path Selection is the process of choosing the best possible path based on an *Path-Size Logit Model* which involves multiple parameters. These parameters generally include but not limited to Travel Time of path, Travel Cost, Number of traffic Signals, Number of Right Turns, Having a Highway on the path, length of the path, time of day when the travel is done, purpose of travel etc.
The core of path selection algorithm(Path-Size Logit Model) consists of the following steps:

1. Find the utility of each path in the path set
2. Calculate logsum as the sum of the exponents of utilities
3. Use Utilities, logsum and a random factor to choose one of the paths from path set.

Here is the function with all the required steps:

### Utility Calculation parameters

From implementation point of view, In order to calculate Utility the following items should be retrieved/calculated first. The values of some of these items(described below) can be saved to database along with the paths without requiring to be changed later, whereas some of them are to be regenerated whenever a path selection is to be performed.

| | Item | To Be Saved | To be Regenerated | Default Coefficient value |
|---|---|---|---|---|
| 1 | Travel Time | | X | -0.01373 |
| 2 | Travel Cost | | X | 0.0 |
| 3 | Number of Signals | X | | -0.13 |
| 4 | Number of Right turns | X | | |
| 5 | Travel Distance | X | | -0.001025 |
| 6 | Highway Distance | X | | 0.5 |
| 7 | Path with Minimum Travel Time | | X | 0.879 |

| | | | | |
|---|---|---|---|---|
| 8 | Path with Minimum Distance | X | | 0.325 |
| 9 | Path with Maximum Highway Usage | X | | 0.00052<br>0.422 |
| 10 | Path with Minimum Right Turns | X | | 0.0 |
| 11 | Path with Minimum Signals | X | | 0.256 |
| 12 | Path Size | X | | 1.0 |
| 13 | work/leisure | X | | 0.0 |

Finding out the values like number of right turn or number of signal and their comparative tags like Path with Maximum Highway Usage is trivial. The rest of this section will be devoted to calculation of Travel Time, Path Size and Travel Cost.
*TODO:*

- There are some coefficients which are defaulted to zero. It doesn't make much sense. Modellers are advised to attend to this.

## Travel Time

Travel Time of a path plays a key role in choosing the path over the others. It is basically calculated from the time taken by drivers to traverse a Road Segment.

#### Types of Travel Time:

There are two types of Travel Time: First type is *Default travel time* which is attributed to each segment based on Some previous calculations. The next type is *Real Time Travel Time* which is based on the average time experienced in previous simulation runs to traverse a segment. Regardless of the type of travel time, each individual Road Segment can have several Travel Time records for different time periods of day. Therefore, every time an agent completes traversing a segment, the traversal duration of that road segment for the corresponding time period is added to a placeholder in a temporary data structure. This value is updated to the corresponding database table in the database. Table names are found in the configuration file in *tables tag*:

```
<tables realtime_traveltime="realtime_travel_time" default_traveltime = "Singapore_Updated_link_default_travel_time"/>
```

*Developer's Note:*
look for instances of *sim_mob::TT::TravelTimeCollector* data structure to see where and how it is used to store travel time information. Expect to see several *typedef*s that have built this structure. They are found in *shared/path/Common.hpp*. You will see that this data structure is eventually used in *TravelTimeManager* class. An instance of this class is in *PathSetManager*.

*TODO:*
As mentioned above, Travel time collector/manager is a member of pathset manager. It is not really necessary. It can be kept outside pathset manager, turned into a singletone or, probably for optimization purposes, have multiple instances.

**How Real Time Travel Time of a Road Segment is Calculated:**
At the end of simulation, the travel time data structure contains travel time of segments classified by *mode of travel, time period of day,total experienced travel times in each period* and *number of agents contributing to this total amount*.
Therefore, the real time travel time of a Road Segment for each mode and for each time period is equal to 'Total time' divided by 'Total Agents'. The time period is configurable through simmobility's main xml configuration file:

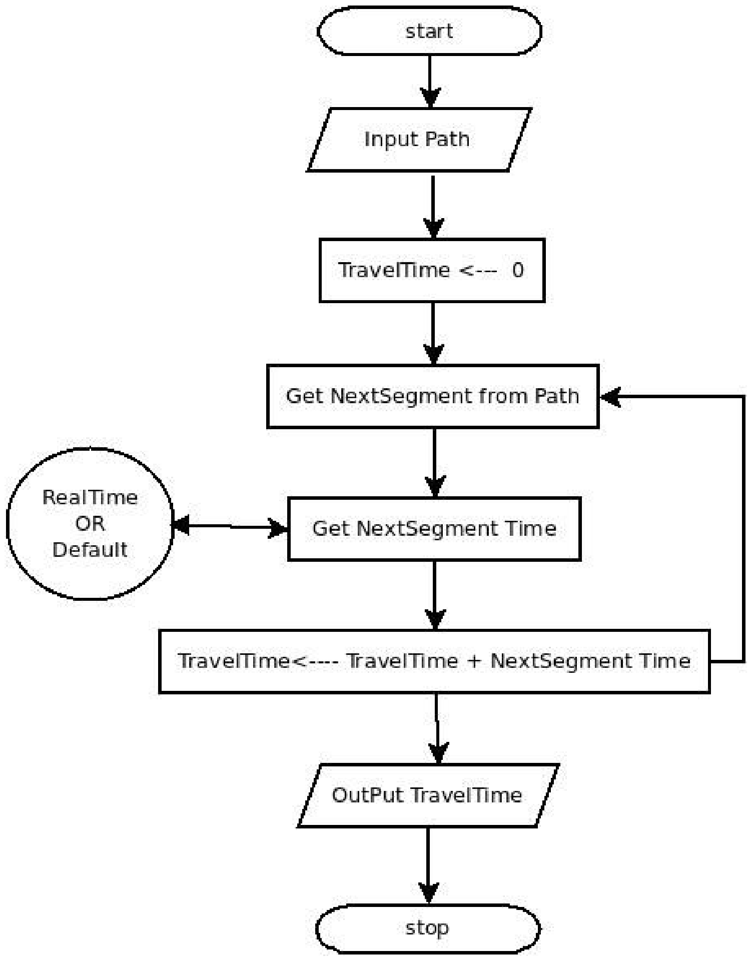
```
<travel_time_update interval="300" alpha="0.5" />
```

*TODO:*
Here, kindly provide a little information about the database table columns. I dont have access to database anymore.

**Real Time Travel Time are Life Cycle:**
Regardless of how the travel time is calculated during simulation and how they are used, the traveltime life cycle is as follows:

1. As with the retrieval from database, simulation initially retrieve default travel time and average travel time from the database and stores them in two different data structures in RAM. Note that these two data structures will not be modified throughout the simulation; the average travel time collected during the current simulation time will be stored in another data structure and it will be used to update the database at the end of simulation; as described in the previous section. Developers can Look for *sim_mob::LinkTravelTime* and *sim_mob::TT::TravelTimeStore* for the internals of the above two data structures. *sim_mob::TT::TravelTimeStore* will also be used for getting *In-Simulation* Travel time used in *re-routing* mechanism.
2. As an agent **enters** a Road Segment, its entry time is marked. The *entry time* decides which time period of the day this final report will belong to.
3. And when it exits the segment, its Road Segment Travel Status is reported to the *TravelTimeManager* to take care of the rest of the storage process considering the travel mode, travel period of the day, number of reports for each segment etc. For developers: This report comes in *Agent::RdSegTravelStat* format.
4. At the end of simulation, the processed simulation time reports are serialized into a file in CSV format.
5. Subsequently, this file containing the Real Time Travel time of all the traversed Road Segments will be UPSERTed into the real time travel time table. The database upsert function is configurable through:
   ```
   <functions pathset="get_path_set_vahid" travel_time="upsert_realtime" />
   ```
   **Note:** The upsert function requires an Alpha parameter which is configurable from simmobility's main configuration file:
   ```
   <travel_time_update interval="300" alpha="0.5" />
   ```

6. In subsequent simulation, this travel time information is retrieved from Database as described in step 1 and used in path selection algorithm(flowchart below).

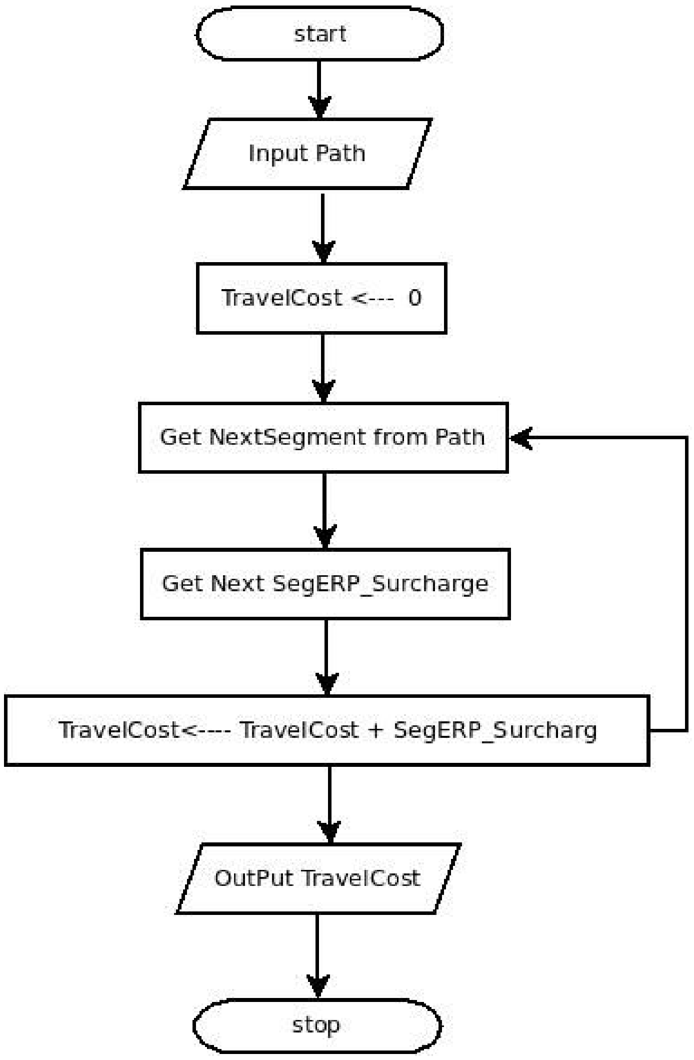


*Developer's Notes:*
Some of the methods involved in the above cycle are as follows:

- sim_mob::PathSetParam::getDataFromDB
- sim_mob::PathSetManager::getTravelTime
- sim_mob::PathSetManager::initTimeInterval
- sim_mob::PathSetManager::storeRTT
- *sim_mob::medium::DriverMovement::updateRdSegTravelTimes*
- *sim_mob::Agent::startCurrRdSegTravelStat*
- *sim_mob::Agent::finalizeCurrRdSegTravelStat*
- *sim_mob::PathSetManager::addSegTT*

## Travel Cost

Travel Cost means how much would cost for a driver if he chooses a specific path. Usually, this costs is incurred in the form of Road Tolls. In case of Singapore, ERP gantries are responsible for collecting Tolls from travellers. Travel cost is a time dependent parameter as the cost of passing gantries depends on the time of day. The information for Singapore’s ERP Gantries are available in Simmobility database and can be accessed when a path needs to calculate the total cost of traversing its segments:


start
Input Path
TravelCost <--- 0
Get NextSegment from Path
Get Next SegERP_Surcharge
TravelCost<---- TravelCost + SegERP_Surcharg
OutPut TravelCost
stop

*Developer’s Note:*

At present, calculating the travel cost with the information retrieved from database is performed in a function in PathSetManager.cpp :: getPathTravelCost()

TODO:

Add fuel cost to the travel cost. apparently, its coefficient is also ready.

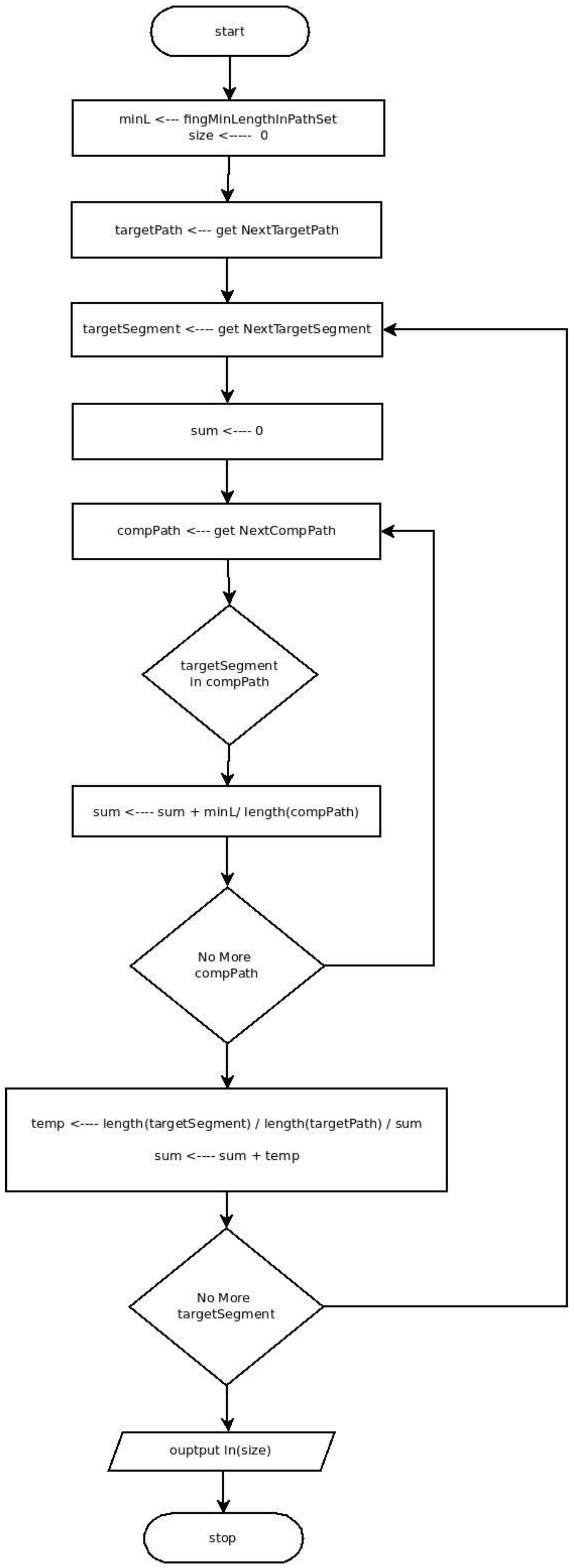
## Path Size calculation

By definition, the size of a path represents the level of overlap with other paths. This is a notable point in route choice model as according to this algorithm, a correction factor is used to take care of a path's common portion within a path set as the amount of shared portion of paths has an impact on utility calculation. PathSize of a path is calculated through the steps described in the flowchart below.

*Developer's Note:*
The Path Size calculation can be found in the following method:

- *sim_mob::generatePathSize*

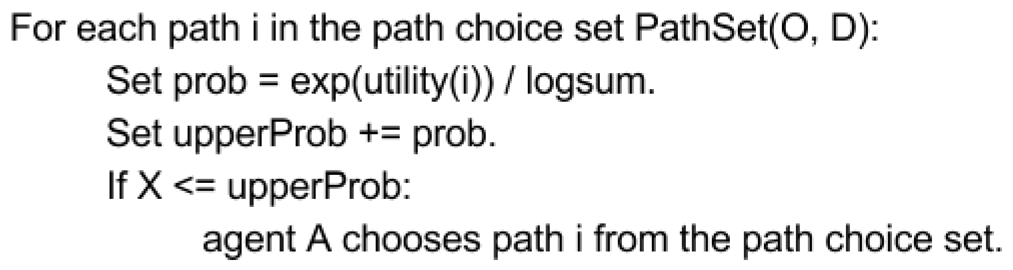
and the calculated value is stored in a member variable of same name in sim_mob::SinglePath.
*TODO:*
pathsize of each path is calculated following the pathset generation. This pathsize is stored along with the path in the databse and will not be recalculated again. It is foreseen that in the future enhancements, during the path selection, some paths in the path set will be temporarily discarded before a path selection meroutine is performed. Then all the pathsize values will be invalid as they had been previously calculated from the entire pathset but now are used to select a path from a smaller set. This is a note for future enhancements. a possible solution would be to recalculate path's pathsize value using the shrinked path set.
[REF]Benchmark Route Choice Model Path Size Calculation::Huang He


start
minL <--- fingMinLengthInPathSet
size <----- 0
targetPath <--- get NextTargetPath
targetSegment <---- get NextTargetSegment
sum <---- 0
compPath <--- get NextCompPath
targetSegment in compPath
sum <---- sum + minL/ length(compPath)
No More compPath
temp <---- length(targetSegment) / length(targetPath) / sum
sum <---- sum + temp
No More targetSegment
ouptput ln(size)
stop

### Utility and logsum Calculation

Finally, when all items are ready, it is time to calculate Utility and logsum. As mentioned before, the utility is calculated using the partial utility and time dependent parameters-travel time and travel cost. Every path will then have a utility. Sums of the exponent of these Utilities will be **logsum.**

After calculating the utilities and logsum, a Random value between 0 and 1 is drawn. Then, the pathset is iterated while accumulating their “exp(utility) / logsum” and comparing it with the random value. This process is continued until the accumulated value is larger than the random value. The following pseudo code can explain the final path selection :



```
For each path i in the path choice set PathSet(O, D):
    Set prob = exp(utility(i)) / logsum.
    Set upperProb += prob.
    If X <= upperProb:
        agent A chooses path i from the path choice set.
```

Developer’s Note:
Find the entire implementation in:
sim_mob::PathSetManager::getBestPathChoiceFromePathSet
[REF] R. Balakrishna, S. Sundaram, Y. Wen, S. Xu, S. Diwan (2008). Development of a Dynamic Traffic Assignment System for Real-Time and Planning Purposes: DynaMIT System User Guide Version 2.0.

## Future Directions:

The next feature foreseen at present is providing en-route choice implementation. The CBD area consideration is poorly implemented in pathsetmanager for demo purposes and should be cleaned up and improved when this topic is on demand. Provision of a better labeling approach as mature as public transit route choice seems necessary.