DhakaSim: A Microscopic Traffic Simulator

Input Format

The 5 input files DhakaSim needs are:

- Parameter.txt
- Node.txt
- Link.txt
- Demand.txt
- Path.txt

Parameter.txt

This file provides the necessary information needed to run the simulation in an input file which can also be given using the GUI. These information are:

- Simulation Speed
- Simulation End Time
- Pixel per Strip
- Pixel per Footpath
- Pixel per Meter
- Probability of Accident
- Strip Width
- Footpath Strip Width
- Maximum Speed of a Vehicle
- Pedestrians Allowed or not

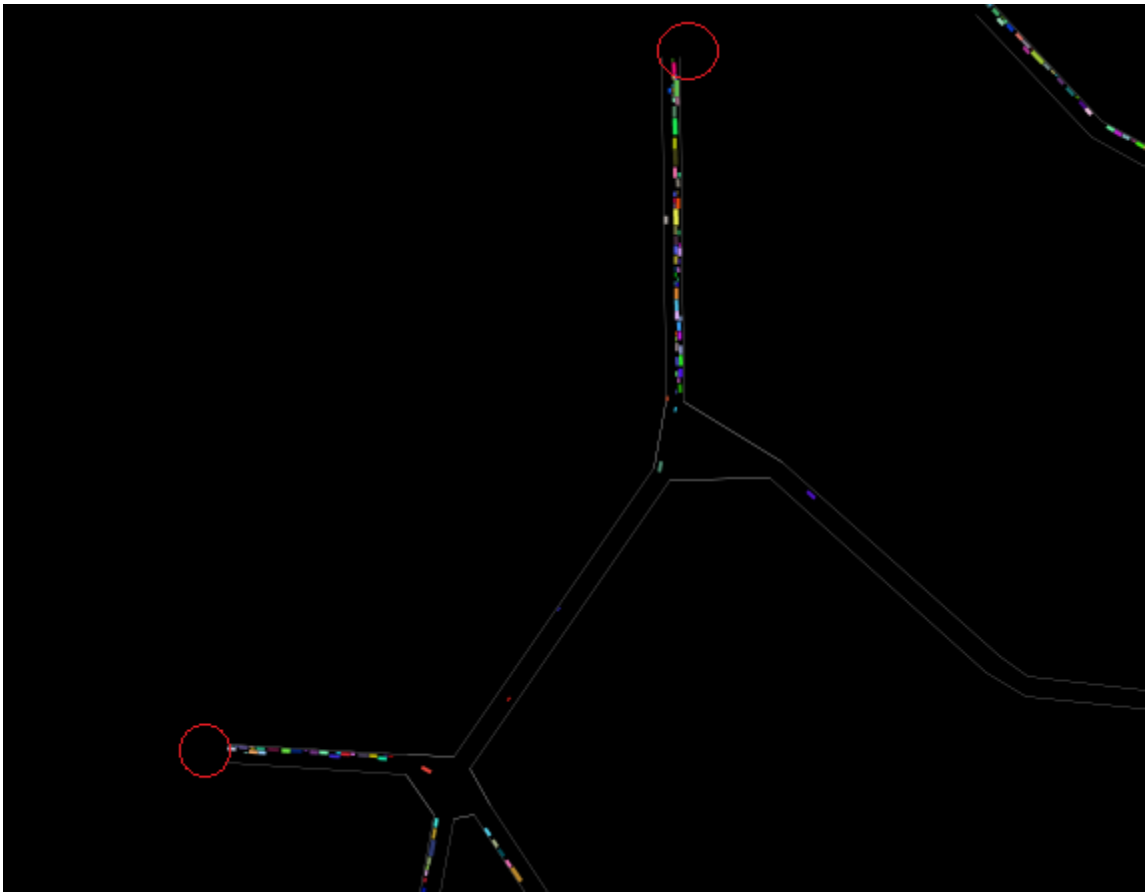
Node.txt

The first line of this file denotes number of nodes/junctions(n) the network has. Then there are n lines for each node/junction. The format of each line is:

node_id x y link_list

x y: Coordinate of the node/junction. No exact point is needed. Any point in the junction will suffice. This is only needed for the nodes/junctions that generate vehicles (red circles). The internal nodes/junctions can be kept 0 0.

link_list: The links that are connected with that node.



Link.txt

The first line of this file denotes number of links(m) the network has. Along with the description of the links there is also the description of its constituent segments. The format is like this:

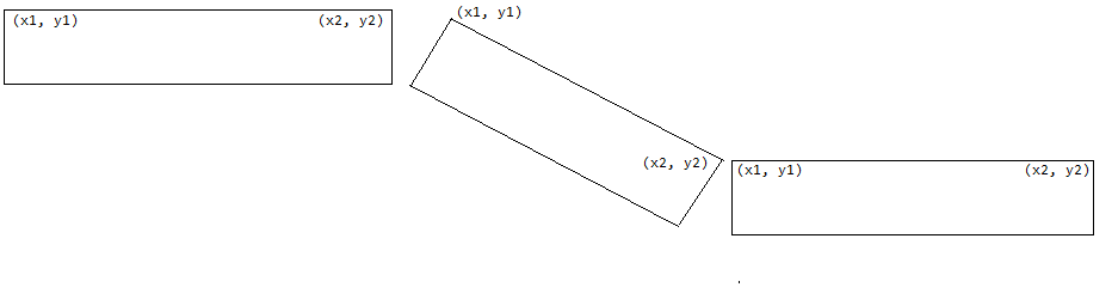
link_id node_id_1 node_id_2 number_of_segments(s)

segment_id_1 x_1 y_1 x_2 y_2 width

segment_id_2 x_1 y_1 x_2 y_2 width

...

segment_id_s x_1 y_1 x_2 y_2 width



Demand.txt

It has the number of entries(d) as the first line. Then there are d entries each of which gives the number of vehicles per hour between a source node and a destination node. The format is:

source_node destination_node vehicles/hour

Path.txt

It also has the number of entries(p) as the first line. Then p entries follows each of which give a path from the start node to end node as a list of links.

start_node end_node link_list

link_list: The constituent links of that path.

Here, d=p. And the demand of a path between two nodes is set based on the distance of that path. The longer the path, the less is the demand for that path between a pair of node.

JAVA Classes

The road network is input as a graph consisting nodes for junctions and roads for links. Each road is divided into segments to allow curved ones. Each segment is further divided into strips. Strips are the unit of measurement along the width of a road.

AccidentCountPlot.java

This java class uses the java library **jmathplot.jar**. First, a Plot2DPanel object is created. The values to plot are first stored in a double array and passed to the addBarPlot() function of the Plot2DPanel object. After the simulation ends an object of this class is called from the actionPerformed() function of the DhakaSimPanel object to output 'how many accidents in each of the segments?'

AccidentCountPlot
+plotPanel: Plot2DPanel
~AccidentCountPlot(double[] x)

Demand.java

The sole purpose of this class is to hold the values of the demand.txt and path.txt files. Objects of these classes are called from the readDemand() and readPath() functions of the DhakaSimPanel object and stored in an ArrayList<Demand> called demandList and an ArrayList<Path> called pathList. Then the path information is merged with demand information by addDemandToPath().

Demand
-source: int -destination: int -demand: int -pathList: ArrayList<Path>
~Demand(int source, int destination, int demand) +getSource(): int +setSource(int source): void +getDestination: int +setDestination(int destination): void +getDemand(): int +setDemand(int demand): void +getPath(int index): Path +addPath(Path path): void +getNumberOfPaths(): int

DhakaSim.java

This class holds nothing but the main function. It creates an instance of DhakaSimFrame class.

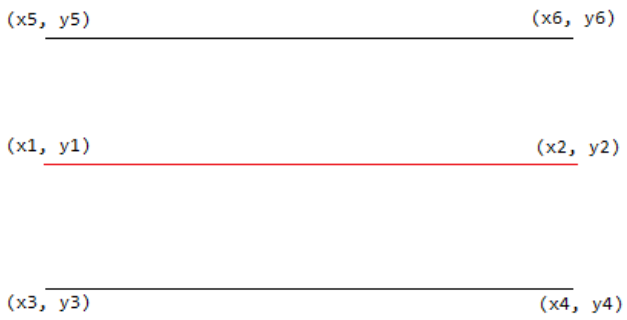
Link.java

This class holds the links after reading it from link.txt. An instance of this class is called from the readLink() function of DhakaSimPanel and stored into an ArrayList<Link> called linkList. The draw() function of Link class simply calls the draw() function of each of its segment. And the other classes are mostly getters and setter. The ids are only used in readLink() function. From there on out, we use the index. Where index is not available we use getIndexFromId() to get the index.

Link
-index: int -id: int -upNode: int -downNode: int -segmentList: ArrayList
~Link(int index, int id, int upNode, int downNode) +getIndex(): int +setIndex(int index): void +getId(): int +setId(int id): void +getUpNode(): int +setUpNode(int upNode): void +getDownNode(): int +setDownNode(int downNode): void +getSegment(int index): Segment +addSegment(Segment segment): void +getNumberOfSegments(): int +draw(Graphics2D g2d): void

Misc.java

The misc.java class has all the miscellaneous functions. One of the most important part of this project is finding the four perpendicular points which are d distance apart from the two endpoints of a line.



The functions **returnX3(), returnY3(), returnX4(), returnY4(), returnX5(), returnY5(), returnX6(), returnY6()** take the input x1, y1, x2, y2 and distance d and return the relevant values.

Calculation of **returnX3()** is shown here:

Let, the vector from (x1, y1) to (x2, y2) be (u, v).

$$u = x2 - x1$$

$$v = y2 - y1$$

Say, the counterclockwise perpendicular vector to (u, v) is (s, t).

$$s = -v$$

$$t = u$$

Then, the perpendicular point d distance apart from (x1, y1) is:

$$x3 = x1 + d \times u$$

$$y3 = y1 + d \times v$$

Remember, the Y axis in jFrame is upside down. So, the (x3, y3) point will be below the line (x1, y1) to (x2, y2) rather than above.

getDistance(): Another heavily used function is the distance between two points. The get distance function take two points and return their distance.

$$d = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

doIntersect(): This function determines if two lines intersect at an any point other than their end points.

shareAnyPoint(): This function determines if two lines share any end point.

isPointOnTheLine(): This function determines if a given point is on a line or not.

Misc
+returnX3(): double +returnY3(): double +returnX4(): double +returnY4(): double +returnX5(): double +returnY5(): double +returnX6(): double +returnY6(): double +doIntersect():boolean +shareAnyPoint(): boolean +isPointOnTheLine(): boolean

Path.java

This class is used to store paths from the Path.txt file. Instances of this class is used to store paths in an ArrayList<Path> from readPath() and addPathToDemand() function of DhakaSimPanel. Demand class also has an ArrayList<Path> called pathList which stores possible paths between two nodes. Most of the functions are only getters and setters.

Path
-source: int -destination: int -demand: int -pathList: ArrayList<Path>
~Path(int source, int destination) +getSource(): int +setSource(int source): void +getDestination(): int +setDestination(int destination): void +getLink(int index): int +addLink(int link): void +getNumberOfLinks(): int

SensorVehicleAvgSpeedPlot.java

This java class uses the library **jmathplot.jar**. Each segment has a point where sensor is placed. For now, it is hard coded to be at the middle of the segment. At this sensor, speed of each vehicle is stored and then the average value for each segment is shown.

SensorVehicleAvgSpeedPlot
+plotPanel: Plot2DPanel
~SensorVehicleAvgSpeedPlot(double[] x)

SensorVehicleCountPlot.java

This java class also uses **jmathplot.jar**. The average number of vehicle passed over the sensor of each segment is shown.

SensorVehicleCountPlot
+plotPanel: Plot2DPanel
~SensorVehicleCountPlot(double[] x)

Vector.java

This class implements 2D vector. It has a x component and a y component. There are also functions available for getting dot product of two vectors, getting the magnitude of a vector or getting the square of magnitude of a vector.

Vector
+x: double +y: double
~Vector(double x, double y) +dotProduct(Vector v): double +magnitude(): double +magnitudeSquare(): double

dotProduct(): This function gives the dot product of two vectors.

$$d = u \,.\, v$$

magnitude(): This function gives the magnitude of a vector.

$$d = \sqrt{x^2 + y^2}$$

magnitudeSquare(): This function just returns the square of magnitude.