

# User Guide

## MobiSim v3 – A Framework to Manage Mobility Models

Copyright © 2006 – 2013

by:

Masoud Moshref Javadi

E-Mail: [masood.moshref.j@gmail.com](mailto:masood.moshref.j@gmail.com)

Website: <http://www.masoudmoshref.com/>

Project website:

- [http://www.masoudmoshref.com/old/myworks/documentpages/mobility\\_simulator.htm](http://www.masoudmoshref.com/old/myworks/documentpages/mobility_simulator.htm)
- <http://sourceforge.net/projects/mobisim/>

## GNU General Public License

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>

## NOTICE

Note that you must cite the following paper in any paper or document which uses this program:

**"Mousavi, S. M., Rabiee, H. R., Moshref, M., and Dabirmoghaddam, A. 2007. MobiSim: A Framework for Simulation of Mobility Models in Mobile Ad-Hoc Networks. In Proceedings of the Third IEEE international Conference on Wireless and Mobile Computing, Networking and Communications (October 08 - 10, 2007). WIMOB. IEEE Computer Society, Washington, DC, 82. "**

## Table of Contents

1 Introduction .....	5
2 Installation .....	5
2-1 Source files .....	5
2-2 Binary files.....	5
2-3 3D Resimulator .....	5
3 How to work with modules .....	6
3-1 Main menu .....	7
3-2 Mobility generator .....	7
3-2-1 Graphical simulation .....	8
3-2-2 Map Editor.....	12
3-2-3 Scenario configuration .....	13
3-2-4 Batch simulation.....	14
3-2-5 Trace visualization.....	15
3-2-6 3D visualization.....	16
3-3 Evaluation .....	17
3-4 Excel diagram creator .....	19
3-5 Mobility model recognizer.....	23
3-5-1 Model recognizer algorithms.....	23
3-6 Evaluating factors in model recognizer algorithms .....	26
3-6-1 Accuracy.....	27
3-6-2 Membership per class.....	27
3-6-3 Right membership average.....	27
4 Mobility models .....	27
4-1 General maps .....	27
4-1-1 Obstacle map .....	28
4-2 Range algorithms .....	29
4-2-1 Fixed.....	29
4-2-2 RNG.....	29
4-2-3 MST .....	29
4-2-4 Null.....	29
4-3 Location initializers .....	30
4-3-1 Fixed Initializer .....	30

4-3-2 Random Initializer .....	30
4-3-3 Null Initializer .....	30
4-4 Random models .....	30
4-4-1 Random Waypoint Model .....	30
4-4-2 Random Direction .....	31
4-4-3 Random Walk .....	32
4-4-4 Levy Walk Model .....	32
4-4-5 Tortoise Model .....	33
4-5 Manhattan models .....	36
4-5-1 Freeway .....	36
4-5-2 Manhattan .....	38
4-6 Temporal dependent models .....	40
4-6-1 Gauss-Markov .....	40
4-6-2 Probabilistic Random Walk .....	41
4-6-3 Exponential correlated model .....	43
4-7 Group models .....	43
4-7-1 Nomadic .....	43
4-7-2 Pursue .....	46
4-7-3 String .....	47
4-7-4 Row .....	47
4-8 Complex Models .....	48
4-8-1 Multi Model .....	48
4-8-2 ThreeDizer Model .....	50
4-9 File Model .....	51
5 Evaluators .....	51
5-1 Average lifetime .....	51
5-2 Average power/range .....	51
5-3 Average distance .....	51
5-4 Clustering coefficient .....	52
5-5 Disconnection ratio .....	52
5-6 Interference .....	52
5-7 Link evaluator .....	52
5-8 Network diameter .....	52
5-9 Node degree .....	53

5-10	Number of disconnection.....	53
5-11	Relative speed.....	53
5-12	Repetitive behavior.....	53
5-13	Spatial dependency.....	53
5-14	Temporal dependency.....	54
5-15	Transition evaluator.....	54
6	File formats.....	54
6-1	Mobility traces.....	54
6-1-1	Plain text.....	54
6-1-2	NS2.....	55
6-1-3	XML.....	55
6-2	Power traces.....	55
6-3	Evaluation file.....	56
7	How to extend?.....	56
7-1	Parameterable Hierarchy.....	56
7-2	Add a new mobility model.....	60
7-3	Add a new map.....	60
7-4	Add a new trace output/input.....	61
7-5	Add a new evaluator.....	61
7-6	Add a new model classifier algorithm.....	61
7-7	Add a new process.....	61
8	References.....	62
9	Appendix 1: Prerequisite installation.....	63
9-1	Java.....	63
9-1-1	Windows.....	63
9-1-2	Linux.....	64
9-2	Ant.....	65
9-3	Java3D.....	65
9-3-1	Windows.....	65
9-3-2	Linux.....	65

Document info:

<b>Author</b>	<b>Date</b>	<b>Changes</b>
Masoud Moshref Javadi	2008/9	First release
Masoud Moshref Javadi	2009/6	ThreeD model, Map Editor, Levy Walk, General Maps, Prerequisite installation appendix
Masoud Moshref Javadi	2010/6	Extension documentation, Fuzzy, Gaussian and SVD classifiers
Masoud Moshref Javadi	2011/1	New UI update, Obstacle map, Transition Evaluator
Masoud Moshref Javadi	2013/7	Release whatever implemented

## 1 Introduction

MobiSim is a mobility simulator application, composed of many modules, using a *pluggable* architecture and a *form generator* in order to help researchers in the field of MANET (Mobile Ad-hoc NETWORKS) deal with mobility in these networks. User can create “*scenarios*”, simulate them in *graphical* or *batch* modes, resimulate the traces, *evaluate* measures on the traces, create *Excel diagrams* from the evaluations, and finally test mobility model recognition algorithms on the traces. It supports plain text, NS format, and xml files for traces and contains many mobility models and maps with obstacles, which can be *mixed together* and configured *graphically*. All of these can be *extended* with an XML configuration file without any need to create additional GUI thanks to the *form generator* and *Parameter framework*.

I developed the very first version of this software for my BSc final project under supervision of Dr. Rabiee and guidance from Mr. Mousavi in Sharif University of Technology in 2007. Besides, I want to thank Mr. Dabirmoghaddam for his codes in network analyzer tool. For version 3, I tried to gather whatever I have developed by now as my field of research may not meet mobility in future. While this software is an AS IS one, I will appreciate if you have any idea to improve it. Just feel free to contact me at [masood.moshref.j@gmail.com](mailto:masood.moshref.j@gmail.com).

## 2 Installation

To start the application, you need the JAVA Runtime Environment (v1.5 or later). If you do not know how to setup java in Windows or Linux see 9-1 .

### 2-1 Source files

The framework source files are packaged as a zip-archive<sup>1</sup>. After extracting it in a new folder, you can use ANT to build the sources (See 8-2 to find out how to install ANT). Change to the directory with framework’s source files and type (in command prompt):

```
ant
```

All source files will be compiled and packaged into a new folder named “dist”. Now change to the “dist” folder and use the directions in the next section, Binary files, to run the framework. Alternatively, use the included .idea folder project definition if you have access to IntelliJ IDEA IDE<sup>2</sup>.

### 2-2 Binary files

The framework’s binary files are compressed to a jar-archive. To launch the framework, change to the directory containing framework’s files and type (in command prompt):

```
java -Xms128M -Xmx512M -jar Core.jar
```

A user interface containing some buttons for each module appears.

### 2-3 3D Resimulator

I have packaged 3D resimulator module separately because it needs an additional prerequisite, Java3D. Besides, not all users want to work with this package.

<sup>1</sup> If the downloaded file has “src” in its name, it is the source package.

<sup>2</sup> They have introduced a free community edition! Enjoy it.

You need Java3D in order to run or compile codes in this module. Java3D helps us render 3D objects and move in the 3D environment. See 9-3 for installing Java3D.

If you have binary files for 3D Resimulator module, just copy its .jar file beside other jar files in the project root folder. However, if you downloaded its source files, copy the source folder in the project root folder. As a result, you should have a jar file “ThreeD.jar” or a folder named “ThreeD” beside the “resource” folder. In order to run 3D Resimulator, change to the project root directory and type:

```
java -jar ThreeD.jar
```

Compiling the source module is just like before. Use ant!

### 3 How to work with modules

In this section, I introduce how to work with the implemented modules. Figure 1 shows the process of using MobiSim. You can create mobility traces using “Mobility Generator” module. Evaluate these traces or traces got from the real-world using various evaluators. Besides, it is possible to use the traces in current network simulators and get statistics (evaluations) that need a more sophisticated network simulation. Model Classifier has many algorithms including fuzzy ones to classify the evaluation file in models. Using file based connection between modules, there are many possibilities of using this software. For example, you can use real-world traces and create evaluation files based on them. Then, you can classify it on various known models.

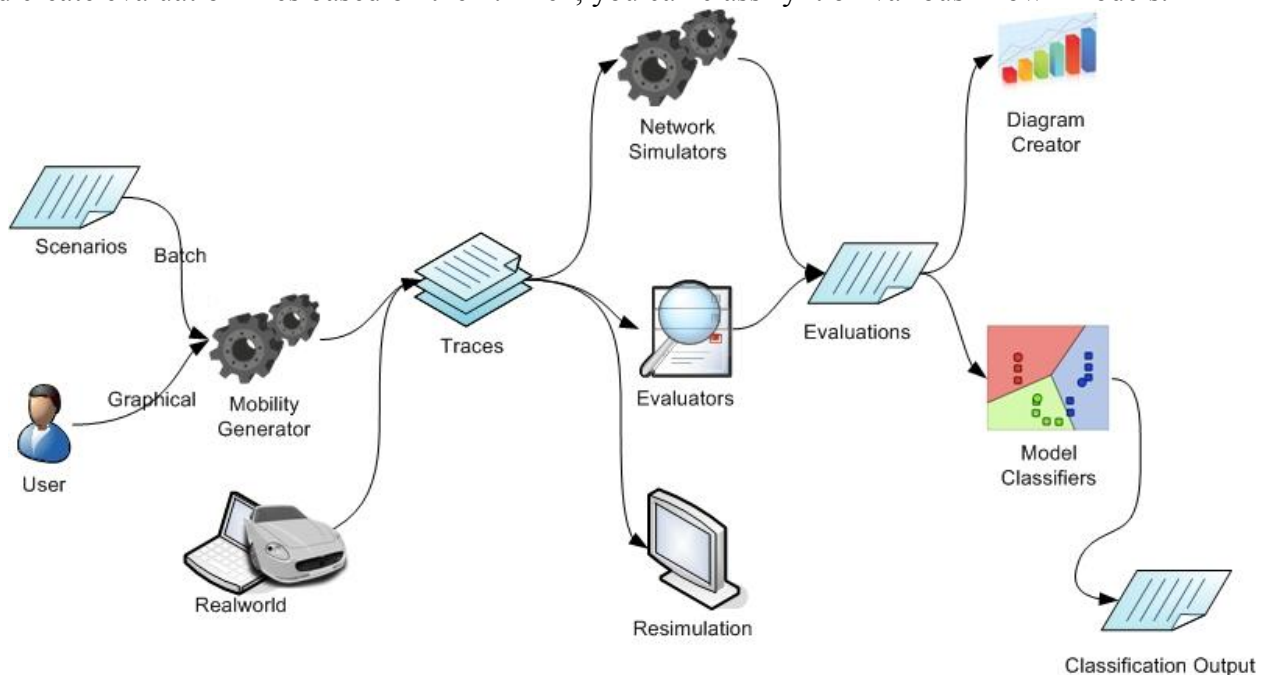


Figure 1: The workflow of using MobiSim

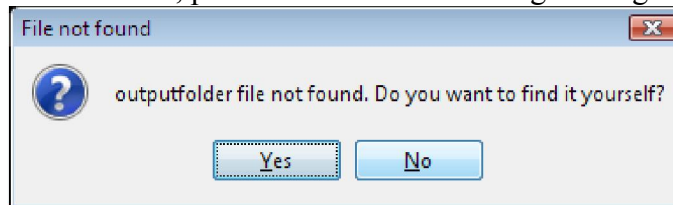
Figure 2 presents the main window of MobiSim, each tab of which represents a step in the simulation workflow.





Figure 2: The main window of MobiSim

Before talking about each module, please note to the following message:



This message, which says the application could not find a file or a folder, means that the user himself should find or create it. The application, then, saves the user choice in a file called “config.properties” in the root folder (To change it see [Preferences menu](#)). Therefore, it will not ask it again for future usages. Note that the file or folder path will be saved relative to the application root folder, so it is not usually necessary to set it again when you move the project folder.

### 3-1 Main menu

Preferences menu is the only menu here that needs to be discussed. It opens the preferences window which is actually a tabular view of the “config.properties” file. There are some situations that you may want to change the default referenced files for each internal module and this window helps you. However, the most important properties here is the “seed” property. If you set it to 0 value, simulations will take a random seed for each run, but you can set it to a number to make a simulation scenario repeatable. This feature really helps when you want to add a new mobility model or track how nodes move.

### 3-2 Mobility generator

This module manages the generation of mobility traces. It contains many models and their related maps. There are two modes of simulation: Graphical and Batch. The former uses graphical fields to configure the simulation parameters during simulation, and the latter uses a scenario to generate mobility traces without any graphical presentation. A *scenario* is a package of configurations with certain variable parameter(s) in order to model the effects of variation of parameters on characteristics of generated traces.

### 3-2-1 Graphical simulation

The graphical simulation implemented to observe the behavior of each model and effect of each parameter on it. Use the “Batch” option (3-2-4 ) to generate mobility traces rapidly.

You can see the following user interface when you select the graphical simulation in the mobility generator module:

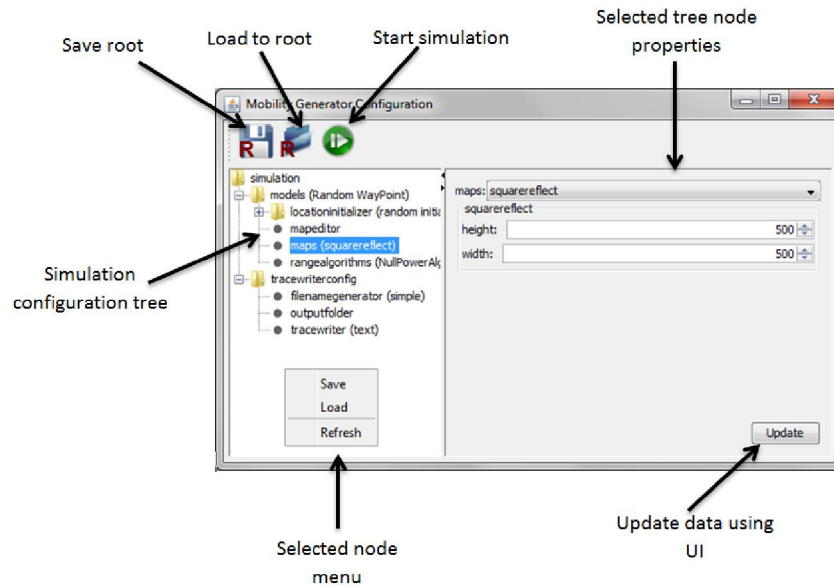


Figure 3: Graphical user interface for simulation configuration

This is the new UI for configuring any configurable object. I will discuss more about each part of the user interface below:

- **Simulation configuration tree:** In MobiSim the objects make a tree to keep the configuration of a simulation scenario. By selecting each node in the tree, its properties will be presented in the right side. Do not forget to press the “Update” button after the changes in this panel.
- **Toolbar:** There were sometimes in the previous versions that you might configure a sophisticated simulation scenario and tested it in the graphical module but there were no option to save it. Now you can save/load the simulation configuration using the toolbar buttons. Using this feature, you can easily by pass the default values.
- **Selected node menu:** There are some situations that you do not want to save/load whole simulation configuration but just part of it. For example, you want to test different models with one specific sophisticated map. As the map is a child of the model you should configure the map for every model in previous versions. However, now, you can save the map in a file and load it whenever you need it. Using this menu (which is available by right mouse click on the configuration tree) you can save/load almost any node in the tree representation.

Note that the node will not change if you load an inappropriate configuration file into a node. For example, you cannot load a configuration for a map into a model! Or a configuration for a RandomWayPoint model into Manhattan model.

When you start a simulation the following UI will show:

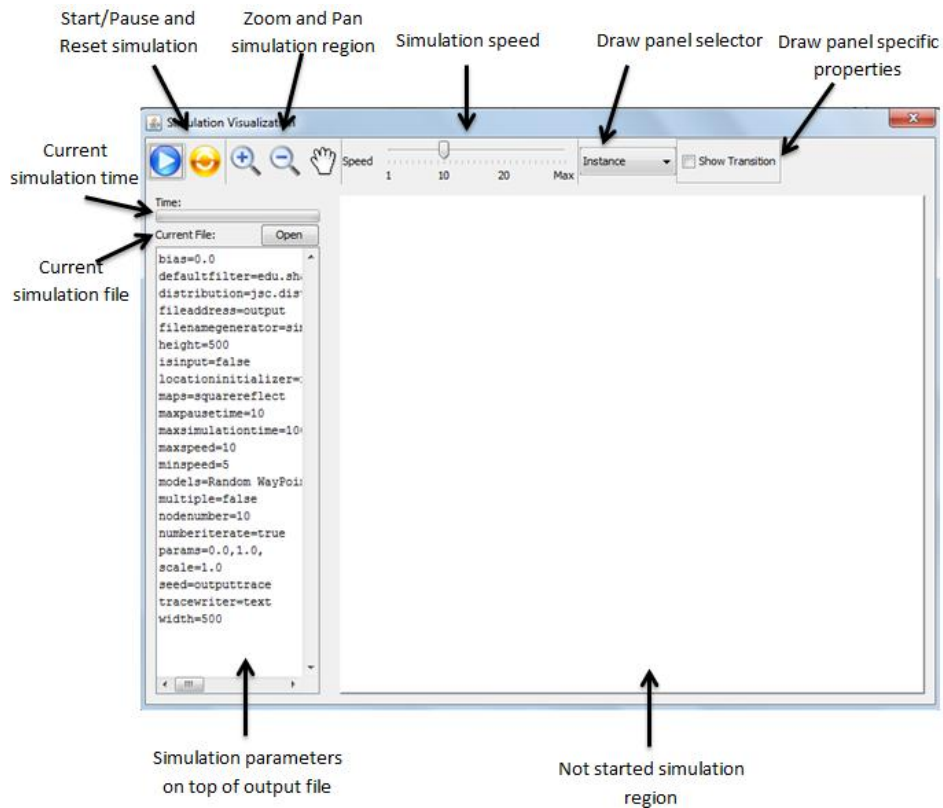


Figure 4: Graphical simulation window

You can pause, start or reset simulation for a specific configuration with different speeds using the toolbar in the graphical simulation window shown in Figure 4. Note that the current file name is generated by the “`tracewriterconfig>filenamegenerator`” under the simulation object and may create different files when you reset the current simulation. Then there is the simulation parameters list in the right of the window which is a brief description of the parameters for the current simulation.

“Draw panel” is a new concept introduced in version 3 which handles different way of presenting a simulation to the user. You can change the current draw panel during simulation or change its settings. Each draw panel may have specific configuration which is available in the toolbar. There are three types of draw panels:

1. **Instance:**

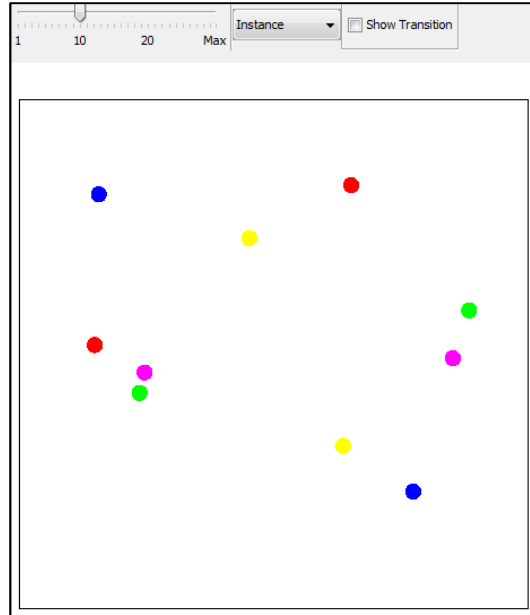


Figure 5: Instance draw panel draws each node position only in one instance

## 2. Footprint:

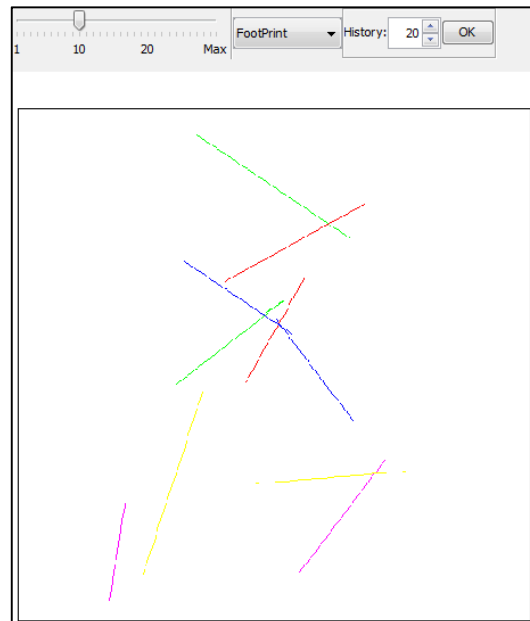


Figure 6: Footprint draw-panel prints the path a node traveled from "History" timestep before

## 3. Fade Footprint:

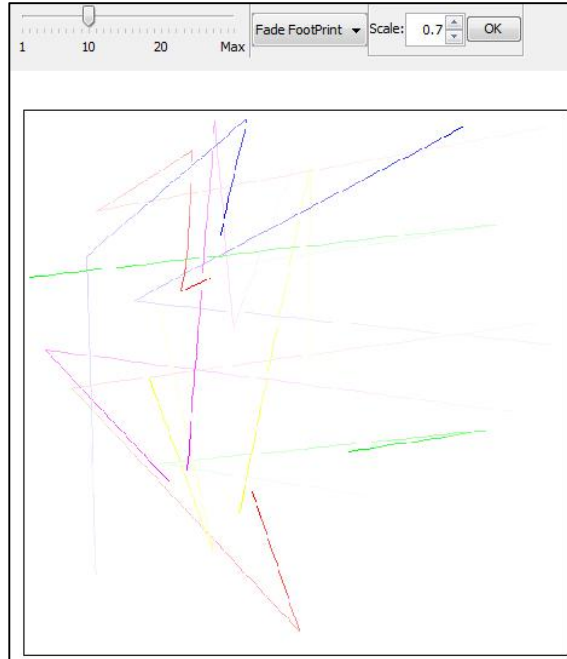


Figure 7: Fade FootPrint draws the path a node travels by fading each timestep using "Scale" parameter

#### 4. Link Panel:

This panel shows the (bidirectional) links between nodes based on the range parameter. It can also draw a disk around nodes representing their ranges and show directional connections.

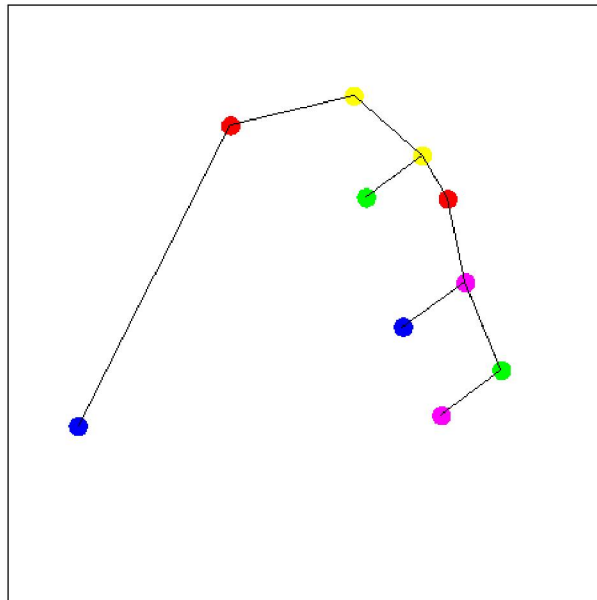
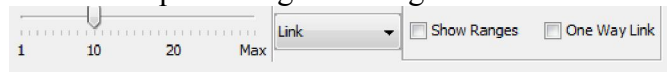


Figure 8: Link draw-panel draws links between nodes

### 3-2-2 Map Editor

Map editor is a facilitating utility to enable users design maps easier. Each model has a map editor which is accessible by clicking on “Show” button presented as a parameter in simulation parameters panel. Each map has some handles that can be dragged in the graphical part or be set in the parameter configuration part. Maps specific handles are described in Section 4-1 .

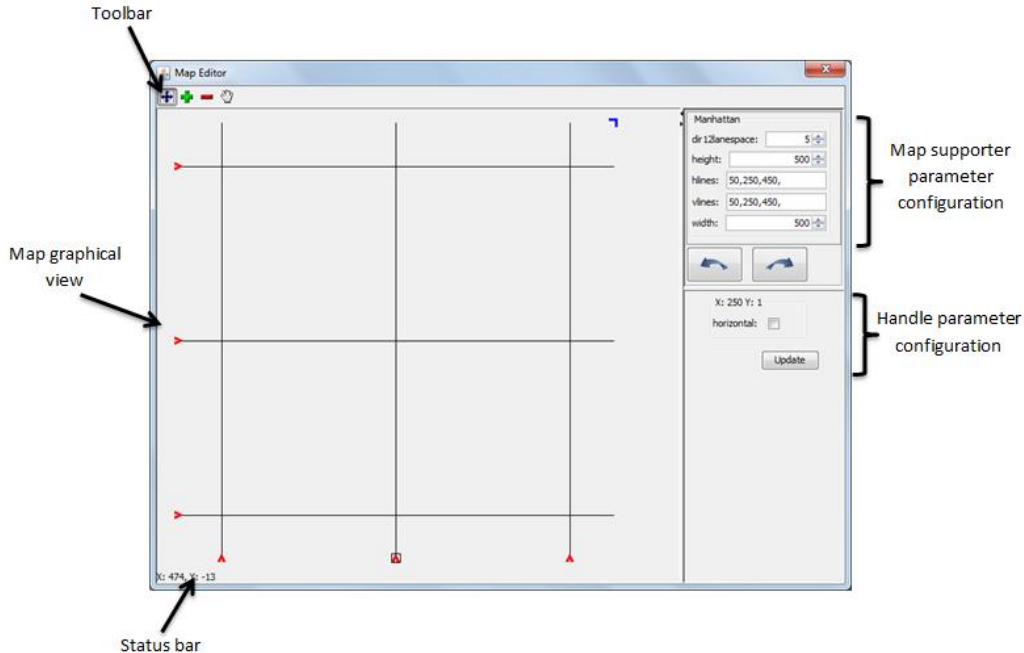








Figure 9: A sample Map Editor window

Map editor window contains five parts:

- **Toolbar:** You can move the graphical view over the map using the “Move Page”  button by activating it and dragging on the map graphical view. You can move most of the handles presented on the map. Click on “Move”  button to enable move state. If the map accepts variable number of handles, “Add”  and “Remove”  buttons will be enabled. Click on the map to create a new handle after activating the add button. Some maps are more complex and show additional controls in the toolbar. For example, they may let you select the internal map to which you wanted to add a handle.
- **Map supporter parameter configuration:** Map supporter can be a “Map” or a “Model”. In the left part of the window, you can see the map supporter’s configuration panel, which lets you set/see parameters’ value accurately. There are two buttons under the panel. “Update”  button updates the “map graphical view” using values in these panels. “Fill”  button fills the parameter configuration panel and objects in the memory from graphical handles. You must click on the fill button before closing the window to save your changes.
- **Handle parameter configuration:** Some map supporters may create handles that have additional parameters; for example, when the sequence of handles is important,

they will have a sequence parameter. Click on the “Update” button  after changing the value.

- **Map graphical view:** This is a canvas that draws the handles and the map. You can adjust map’s parameters by dragging its handles. Note that some handles cannot be moved, but their parameters can be adjusted using handle parameter configuration. You cannot move handle wherever you want because the map supporter validates every movement. There are some situations that you cannot move any handle and the status bar shows you an error. In these situations, you should solve the problem using the configuration panel. Read the model/map constraints to find out what the problem is.
- **Status bar:** By default, the status bar shows the x and y of the current mouse position, but while you are dragging a handle, it shows the current position for the handle. If the model/map does not validate the new position of the dragging handle, it will alert you about that.

Position of a handle is invalid, try moving it or set it using the config panel in the right

Figure 10: Alert in Map Editor status bar

### 3-2-3 Scenario configuration

“Scenario” is a configuration of parameters plus number of runs and some variables. A variable is an integer or double parameter which has an initial value, a step value and a number of changes. Scenarios are saved in a file, “Config” file, and will be used in a “batch simulation”. It can be used beside the traces for documentation. Besides the configuration, remember to change the title field of scenario, and the seed field for the filename, too. Most of the times, you want to check some models in a specific situation or check one model in different situations, so there are many common configurations among scenarios. Therefore, the save and load function for a tree node can be used to clone a configuration.

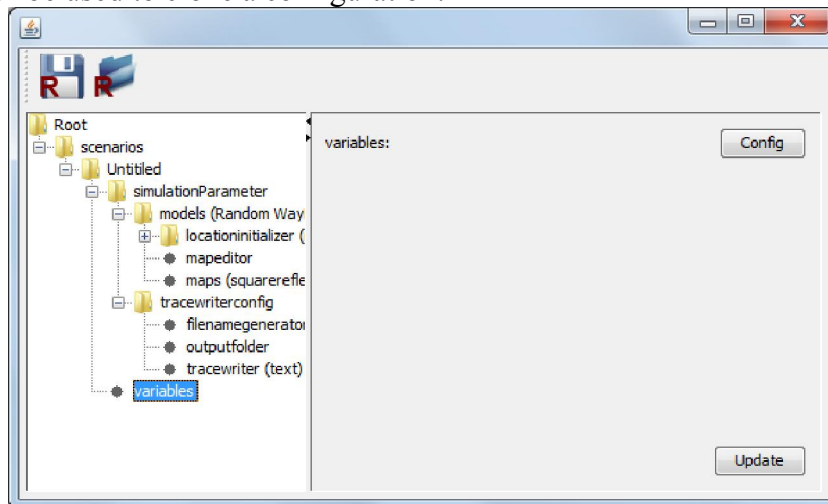


Figure 11: Main window for a scenario configuration



Figure 12: A sample list of detected variables in the current scenario

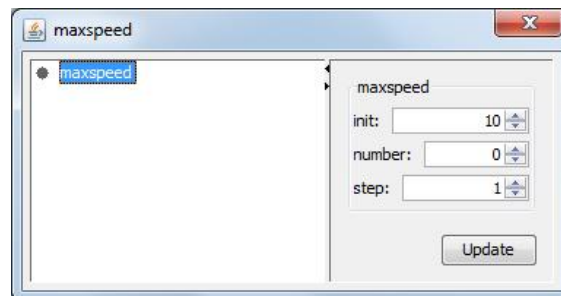


Figure 13: Variable configuration window

### 3-2-4 Batch simulation

Batch simulation is for generating mobility traces rapidly. It uses a Config file and creates a trace file for each unique composition of scenario, variable(s) number, and run number as follows:

```
<run number>-<variable1.value>-<variable2.value>...<variablen.value>
<filenamegenerator.seed>.<tracewriter.postfix>
```

Note that the dots in variables' value will be replaced by a comma. When you select the Config file and click on the next button, a progress window, in which you can see the output files and elapsed time for each one, will appear like Figure 14.



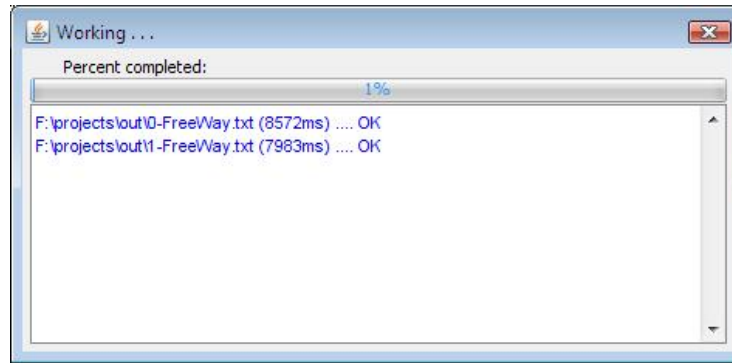


Figure 14: Progress window for batch simulation

You may see red lines in the progress window if an error occurs during a run (Figure 15). In this situation, the trace files for faulty runs are empty, but others are OK. See the command line to find out the cause of error (usually bad configuration for initializing nodes position).

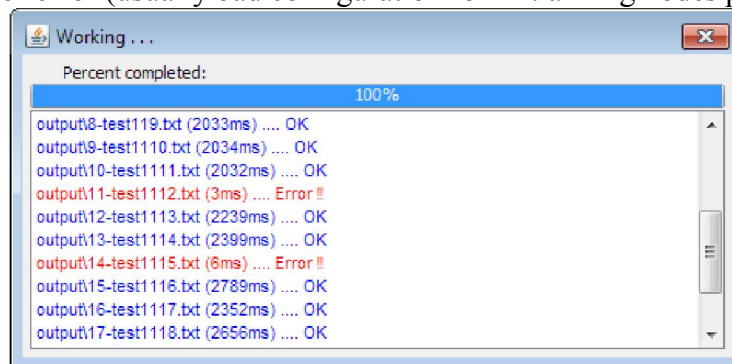


Figure 15: Progress window with some errors

### 3-2-5 Trace visualization

This module is implemented in order to resimulate (visualize) the movement of nodes in a trace file. The features of this module are very similar to the graphical simulation window<sup>3</sup>:

<sup>3</sup> Actually it uses File Model

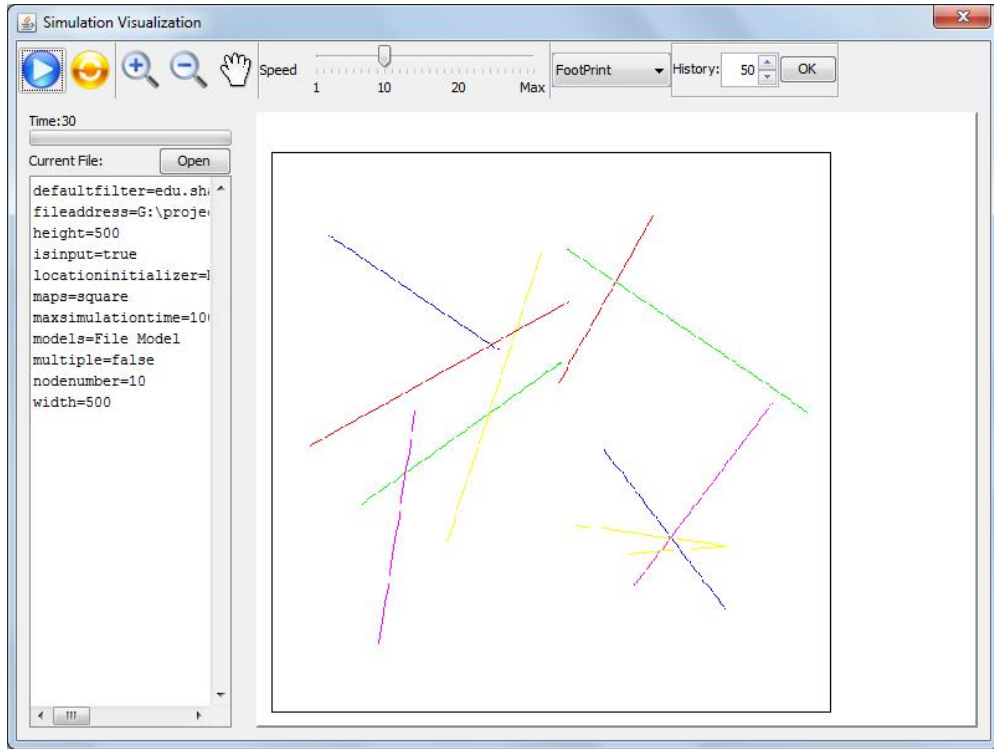


Figure 16: Simulation visualization window

### 3-2-6 3D visualization

3D Visualization uses java3d package to create a 3D environment. Therefore, you should have installed the package. Initially, you must select the trace file which contains Z positions for nodes and the 3D map configuration file exported in the simulation process.

Click on “Start” menu item to run the simulation. You can move forward and backward using up and down arrow keys, turn left and right using left and right arrow keys, and turn up and down using page up and page down keys. Use home key to reset the camera position. The following paragraphs explain how the ground map created using the user-defined points.

The java3d package gets a list of strips, triangulates them, and puts the triangles together to create a surface. My algorithm tries to create these strips such that the resulting triangles would be similar to what the 3D model is used in mobility generation. In each block, created based on the map configuration, the implemented algorithm tries to walk clockwise on the block and create strips. Figure 17 (a) shows the clockwise walk and Figure 17 (b and c) presents some sample strips. As the strip creation algorithm is a complex one, I will not present it here. Refer to the development documentation for more details.

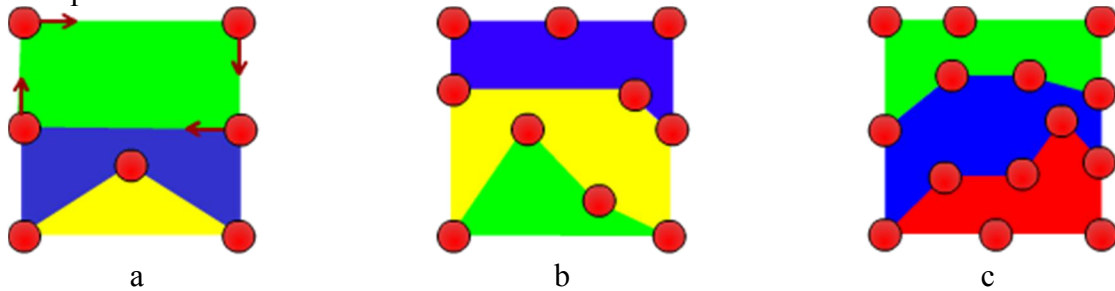


Figure 17: Ground Strips Samples

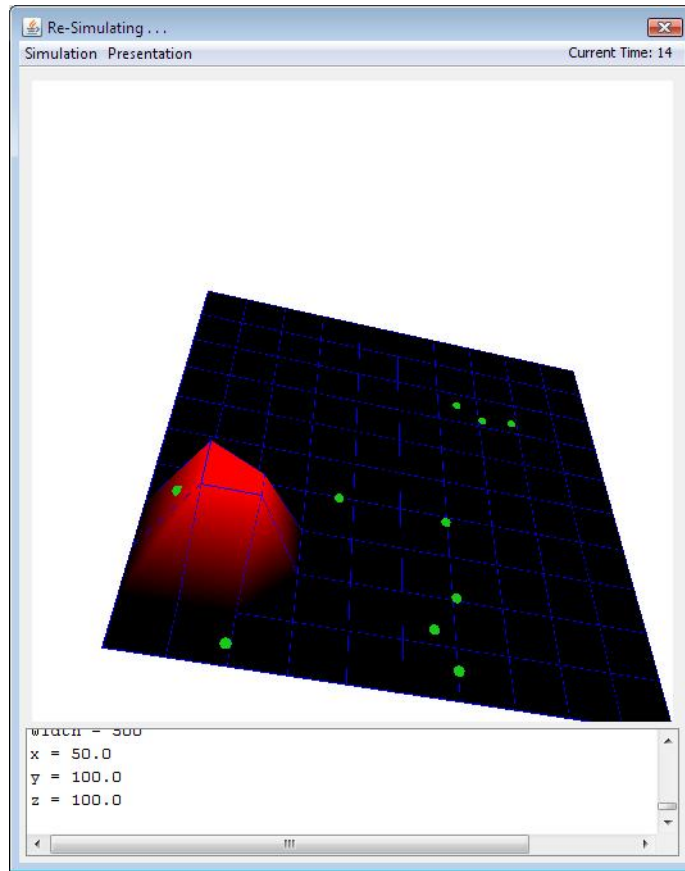


Figure 18: A preview of 3D resimulator window

### 3-3 Evaluation

This module uses pluggable classes, named evaluators, to measure a quantity or quantities. There are two types of evaluators in this section: evaluators which need power values and which do not. Power values may be set using “linear power generator” or other software implementing more complex algorithms such as TopoSim or OMNET++ simulator. Every evaluator may have some parameters, accessible from “Config Evaluators” button. The following steps explain how to use the dialog to evaluate traces. Section 4-8 explains evaluators.

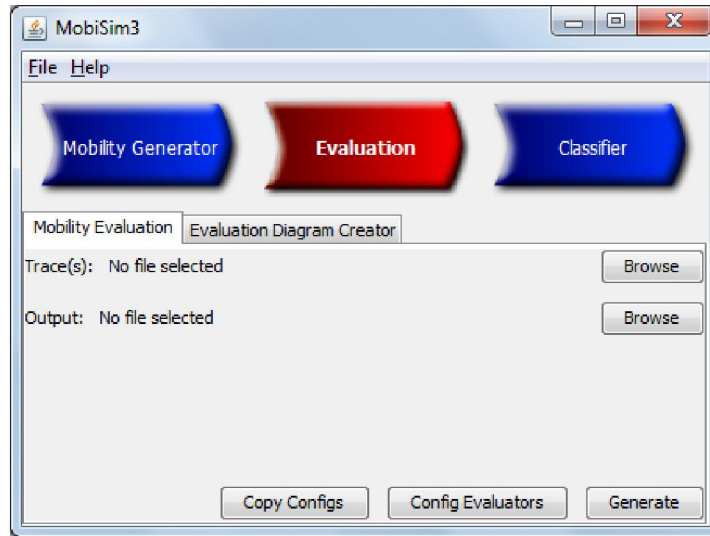


Figure 19: Evaluation dialog

1. Select the traces that you want to evaluate.
2. Select the output file.
3. Use “Copy Configs” button to copy any “simulation parameter” which you want to copy from traces to the output file (see Section 6 for simulation parameter). This feature implemented to copy some hints about the parameters of the simulation to the evaluations file especially for diagram creation (see 3-4 ) and model recognition (see 3-5 )
4. Use “Config Evaluators” button to select evaluators and configure their parameters. The config window is pictured in Figure 20. While each evaluator may have its own parameters, all of them have *sampletime* and *sortpriority*. *Sampletime* determines how often the evaluation must be run. Using *sortpriority* parameter, you can sort the output file according to the value of the selected evaluator. Note that positive values make ascending order and negative ones make descending order. Higher *sortpriority* absolute values mean higher priority in the sorting algorithm. The priority among evaluators with equal *sortpriority* absolute value is not determined.
5. Click the “Generate” button.

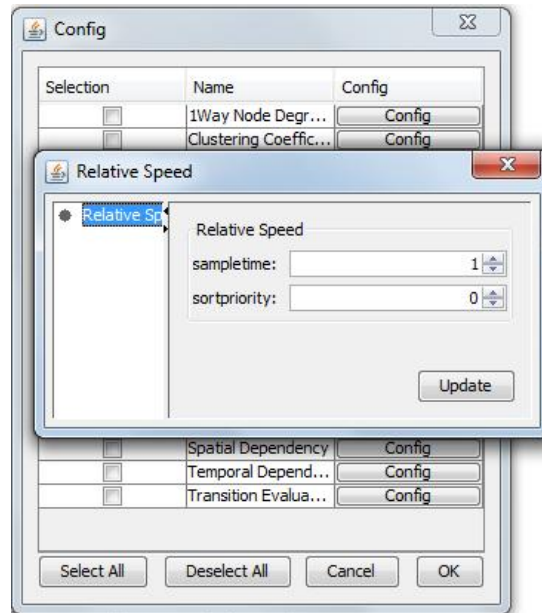


Figure 20: Configure Evaluators dialog

### 3-4 Excel diagram creator

To observe the behavior of mobility models, see the result of evaluations better, and write any paper, we need to have diagrams. This module loads an evaluation file and creates Microsoft Excel file, containing a macro to draw diagrams.

I explain the process using an example: Supposed that I want to observe the effect of minspeed and maxspeed parameter on the relativespeed evaluator for each model. As a result, the models will be on rows and the combination of these parameters constructs the columns.)

1. Before starting creating the diagram, prepare your evaluation files. Note that the evaluation file could have more information than just evaluators because of the “Copy Configs” feature. For example, to create the output of this process I used the dialog to copy model, minspeed and maxspeed from all traces to the evaluation file.

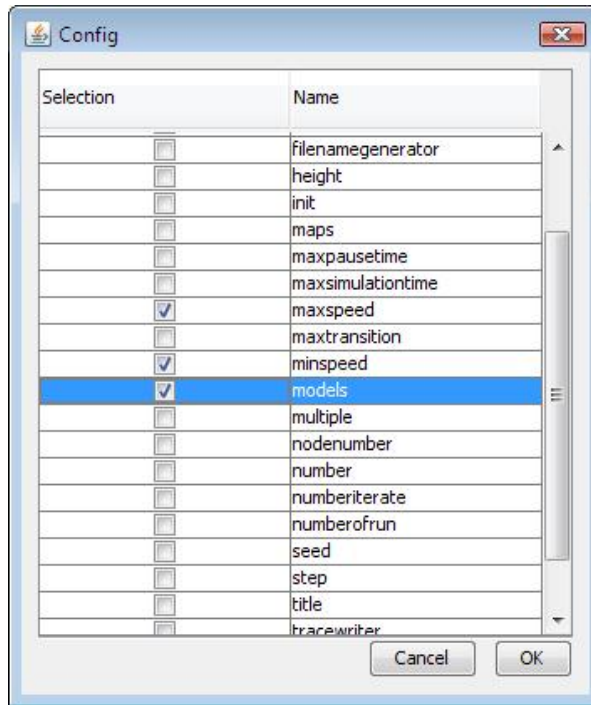


Figure 21: Copied parameters from traces to the evaluation file to create the following diagram

Therefore, the evaluation file has a first line like this:

FileName	Temporal	Dependency	Spatial	Dependency	RelativeSpeed	maxspeed	minspeed	models
0-10,0-5,0 Markov1.txt	0.28	0.03	1.76	10	5	Markov		
0-10,0-5,0 outputtracel.txt	0.22	-0.01	4.38	10	5	Manhattan		
0-10,0-5,0 RandomDirection1.txt	0.48	-0.01	2.27	10	5	RandomDirection		
0-10,0-5,0 RandomWaypoint1.txt	0.39	0	2.52	10	5	RandomWayPoint		
0-10,0-5,0 RPGM1.txt	0.14	0.77	0.54	10	5	RPGM		
0-20,0-15,0 Markov3.txt	0.14	0.04	1.58	20	15	Markov		

Figure 22: An example of evaluation file

2. Select an evaluation file
3. Select the output Excel file

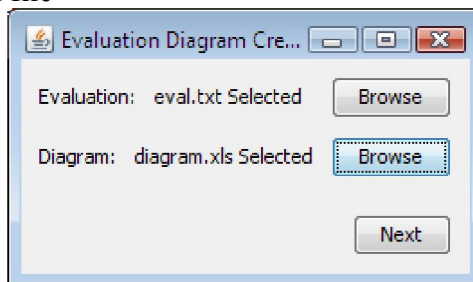


Figure 23: First dialog to create the diagram

4. Select the column which will be used for constructing the rows

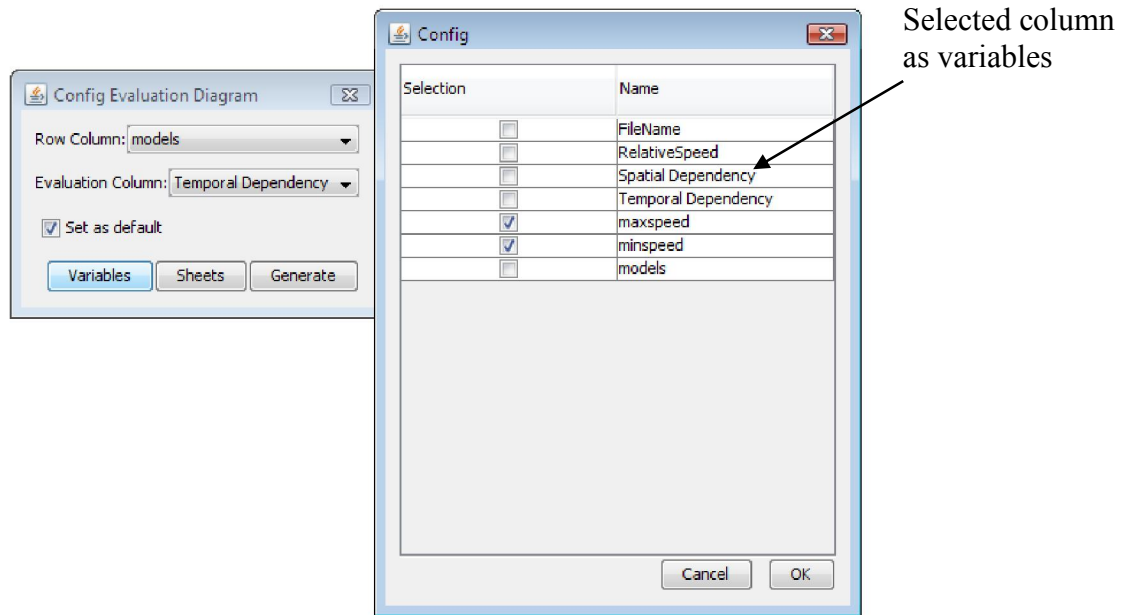


Figure 24: Second dialog to configure the output diagram

5. Select the evaluation column. Evaluation column values will be the numbers in the output table and construct the diagram. Note that if there were more than one row in the evaluation file for a cell in the output table, the value of the cell will be the average of them. The columns which you selected for rows, variables and sheets make a cell in the table unique.
6. Select variables. Variable columns will construct the columns of the output table. If you select more than one variable, the columns of the table will contain all available combination of these variables in the evaluation file.
7. Select the columns which will be used for constructing the sheets of the excel file. There will be a few situations that you want to test the impact of a variable on an evaluation in several models with different configurations for example the effect of max/minspeed on “Temporal Dependency” in several models if we have different transition pausetime values. You can either select max/minspeed and transition pausetime as variables and have a big table or break it in some sheets. For this example, I select none of them.
8. Click on generate button. The program creates the Excel file and opens it. The output may be like Figure 25.

1	Parameters					
2	Variables	maxspeed-minsp	Evaluation on	RelativeSpeed	Create Diagrams	
4	models	10-5	20-15	30-25	40-35	50-45
5	Manhattan	4.48	8.02	8.85	8.575	8.645
6	Markov	1.66	1.595	1.37	1.445	1.2
7	RandomDirection	2.345	5.565	9.26	13.72	17.315
8	RandomWayPoint	2.54	6.295	11.555	17.095	22.93
9	RPGM	0.555	0.765	1.165	1.295	1.575

Figure 25: Created Excel file (note to the macro security warning)

- To draw the diagram in all sheets, the implemented macro should be enabled. Open the security warning and enable the macro.

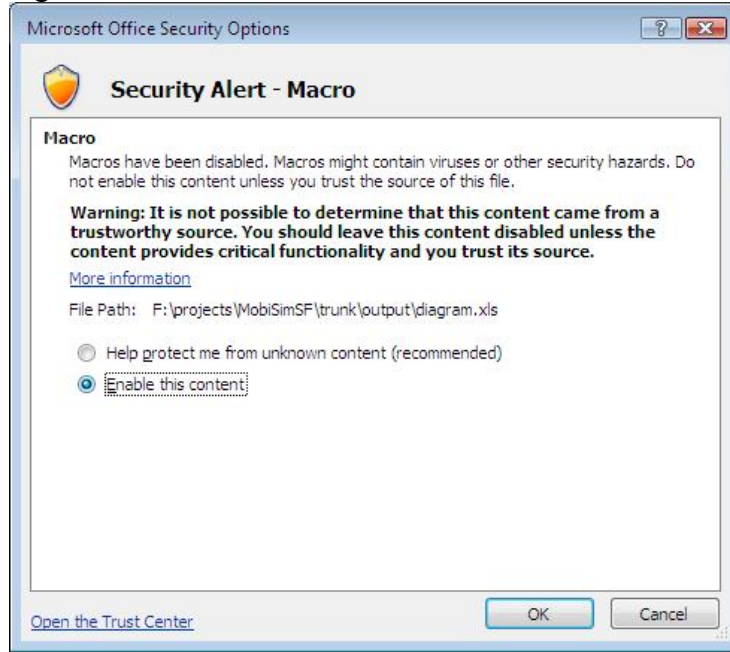


Figure 26: How to enable the macro in Excel

- Click on the “Create Diagrams” button in the first sheet. Figure 27 shows an example. If there were more than one sheet, the parameters which make a sheet unique would be listed in the first row, Parameters row (which is empty in this example), and the value of them would form the sheet title.

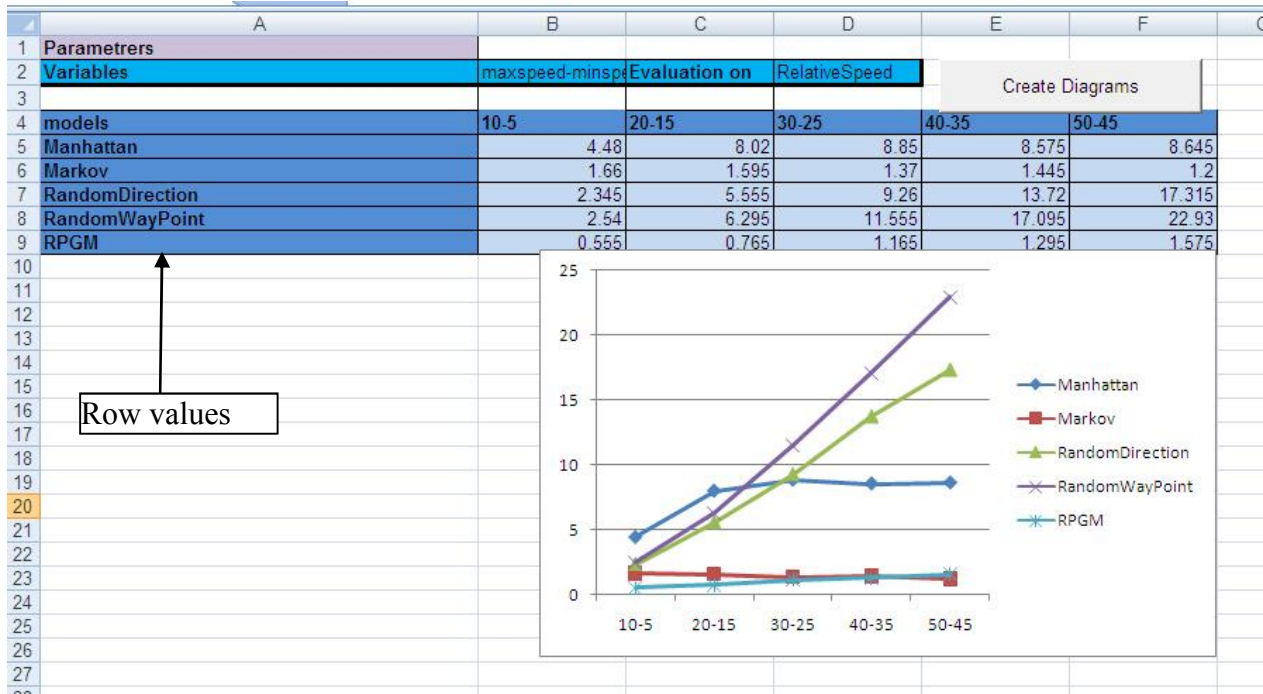


Figure 27: The result diagram, comparing the effect of changing minspeed and maxspeed on relative speed of nodes in five models



### 3-5 Mobility model recognizer

This module uses a clustering or classification algorithm to cluster or classify the evaluations in groups, usually the mobility models. Use this module as follows:

1. Select the evaluation file.
2. Select the output file
3. Select the algorithm.

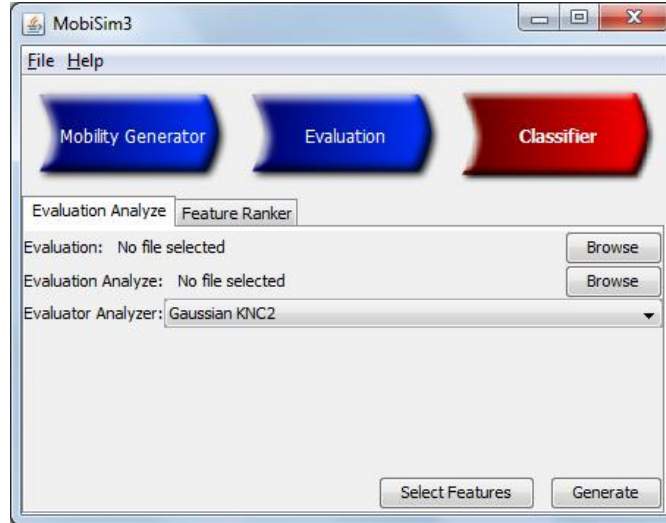


Figure 28: Evaluation analyzer module dialog

4. Select the factors that the algorithm may use to cluster/classify the files (note that factors will be normalized). The factors must be common between test and train files but their position and order are not important.
5. Configure the algorithm. It may need a file to learn, so you should have created a training file. The training file must have the selected factors and an additional column which will be used for training and will be selected later. For example, I selected the “models” column for training (besides I could add an additional column for this purpose).
6. Click on generate button and select learning factor.

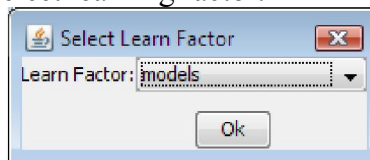


Figure 29: Select a column as a learning factor

The output file will have an additional column named “groupname”. In fuzzy algorithms output files will have an additional column for each learning class which shows what is the membership of each trace in that class.

#### 3-5-1 Model recognizer algorithms

- **KNC Classification** (Mousavi, Rabiee, Moshref, & Dabirmoghaddam, 2007): The algorithm works like this:
  1. Creates a point in an Euclidean space for each training file using the selected factors

2. Creates the center of each class using the training data by averaging the coordinates.
3. Calculates the coordinates of each trace file (a row of evaluation file) and put them in the nearest class.

In the output file, you can find the found class for each trace file in the “groupname” column.

- **Nearest Distance:** It is possible to create a coordinate system using the distances of center of groups used in the KNC method. The number of dimensions of the new system is equal to the learning classes. Each sample is labeled to a class if it has the smallest distance to the group in the new system.
- **Fuzzy KNN (Khaledi, Rabiee, & Khaledi, 2010):** It is alike KNN in which K nearest neighbors of a point are considered and the evaluated node will be in the class with the most nearest neighbors to the point. However, this is a fuzzy version of that which means that a point (trace) can be member of every class but with a membership factor ( $<1$ ). The output file has additional columns which tell the membership of each trace file to each class.
- **Gaussian KNC:** There was a theory in our group that there may be a Gaussian relation between the points in classes. I have implemented the algorithm in three versions. Firstly, the distances of members of each group from its centers may create a **distinct** Gaussian distribution. The following blocks present each algorithm.

```

For each learned group G with C as its center
  for each learned sample "in G" as L
    find distance of L to C as D
    calculate mean(G) and sigma(G) of Ds

for each test sample as T
  for each learned group G with C as its center, M as its mean, Sigma as
its sigma
    D= distance of T to C
    alpha(T,G) = f(D,M,Sigma)
    sumAlpha=sumAlpha+alpha
  for each learned group G
    alpha(T,G)=alpha(T,G)/sumAlpha
end

```

**Algorithm 1: Gaussian KNC algorithm (f is the Gaussian distribution function)**

Secondly, when the number of samples in each group is small, we can consider that the distances of members of each group from its centers could create **one** distribution.

```

For each learned group G with C as its center
  for each learned sample "in G" as L
    find distance of L to C as D
    calculate mean(G) and sigma(G) of Ds

for each test sample as T
  for each learned group G with C as its center, M as its mean, Sigma as
its sigma
    D= distance of T to C
    alpha(T,G) = f(D,M,Sigma)
    sumAlpha=sumAlpha+alpha
  for each learned group G
    alpha(T,G)=alpha(T,G)/sumAlpha

```

end

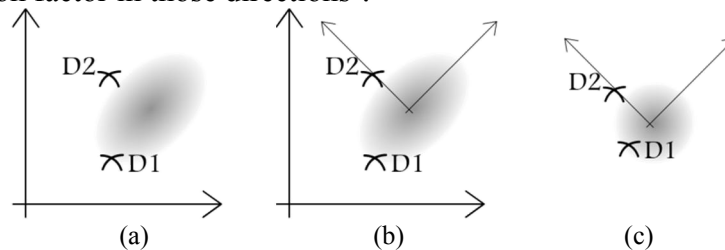
**Algorithm 2: The second version of Gaussian KNC**

The third version supposes that the distances of points in a class **from the center of other classes** create a Gaussian distribution.

```
For each learned group G with C as its center
  for each learned sample "in G" as L
    find distance of L to C as D
    calculate mean(G) and sigma(G) of Ds

for each test sample as T
  for each learned group G with C as its center, M as its mean, Sigma as
  its sigma
    D= distance of T to C
    alpha(T,G) = f(D,M,Sigma)
    sumAlpha=sumAlpha+alpha
  for each learned group G
    alpha(T,G)=alpha(T,G)/sumAlpha
end
```

- **Group SVD:** The Gaussian algorithms suppose that the distribution of data around the mean in all dimensions is Gaussian with equal variance. In this algorithm I first find the distances of all members of each group in the learning classes from the center. Then using SVD<sup>4</sup> method, find a new basis for the points in each class. In Figure 30, I have presented an example. Suppose that we have two evaluation metrics and presented the traces for a class (for example, Manhattan) on this coordination which create the sphere distribution. Now, we have two test data, D1 and D2. The question is which one is closer to this class. If we use Euclidean distance definition, they have equal distance from the center of the class. However, we can see that the class distribution is much denser in one direction, so we can say that the probability for D2 to be in the class is less than for D1. To do this, first we find a new coordination, part (b), and then scale all data (learn and test) in the new coordination system to normalize them, part (c). Now the Euclidean distance definition is used to find the distances in the new coordination and we can see that D1 is much closer to the center of class. Note that to use this algorithm the number of learn samples in each class must be larger than dimensions (selected features). There is one unanswered question in this method: how should we deal with data in a class that all are located on one direction? In this case, we cannot normalize data in some directions. I have defined a parameter for this algorithm, *singularcost*, which will be used as the normalization factor in those directions<sup>5</sup>.



<sup>4</sup> Singular Value Decomposition

<sup>5</sup> This should mimic Mahalanobis distance

Figure 30: Distribution of points in a class and two test data.

- **Nearest SVD:** the logic behind Group SVD can also be used in the Nearest Distance algorithm. As this method works on the coordination system created based on the distances among groups (classes), distance matrices are always symmetric and doesn't need the limitation on the number of learn samples in each class.

### 3-6 Evaluating factors in model recognizer algorithms

There are many evaluators for mobility models and each category of them can separate some specific class of models. Therefore, I have added a process to run a classification algorithm on an evaluation trace with all combination of features. Besides, you can evaluate each algorithm to find out with which factors it works better. To start evaluation, you must have some samples with known class like the learning samples. The process needs a test file in "Evaluation" field, an output file to write the evaluation of classifying algorithm on specified on "Evaluation Analyze" field. The output of each classification run will be written in a file in the folder specified in "Classification output folder" field.

"Label Column" field contains the column header of known classes of the input traces. This column will be used for evaluation of classification. Each type of classifying algorithms, (Fuzzy) classifiers, and clustering algorithms has its own evaluators while accuracy is common among them. Configure them using "Config Ranking Evaluators" button. Then, specify minimum and maximum number of features that must be checked in this process. Determine which columns in input file must be in the pool of features using "Select Features" button and at last click on the generate button. Note that the process may ask you about the learning file address and the class column in the learning file later.

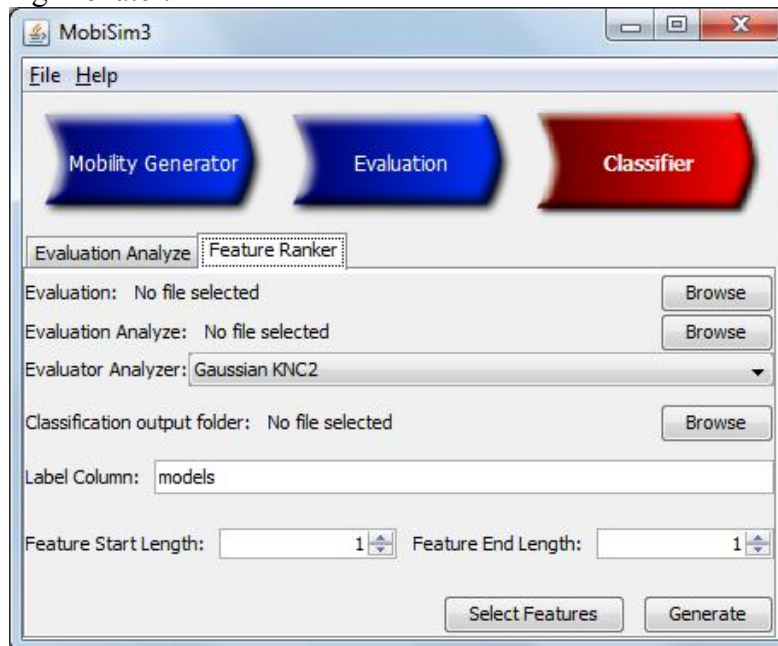


Figure 31: Feature Ranker window

Evaluators for classification algorithms are as follow:

### 3-6-1 Accuracy

It measures the number of traces the classifier could put in the right class and expresses the value in percent. For fuzzy algorithms, it assumes that the class with the largest membership value is the selected one and works based on it.

### 3-6-2 Membership per class

For fuzzy classification algorithms, it calculates what percent of traces are classified in which classes. For example, it says that by average each traces has been %25 in Manhattan class.


### 3-6-3 Right membership average

In fuzzy classification, each trace may be in every class but with a low membership value. This evaluator averages the membership of the traces in their right class.

## 4 Mobility models

Each model has a map, a range algorithm, and a location initializer. Note that some models may have special maps or location initializers. I will discuss the general ones first but the special choices are described when needed.

### 4-1 General maps

- **Square Map:** This is a square map which doesn't propose reflection functionality and only has a size handle .

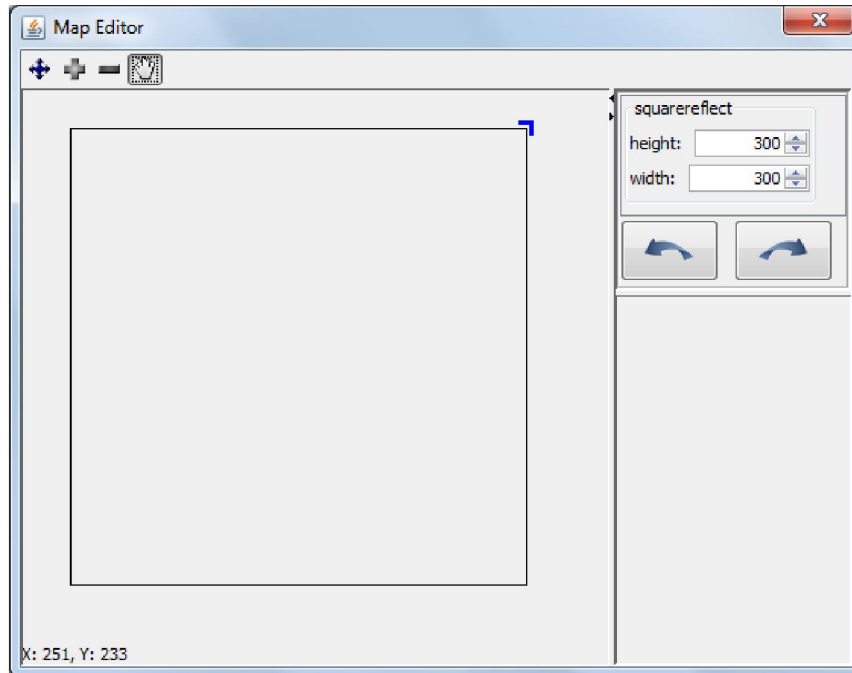


Figure 32: Square map Map Editor window

- **Disk Map:** This is a circular map which only allows a node to walk in the disk and have its transition destination in the disk. This map doesn't propose reflection functionality and only has a size handle which determines its radius.

- **Circle Map:** This map is similar to the Disk Map but only permits the transition destination on its border.

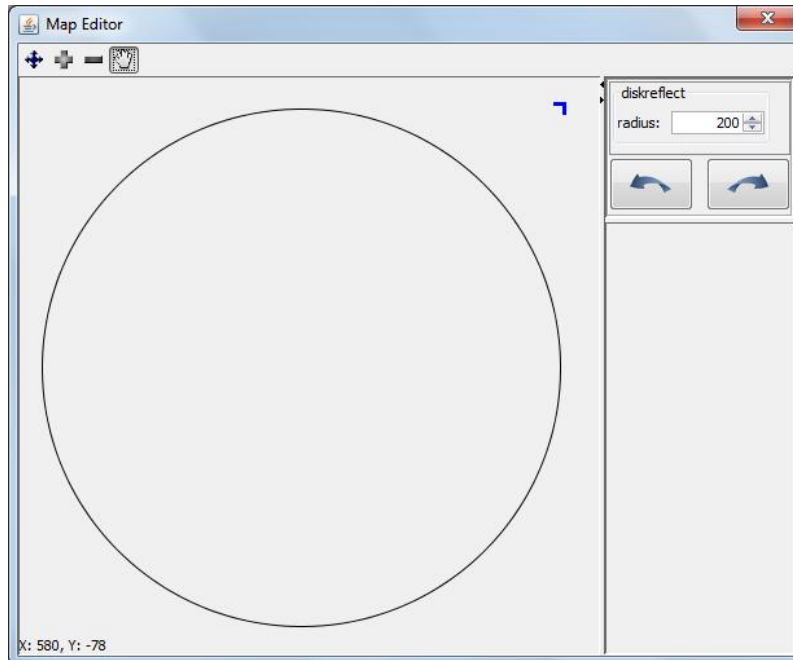


Figure 33: Circular and Disk map Map Editor window

- **Square Reflect Map:** It is a square map which has a reflective border and has similar parameters to square map.
- **Disk Reflect Map:** It is a circular map which has a reflective border and has similar parameters to disk map.

#### 4-1-1 Obstacle map

Obstacle map as a square reflect map can have multiple obstacles. Each obstacle has a rotation handle which enables you to rotate the obstacle. Obstacles can overlap each other and a node may hit more than an obstacle in one instance. Currently, only rectangular obstacles have been implemented.

You should add obstacles using the “Config” button. Then you can move, scale, or turn them.

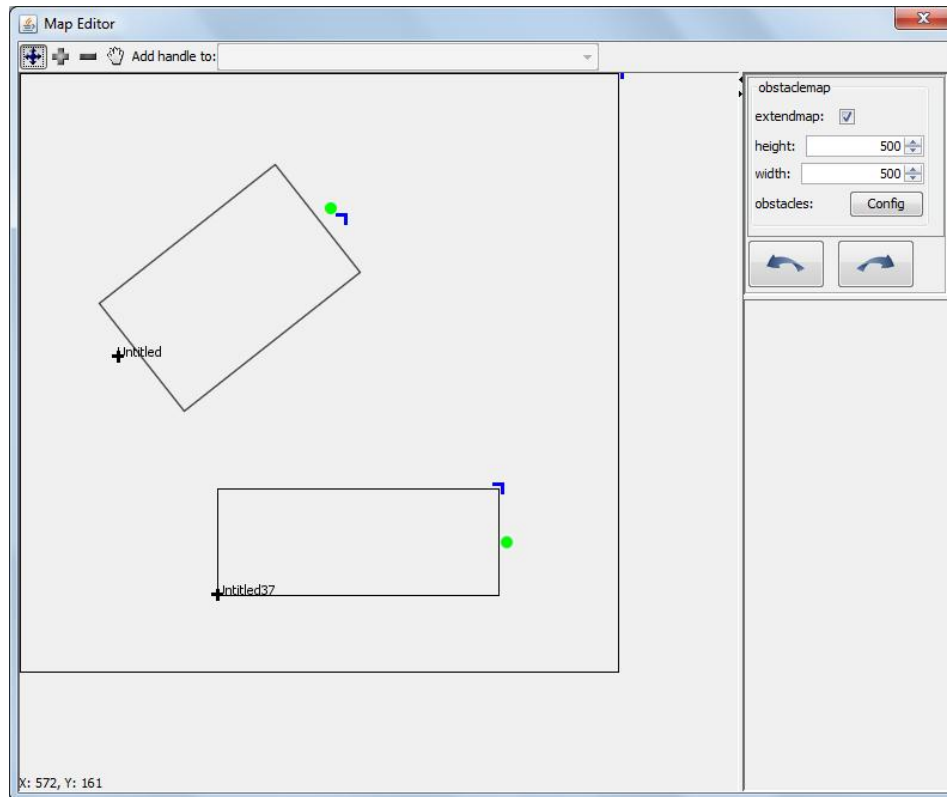


Figure 34: The view of the obstacle map

## 4-2 Range algorithms

The range algorithms set the range attribute of the nodes. Later you can use these range values in a network simulator to run a network protocol or compute the power used to keep the connection among nodes.

### 4-2-1 Fixed

Simply, puts a fix number for the range of nodes. You can set this fix number using the “value” parameter.

### 4-2-2 RNG

Changes the ranges of the nodes too create a [Relative Neighborhood Graph](#): “connect two points  $p$  and  $q$  by an edge whenever there does not exist a third point  $r$  that is closer to both  $p$  and  $q$  than they are to each other.” This graph keeps multiple paths between two nodes in contrast to a tree.

### 4-2-3 MST

This changes the ranges of nodes to make the Minimum Spanning tree. Note that as it wants to keep two directional connections, still you may see loops in the graph.

### 4-2-4 Null

This choice does not change the range of the nodes. When is it useful?

- Consider you want to replay a file trace using the File Model and you want to use the ranges in the file. To not mess with the range values loaded from file select the Null power algorithm for the File Model

- In the Multit-Model mobility model, you may want for the nodes of each model to keep their own communication graph irrespective of the nodes in the others. For that, set the range algorithm for the multi-model to Null.
- In the group model you may want that each group keep its own communication but they should not try to connect to each other. Set the range algorithm for leader model to Null and check the pergroup range option for the group model.

### 4-3 Location initializers

Location initializers help you specify the initial position of nodes for the simulation.

#### 4-3-1 Fixed Initializer

In robotic applications and for debugging the new mobility models you may need to put each node in a predefined position. The Fixed Initializer helps you put the node in the two dimensional coordination. It gives each node a position from its list and if there were more nodes it restarts from the first position.

#### 4-3-2 Random Initializer

Random Initializers sets the X and Y position of each node based on a random distribution. This initializer has two random parameters for each coordination axis. You can choose different random distributions which will be described in 4-4-5-1 . Besides, note how the random seed can be fixed using Main menu.

#### 4-3-3 Null Initializer

It is the default location initializer which models that do not support general initializers may choose and usually you should not worry about it.

### 4-4 Random models

Models in this category use random numbers to determine node's direction and speed, which will be independent of last and other nodes' speed and direction.

#### 4-4-1 Random Waypoint Model

The Random Waypoint Model was first proposed by Johnson and Maltz (Broch, Maltz, Johnson, Hu, & Jetcheva, October 1998). Soon, it became a 'benchmark' mobility model to evaluate the MANET routing protocols, because of its simplicity and wide availability.

The implementation of this mobility model is as follows: as the simulation starts, each mobile node randomly selects one location in the simulation field as the destination. It then travels towards this destination with constant velocity chosen uniformly and randomly from  $[0, V_{Max}]$ , where the parameter  $V_{Max}$  is the maximum allowable velocity for every mobile node (L. Breslau, 2000). The velocity and direction of a node are chosen independently of other nodes. Upon reaching the destination, the node stops for a duration defined by the 'pause time' parameter. If  $T_{Pause} = 0$ , this leads to continuous mobility. After this duration, it again chooses another random destination in the simulation field and moves towards it. The whole process is repeated again and again until the simulation ends (Bai, Sadagopan, & Helmy, 2003).

There are two implementation of this model in the software: First, Random Waypoint without reflection, which only selects a destination point inside the map, so it needs no reflection. This model works with **Error! Reference source not found. (Error! Reference source not found.)**.



Secondly, Random Waypoint model, which works with **Error! Reference source not found.** (**Error! Reference source not found.**).

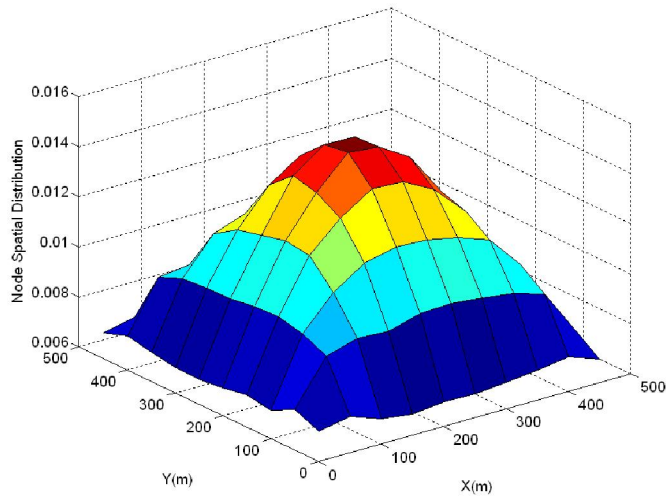


Figure 35: Spatial Node Distribution in Random Waypoint Mobility Model

#### 4-4-2 Random Direction

The Random Direction model based on similar intuition is proposed by Royer, Melliar-Smith and Moser (Royer, Melliar-Smith, & Moser, June 2001). This model is able to overcome the non-uniform spatial distribution problem. Instead of selecting a random destination within the simulation field, in the Random Direction model the node randomly and uniformly chooses a direction by which to move along until it reaches the boundary. After the node reaches the boundary of the simulation field it stops with a pause time  $T$ , then it randomly and uniformly chooses another direction to travel. This way, the nodes are uniformly distributed within the simulation field (Camp, Boleng, & Davis), (Bai, Sadagopan, & Helmy, 2003).

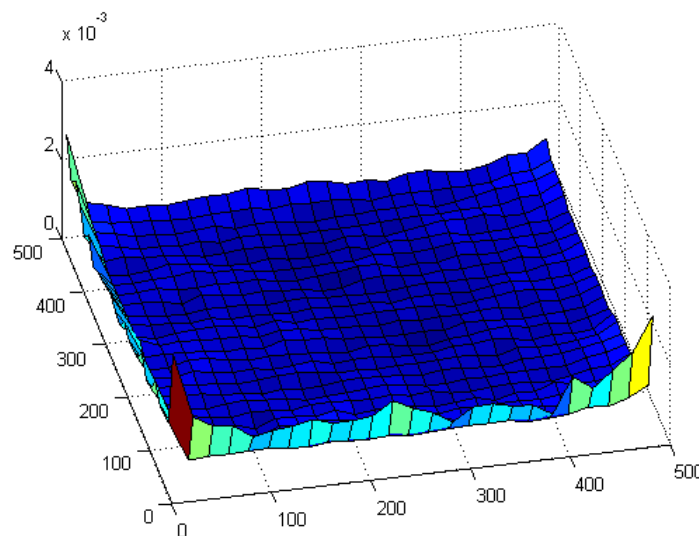


Figure 36: Spatial Node Distribution in Random Direction Mobility Model

### 4-4-3 Random Walk

The Random Walk model was originally proposed to emulate the unpredictable movement of particles in physics. It is also referred to as the Brownian Motion. Because some mobile nodes are believed to move in an unexpected way, Random Walk mobility model is proposed to mimic their movement behavior (Bai, Sadagopan, & Helmy, 2003). The Random Walk model has similarities with the Random Waypoint model because the node movement has strong randomness in both models. We can think the Random Walk model as the specific Random Waypoint model with zero pause time.

However, in the Random Walk model, the nodes change their speed and direction at each time interval. For every new interval  $t$ , each node randomly and uniformly chooses its new direction  $\theta(t)$  from  $(0, 2\pi]$ . In similar way, the new speed follows a uniform distribution from  $[0, V_{Max}]$ . Therefore, during time interval  $t$ , the node moves with the velocity vector  $(v(t)\cos(\theta(t)), v(t)\sin(\theta(t)))$ . If the node moves according to the above rules and reaches the boundary of simulation field, the leaving node is bounced back to the simulation field with the angle of  $\pi - \theta(t)$ . This effect is called border effect (Bettstetter & Wagner, Mar 25-26, 2002) or reflection rule.

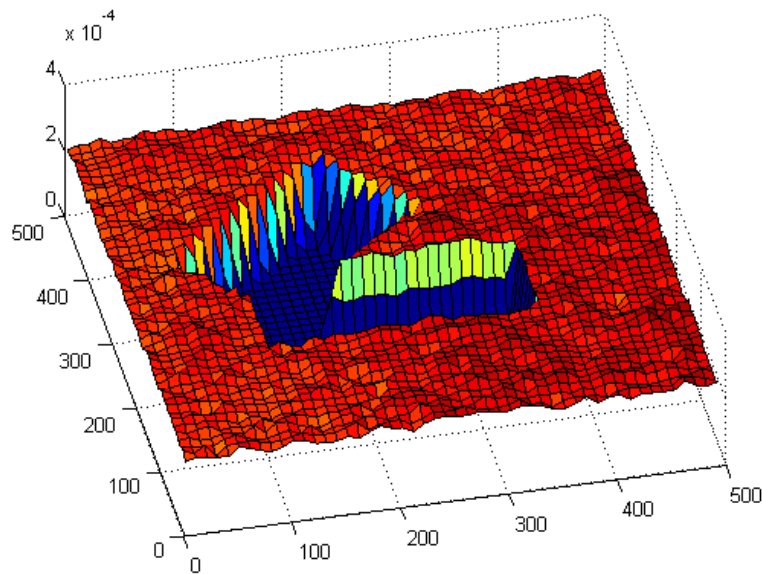


Figure 37: Spatial node distribution in Random Walk model on obstacle map

### 4-4-4 Levy Walk Model

Levy walk is similar to random walk, but its walk time (flight time) and pause time are more complex. Each step is a tuple  $S=(l, \theta, \Delta t_f, \Delta t_p)$ . In this model, each node takes a random direction  $\theta$ , and a flight length  $l$ . These values specify a destination point in or out the map.  $\Delta t_f$  indicates the flight duration and picked for each flight from a probability distribution  $p(l)$ . Using the  $\Delta t_f$  and  $l$ , the model calculates speed of flight.  $\Delta t_p$  specifies pause time at the end of a flight and it will be calculated using a configurable distribution  $\psi(\Delta t_p)$  (Chong, 2008).

In order to create random numbers for needed distributions, I used random generator algorithm presented in MatLab codes that the writers of model uploaded in

<http://netsrv.csc.ncsu.edu/twiki/bin/view/Main/MobilityModelsDnld>. The random generator needs some parameters which follow:

- *minflight* & *maxflight*: specifying the range of flight length.
- *alpha*
- *beta*
- *minpausetime* & *maxpausetime*: specifying the range of pause time at the end of flight.
- *flightscale*.



Figure 38: Traveling pattern of nodes in Levy Walk model

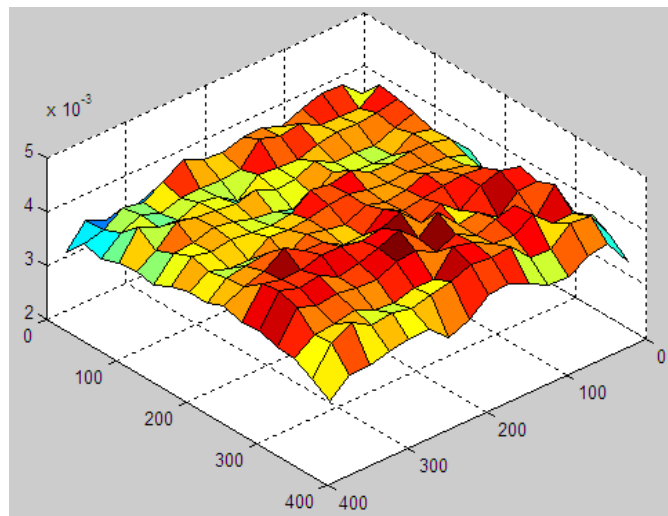


Figure 39: Spatial node distribution in Levy Walk model

#### 4-4-5 Tortoise Model

This is one of the most general and flexible models that can create many mobility behaviors. In Tortoise model for each node, you specify a sequence of transitions in some loops while a sequence or loop can be inserted as a step of another one. Therefore, it is possible to create a tree of sequences and loops of transitions. I call each step an action. Besides, it is possible to choose a random number generator for each parameter of transition. As a result, you can create almost any model in the random mobility models category using this model. Moreover, this model is very useful for the leader model of group models in which you can control the general movement of a

group. As the last point, it is also possible to create fancy movements. See Figure 40 for some examples.

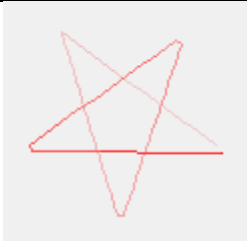
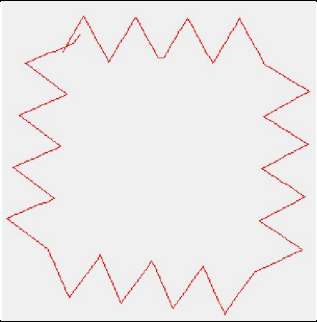
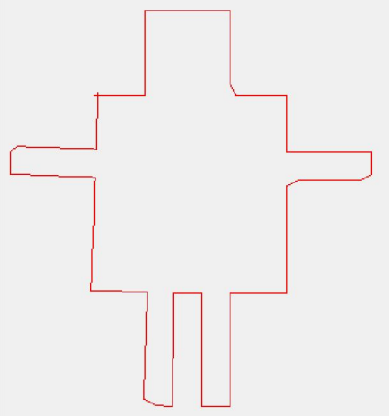
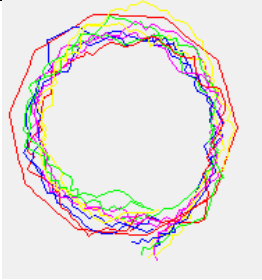
	
<ul style="list-style-type: none"> <li>• LS(l=5) <ul style="list-style-type: none"> <li>○ T(a=144,d=20,s=5)</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• LS(l=0) <ul style="list-style-type: none"> <li>○ T(a=60,d=10,s=5)</li> <li>○ LS(l=3) <ul style="list-style-type: none"> <li>▪ LS(l=2) <ul style="list-style-type: none"> <li>• T(a=240,d=10,s=5)</li> <li>• T(a=120,d=10,s=5)</li> </ul> </li> <li>▪ T(a=240,d=10,s=5)</li> <li>▪ T(a=30,d=10,s=5)</li> </ul> </li> </ul> </li> </ul>
	
<p>A very complex long list of actions!</p>	<ul style="list-style-type: none"> <li>• Pursue with leader model=tortoise <ul style="list-style-type: none"> <li>○ LS(l=10) <ul style="list-style-type: none"> <li>▪ T(a=36,d=10,s=5)</li> </ul> </li> </ul> </li> </ul>

Figure 40: Sample movements and their sequence of actions using Tortoise model

There are three types of actions: LoopSequence, Transition, and RandomizedTransition. Each action has an order parameter, which shows the position of the action in the parent action sequence.

The Tortoise model has a LoopSequence action as the root action. This action has a sequence of actions that will be run in order. If its “loop” parameter is greater than zero, it runs the sequence “loop” multiple times. When the root action reaches its end, nodes will stand still till the end of simulation time.

A Transition action makes a node move toward *angle* for a *duration* with a *speed*. Then when it reaches the destination it will pause to pass *pausetime*. So angle, duration, speed and pausetime are parameters of a transition. The Transition decides to select the angle relative to current movement or the coordination system based on the value of *relativeangle* parameter.

Using these two action types, you can create any kind of transition based movement. However, if you want to set the value of one or more of these parameters using a random function, you must use `RandomizedTransition`.

#### 4-4-5-1 *Random variables:*

I have used [JSC](http://www.jsc.nildram.co.uk/) (<http://www.jsc.nildram.co.uk/>) library for random distributions. Each distribution function has some parameters that are passed to it using *params* array. If the required parameters would be integer, the program converts the double values. It also adds zero parameters if needed. Note that the output value will be the absolute value of *random\*scale+bias*. For each parameter of a transition you can use one of the following distribution functions<sup>6</sup> (Note that the requirements in parentheses must be met):

- **Bernoulli**: the probability of success=*param1* ( $0 < p1 < 1$ )
- **Beta**: *p*=*param1* ( $>0$ ), *q*=*param2* ( $>0$ )
- **Binomial**: number of trials=*param1* ( $>0$ ), probability of success=*param2* ( $0 < p2 < 1$ ).
- **Cauchy**: location (median)=*param1*, scale=*param2* ( $>0$ )
- **DiscreteUniform**: minimum=*param1*, maximum=*param2* ( $p2 > p1$ )
- **Exponential**: mean=*param1* ( $>0$ )
- **ExtendedHypergeometric**: *n1*=*param1*, *n2*=*param2*, sum of the observed binomial variates=*param3* ( $0 \leq p3 \leq n1+n2$ ), the odds ratio=*param4* ( $0 < p4$ ), tolerance=*param5* (optional (Default=0))
- **ExtremeValue**: location=*param1*, scale=*param2* ( $>0$ )
- **Fixed**: value=*param1*
- **FisherF**: freedom degree in numerator=*param1* ( $>0$ ), freedom degree in denominator=*param2* ( $>0$ )
- **Gamma**: shape=*param1* ( $>0$ ), scale=*param2* ( $>0$ )
- **Geometric**: the probability of success=*param1* ( $0 < p1 < 1$ )
- **Hypergeometric**: sampleSize=*param1* ( $0 \leq p1 \leq p2$ ), populationSize=*param2* ( $0 \leq p2$ ,  $p3 \leq p2$ ), markedItemsCount=*param3* ( $0 \leq p3 \leq p2$ )
- **Laplace**: mean=*param1*, scale=*param2* ( $>0$ )
- **Levy**: alpha=*param1* ( $.1 \leq p1 \leq 2$ , beta=*param2* ( $-1 \leq p2 \leq 1$ ), c=*param3*, delta=*param4*
- **LogarithmicSeries**: alpha=*param1* ( $0 < p1 < 1$ )
- **Logistic**: mean=*param1*, scale=*param2* ( $>0$ )
- **Lognormal**: location=*param1*, scale=*param2* ( $>0$ )
- **NegativeBinomial**: nth success=*param1* ( $>0$ ), probability of success=*param2* ( $0 < p2 < 1$ )
- **NoncentralBeta**: *p*=*param1* ( $>0$ ), *q*=*param2* ( $>0$ ), noncentrality parameter =*param3* ( $\geq 0$ )
- **NoncentralChiSquared**: degrees of freedom=*param1* ( $>0$ ), noncentrality parameter=*param2* ( $\geq 0$ )
- **NoncentralFishersF**: freedom degree in numerator=*param1* ( $>0$ ), freedom degree in denominator=*param2* ( $>0$ ), noncentrality parameter=*param3* ( $\geq 0$ )
- **NoncentralStudentsT**: degrees of freedom=*param1* ( $>0$ ), noncentrality parameter=*param2* ( $\geq 0$ )

---

<sup>6</sup> See JSC documentation for the complete list of distributions and detailed information.

- **Normal**: mean=param1, standard deviation=param2 (>0)
- **Pareto**: location=param1 (>0), shape=param2 (>0)
- **Poisson**: mean=param1 (>0)
- **PowerFunction**: scale=param1 (>0), shape=param2 (>0)
- **StudentsT**: degrees of freedom=param1 (>0)
- **Uniform**: minimum=param1, maximum=param2 (p2>p1)
- **Weibull**: scale=param1 (>0), shape=param2 (>0)

Just type the bold name in the *distribution* field. You can add your random generator function by implementing “Distribution” class of JSC library. For example, I have implemented an “Adapter (Gamma, Helm, Johnson, & Vlissides, 1994)” for the Levy distribution (see 4-4-4 ). To use it you must type “Levy” in the distribution field. Therefore, you can regenerate LevyWalk model using Tortoise model.

#### 4-5 Manhattan models

Models in this category are under geographical constraints, and they usually use a map to limit node’s motion.

##### 4-5-1 Freeway

This model is proposed to emulate the motion behavior of mobile nodes on a freeway (Bai, Sadagopan, & Helmy, 2003). The freeway map used in our simulations is shown in Figure 41. Freeway model can be used in applications such as exchanging traffic status between vehicles or tracking a vehicle on a freeway.

There can be one freeway on the map and a freeway has lanes in both directions. The differences between Random Waypoint and Freeway are the following:

- a) Each mobile node is restricted to its lane on the freeway.
- b) The velocity of mobile nodes is temporally dependent on its previous velocity.
- c) If two mobile nodes on the same freeway lane are within the safety distance (SD), the velocity of the following node cannot exceed the velocity of preceding node.

The inter-node and intra-node relationships involved are: If node  $j$  is ahead of node  $i$  in its lane then:

$$|\vec{V}_i(t+1)| = |\vec{V}_i(t)| + \text{random}() * |\vec{a}_i(t)|$$

$$\forall i, \forall j, \forall t, D_{i,j}(t) \leq SD \Rightarrow |\vec{V}_i(t)| \leq |\vec{V}_j(t)|$$

Due to the above relationships, the Freeway mobility pattern is expected to have spatial dependence and high temporal dependence. It also imposes strict geographic restrictions on the node movement by not allowing a node to change its lane.


In our proposed simulation, the model has 4 new parameters. This new parameters make our simulation scheme more similar to real world motion of cars in a freeway.

- 1- *Fixed acceleration*: it is a Boolean parameter that specifies whether a node should have fixed acceleration in each transition or not.
- 2- *Max acceleration* which specifies the maximum acceleration that a node can have in a time slot.
- 3- *Positive speed ratio*: if the node can change its acceleration in each transition, its acceleration will be random. Therefore, this parameter specifies the weight of positive acceleration to negative one. For example, the value 2 means that the probability that a node accelerate is two times of that a node decelerate.

- 4- *Safe distance ratio*: In real world, each car should keep a safe distance from the forward car. This distance depends on the speed of the vehicle. For example, they may keep distance equal to three times their current speed. This parameter specifies the ratio of safe distance to the current speed of the node (actually it is the duration needed to reach the forward car if it stops at once).

The model's map also has its special parameters that follow:

- 1- Points to make the freeway Map,
- 2- Number of lanes in each direction,
- 3- Horizontal space between lanes,
- 4- Horizontal space between lanes in opposite direction

Each point in this model is a handle . You can add, remove, or move a handle. Note that these handles should be in the map area presented by the size handle. As the sequence of these points is important in constructing the map, freeway handles have an additional parameter, sequence, which determines the position of a handle among others. As you see in Figure 42, you are free to create any map you want.

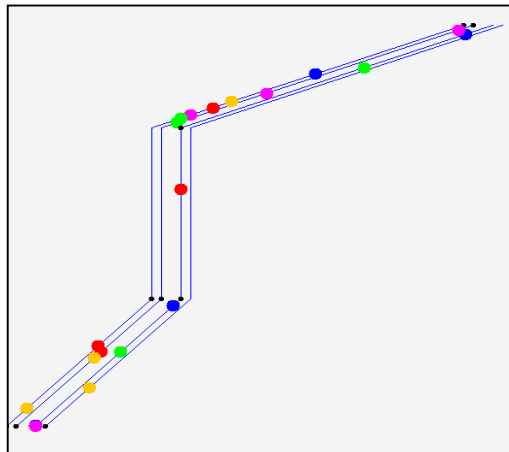


Figure 41: Freeway Mobility Model Map

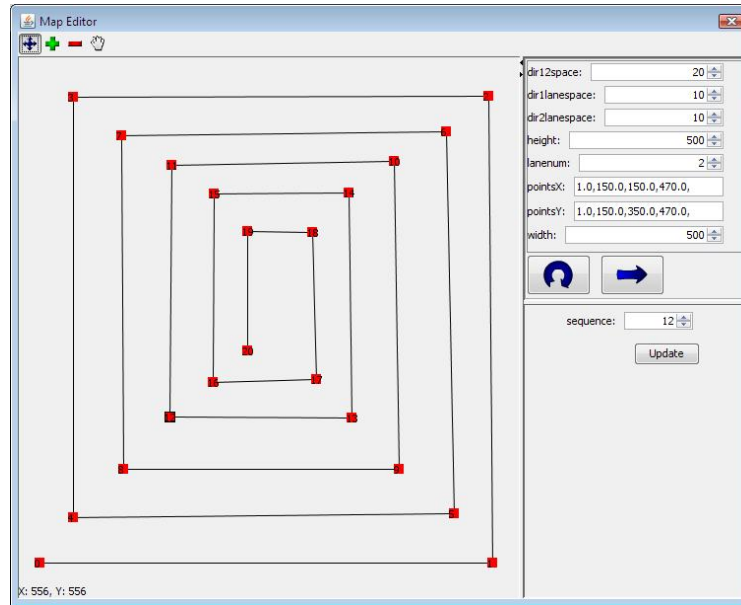


Figure 42: Freeway Model Map Editor window

#### 4-5-2 Manhattan

The Manhattan mobility model emulates the movement pattern of mobile nodes on streets defined by maps. It can be useful in modeling movement in an urban area where a pervasive computing service between portable devices is provided (Bai, Sadagopan, & Helmy, 2003).

This model also uses its own map. The map is composed of a number of horizontal and vertical streets. Each street has two lanes for each direction (north and south direction for vertical streets, east and west for horizontal streets). The mobile node is allowed to move along the grid of horizontal and vertical streets on the map. At an intersection of a horizontal and a vertical street, the mobile node can turn left, right or go straight. This choice is probabilistic: the probability of moving on the same street is 0.5, the probability of turning left is 0.25 and the probability of turning right is 0.25. The velocity of a mobile node at a time slot is dependent on its velocity at the previous time slot. In addition, a node's velocity is restricted by the velocity of the node preceding it on the same lane of the street. The inter-node and intra-node relationships are the same as in the Freeway model. Thus, the Manhattan mobility model is also expected to have high spatial dependence and high temporal dependence. It also imposes geographic restrictions on node mobility. However, it differs from the Freeway model in giving a node some freedom to change its direction.

In proposed simulation scheme for Freeway and Manhattan in (Bai, Sadagopan, & Helmy, 2003), the writers did not describe the nodes behavior when it reaches the simulation boundaries. In my simulator, when a node reaches boundaries of the simulation area, it chooses the opposite lane and turns back on the opposite lane. Figure 43 shows a Manhattan map.

While model parameters are similar to the freeway parameters, its map has its own parameters:

1. *dir12lanespace* determines free space between two lanes in a street.
2. *hlines* presents horizontal lines.
3. *vlines* presents vertical lines.



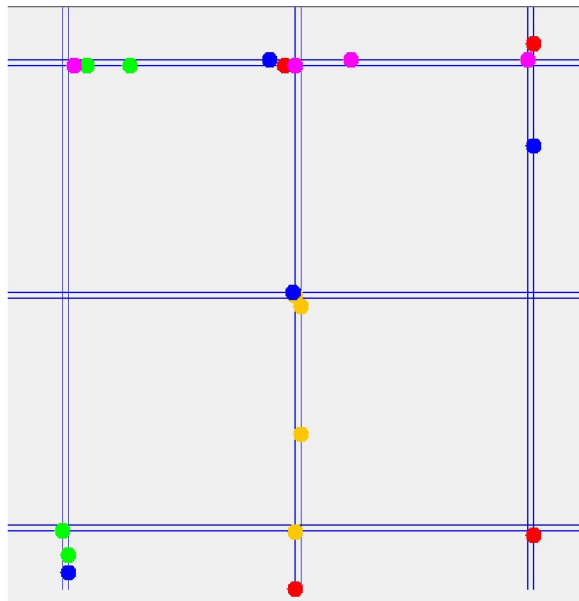


Figure 43: Nodes' movement in Manhattan Mobility Model

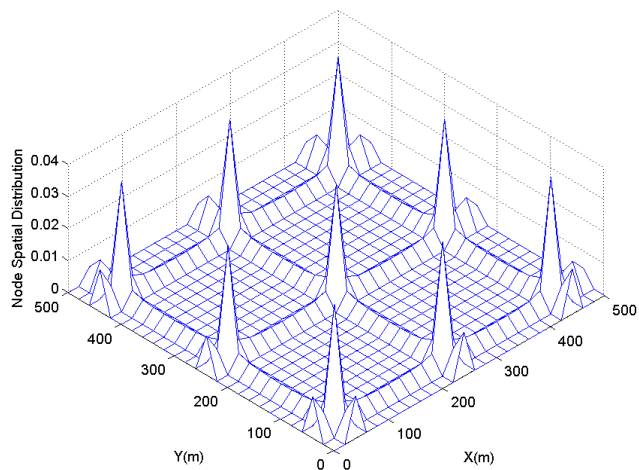




Figure 44: Spatial Node Distribution in Manhattan Mobility Model

In the Manhattan map editor, handles define the horizontal (  ) and vertical (  ) lines. You can add, remove, or move these handles, but note that these handles must be in the map area. Manhattan map handles have a parameter specifying whether the handle is for horizontal lines or not.

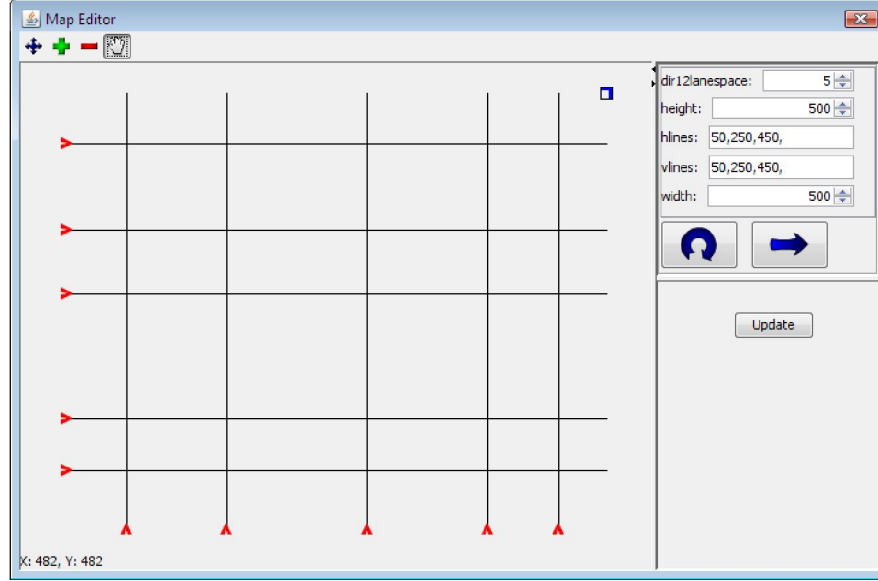


Figure 45: Manhattan map MapEditor window

## 4-6 Temporal dependent models

In the real world, current time node's motion is related to its past behavior. Models in this category manage to relate the current motion characteristics to the past ones.

### 4-6-1 Gauss-Markov

The Gauss-Markov Mobility Model was designed to adapt to different levels of randomness via a tuning parameter (Liang & Haas, March 1999). Initially each mobile node is assigned a current speed and direction. At fixed intervals of time  $n$ ; node's movement occurs by updating the speed and direction of each mobile node. Specifically, the value of speed and direction at the  $n$ <sup>th</sup> instance is calculated based upon the value of speed and direction at the  $(n-1)$ <sup>st</sup> instance and a random variable using the following equation:

$$s_n = \alpha s_{n-1} + (1-\alpha)\bar{s} + \sqrt{(1-\alpha)^2} s_{x_{n-1}}$$

$$d_n = \alpha d_{n-1} + (1-\alpha)\bar{d} + \sqrt{(1-\alpha)^2} d_{x_{n-1}}$$

Where  $s_n$  and  $d_n$  are the new speed and direction of the mobile node at time interval  $n$ ;  $\alpha$ , where  $0 \leq \alpha \leq 1$ , is the tuning parameter used to vary the randomness;  $\bar{s}$  and  $\bar{d}$  are constants representing the mean value of speed and direction as  $n \rightarrow \infty$ ;  $s_{n-1}$  and  $d_{n-1}$  are random variables from a Gaussian distribution. Totally random values or Brownian motion are obtained by setting  $\alpha = 0$  and linear motion is obtained by setting  $\alpha = 1$  (Liang & Haas, March 1999). Intermediate levels of randomness are obtained by varying the value of  $\alpha$  between 0 and 1.

At each time interval the next location is calculated based on the current location, speed, and direction of movement. Specifically, at time interval  $n$ , a Mobile node's position is given by the equations:

$$x_n = x_{n-1} + s_{n-1} \cos d_{n-1}$$

$$y_n = y_{n-1} + s_{n-1} \sin d_{n-1}$$

Where  $(x_n, y_n)$  and  $(x_{n-1}, y_{n-1})$  are the  $x$  and  $y$  coordinates of the mobile node's position at the  $n^{th}$  and  $(n-1)^{st}$  time intervals, respectively, and  $s_{n-1}$  and  $d_{n-1}$  are the speed and direction of the mobile node, respectively, at the  $(n-1)^{st}$  time interval (Camp, Boleng, & Davis).

I changed the specification that discussed in (Liang & Haas, March 1999), so I do not force nodes be away from borders but when they reach the border they reflect using the reflection rule I explained for Random Walk mobility model. The model parameters are memory factor ( $\alpha$ ), and random amplitude which is represented by  $\sigma$ . If I set the memory factor to 1, the model behavior becomes like Random Walk and if I set it to zero model behavior becomes like Brownian Motion. Figure 46 shows traveling pattern of 5 nodes with  $a = 0.1$ ,  $\sigma = 5$  and average speed=20m/s.

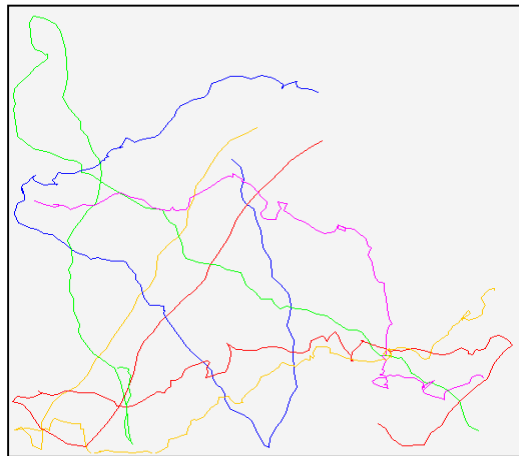


Figure 46: Traveling pattern of Mobile Nodes in Gauss-Markov Mobility Model

#### 4-6-2 Probabilistic Random Walk

Chiang's mobility model utilizes a probability matrix to determine the position of a particular mobile node in the next time step, which is represented by three different states for position  $x$  and three different states for position  $y$  (Chiang, 1998). State 0 represents the current ( $x$  or  $y$ ) position of a given mobile node, state 1 represents the mobile node's previous ( $x$  or  $y$ ) position, and state 2 represents the mobile node's next position, if the mobile node continues to move in the same direction. The probability matrix used is where each entry  $p(a,b)$  represents the probability that a mobile node will go from state  $a$  to state  $b$ . The values within this matrix are used for updates to both the mobile node's  $x$  and  $y$  position. In Chiang's simulator each node moves randomly with a preset average speed. The following matrix contains the values Chiang used to calculate  $x$  and  $y$  movements:

$$P = \begin{bmatrix} p(0,0) & p(0,1) & p(0,2) \\ p(1,0) & p(1,1) & p(1,2) \\ p(2,0) & p(2,1) & p(2,2) \end{bmatrix}$$

These values are illustrated via a flow chart in Figure 47.

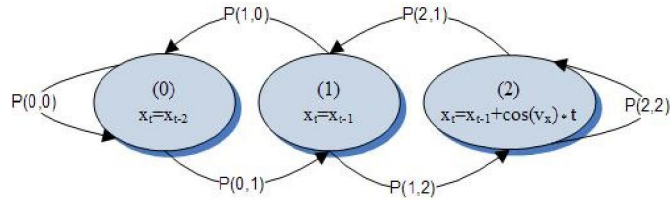


Figure 47: Flow chart of the Probabilistic Random Walk model

A mobile node may take a step in any of the four possible directions (i.e., north, south, east, or west) as long as it continues to move. In addition, the probability of the mobile node continuing to follow the same direction is higher than the probability of the Mobile Node changing directions. Lastly, the values defined prohibit movements between the previous and next positions without passing through the current location. This implementation produces probabilistic rather than purely random movements, which may yield more realistic behaviors. For example, as people complete their daily tasks they tend to continue moving in a semi-constant forward direction. Rarely do we suddenly turn around to retrace our steps, and we almost never take random steps hoping that we may eventually wind up somewhere relevant to our tasks (Camp, Boleng, & Davis).

To force nodes to move in map region, I made an innovative solution. When a node moves outside of region its state will be changed to opposite state, it means if its state is 0, it will be changed to 2 and vice versa. By this technique, the node comes back into region, and I can emulate the reflection effect.

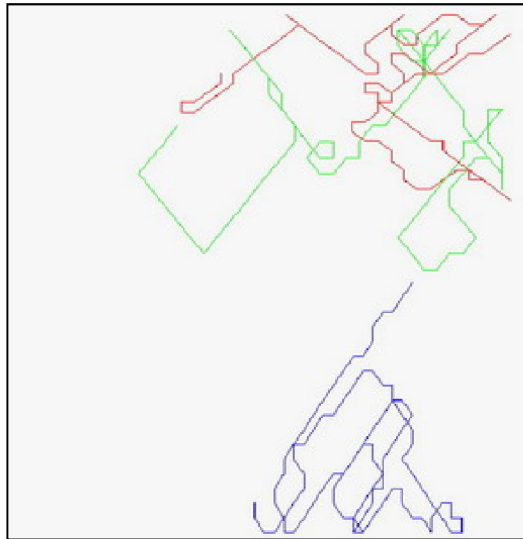


Figure 48: Traveling pattern of Mobile Nodes in Probabilistic Random Walk Mobility Model

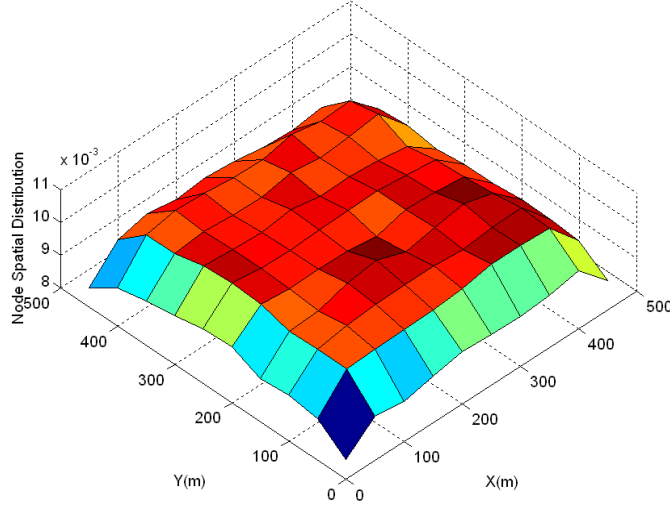


Figure 49: Spatial Node Distribution in Probabilistic Random Walk Mobility Model

#### 4-6-3 Exponential correlated model

Movement of nodes in this model exponentially relates to the last movement (speed and direction).  $\alpha$  is the memory factor.  $A$  is a parameter to set amplitude of randomness and  $r$ , and  $r'$  are random variables.

$$\begin{cases} s_{t+1} = \alpha * s_t + A * \sqrt{1 - \alpha^2} * r \\ d_{t+1} = \alpha * d_t + A * \sqrt{1 - \alpha^2} * r' \end{cases}$$

This model can be used as an internal model in other mobility models to simulate the behavior of people in a line or in the pause time at the end of each transition.

#### 4-7 Group models

In this category, node's motion depends on other nodes, usually the group leader. The group leader may have any mobility model, including group models. This is a big capability to generate creative traces.

##### 4-7-1 Nomadic

In line with the observation that the mobile nodes in MANET tend to coordinate their movement, the Nomadic<sup>7</sup> Model is proposed in (Hong, Gerla, Pei, & Chiang, August 1999). One example of such mobility is that a number of soldiers may move together in a group or platoon. Another example is during disaster relief where various rescue crews (e.g., firemen, policemen and medical assistants) form different groups and work cooperatively.

In the Nomadic model, each group has a center, which is either a logical center or a group leader node. For the sake of simplicity, I assume that the center is the group leader. Thus, each group is composed of one leader and a number of members. The movement of the group leader determines the mobility behavior of the entire group. Initially, each member of the group is uniformly distributed in the neighborhood of the group leader. Subsequently, at each instant, each node has a speed and direction that is derived by randomly deviating from that of the group leader. The movement in group mobility can be characterized as follows:

<sup>7</sup> It was called Reference Point Group Model (RPGM) in previous versions. But RPGM is a more general mobility model which most of the group models can be implemented using it.

$$\begin{cases} |V_{member}(t)| = |V_{leader}(t)| + random() * SDR * max\ speed \\ \theta_{member}(t) = \theta_{leader}(t) + random() * ADR * max\ angle \end{cases}$$

SDR is the Speed Deviation Ratio and ADR is the Angle Deviation Ratio. SDR and ADR are used to control the deviation of the velocity (magnitude and direction) of group members from that of the leader. Therefore, model parameters will be ADR, SDR, initial distance (members initial distance from the leader node), and group size which determines number of group nodes (Bai, Sadagopan, & Helmy, 2003).

I used Random Walk Mobility Model for motion behavior of leader node in each group as the default model, but you can select any model for the leader behavior. For example, you can use a Multi Model or a group model for leaders. The model's hierarchy can have infinite number of levels.

I also proposed a scheme to prevent nodes from passing the simulation regions. When the leader node reaches simulation region, it reflects using reflection rule I explained in Random Walk Mobility Model, but Group member nodes are bounded to simulation region.

This model adds two columns in the trace files: "GroupNumber" tells group number of the node, and "Is Leader" shows if the corresponding node is the leader of its group. While these information could be helpful, when you are combining group models, for example when you set a group model as the mobility model of the leader nodes, makes a complex trace. You can turn off this extra trace feature by unchecking the "extratrace" parameter.

Figure 50 shows traveling pattern of two groups with 5 nodes with average speed=20m/s, SDR=0.05, ADR=0.05, using Nomadic Mobility Model.

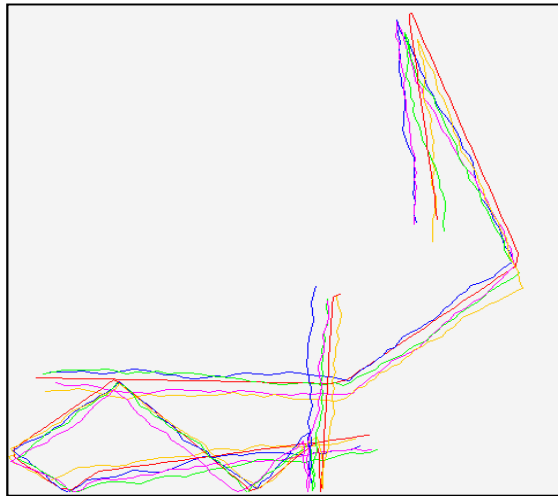


Figure 50: Traveling pattern of Mobile Nodes in Nomadic Mobility Model

As we can see in Figure 51, the node spatial distribution is similar to Random Walk Mobility Model because the leader node in each node travels with Random Walk Mobility Model.

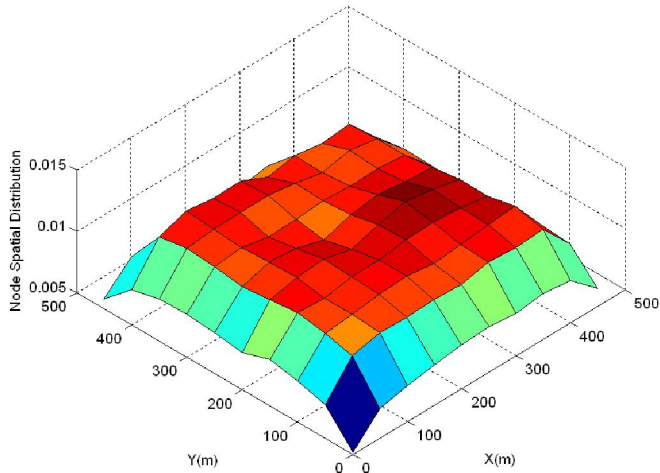


Figure 51: Spatial Node Distribution in Nomadic Mobility Model

The group models have an additional property for the ranges names “pergrouprange” that allows to run the range algorithm just inside each group. For example, suppose that you want only the group leaders to keep contacts using the MST algorithm. Then the nodes inside each group should keep contact with just members in the same group. Figure 52 shows a snapshot of the links when you follow this instruction:

1. Set range algorithm for Nomadic model to MST
2. Check the pergrouprange property
3. Set range algorithm for leaders model to MST

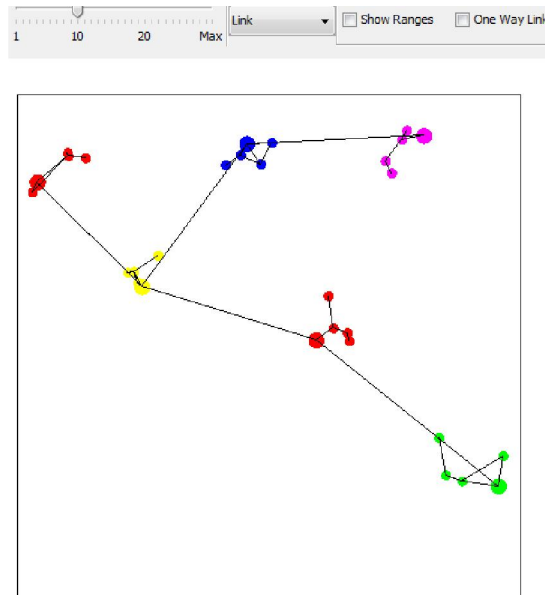



Figure 52: Per group range setting for a group model

Nomadic Model is a complex model because it has a model parameter (Group Leader’s model). As a result, in its map editor, you can see map handles for its map beside the map of its internal model. Moreover, there is an “offset” map handle  which states the relative position of map of internal model to the base map. Note that internal map could not have a point out of

the base map. This criterion has been implemented by asking the base map to check if it can include some border points of the map of internal model.

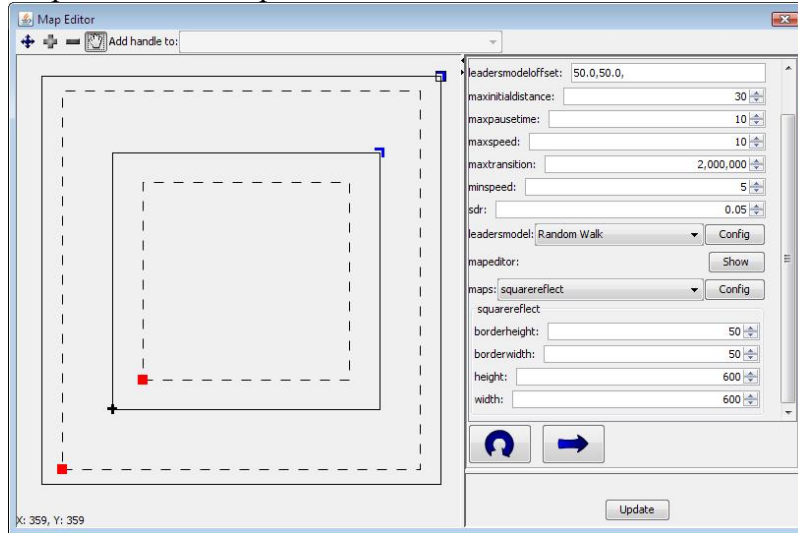


Figure 53: Group Models' Map Editor (with a Square Reflect map for its leaders' model)

#### 4-7-2 Pursue

This model is based on the Reference Point Group model, but the group members pursue the leader like cops follow a thief!

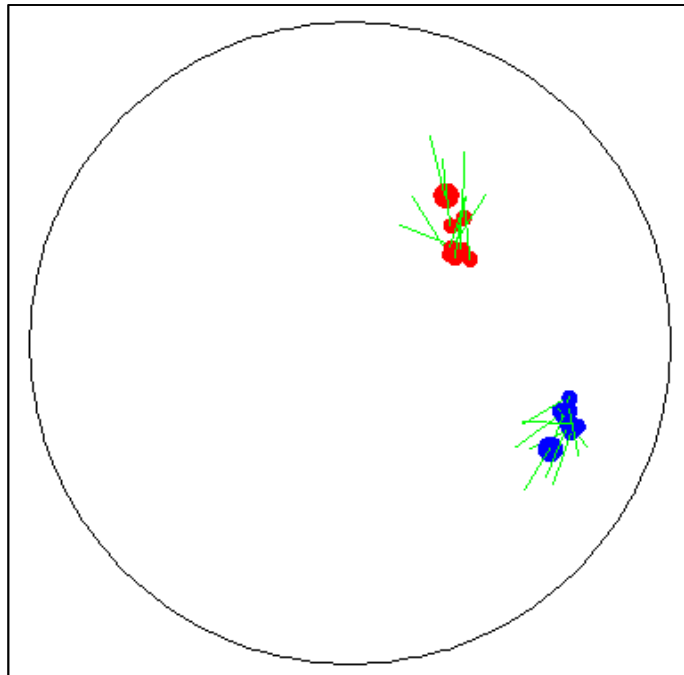


Figure 54: Pursue mobility model in a disk map



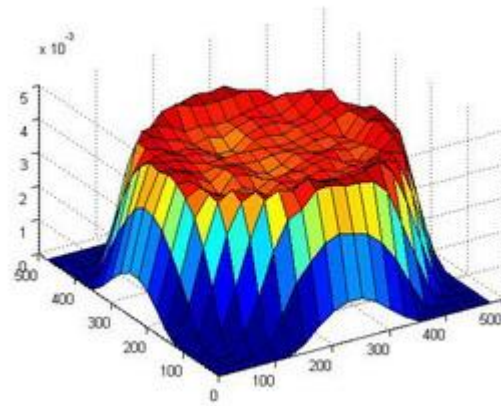


Figure 55: Spatial Node Distribution in Pursue model in a disk map

#### 4-7-3 String

The string model is based on the RPGM, but nodes make a line and follow the beforehand node movement. Thus, their movement will be like a string moving in the map.

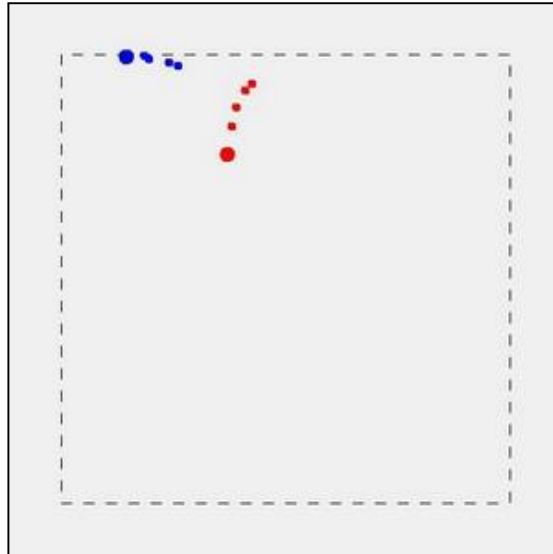


Figure 56: node's movement in the String mobility model

#### 4-7-4 Row

This model is based on RPGM model; however attempts to arrange the nodes in a row, near the leader of the group.

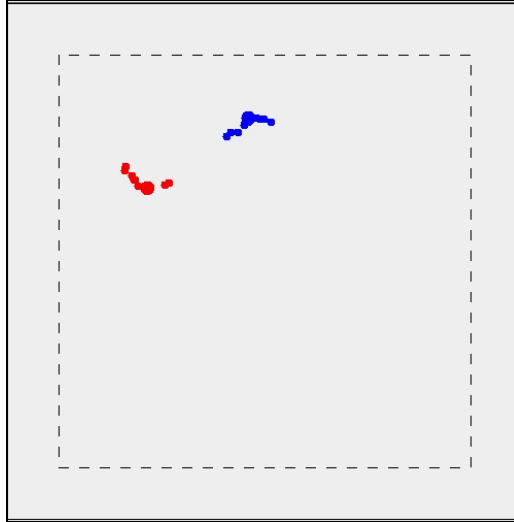


Figure 57: node's movement in the Row mobility model

## 4-8 Complex Models

Complex models are models that have internal models. While the group models are a kind of complex models, I have mentioned them in a different category to respect categorization in the literature.

### 4-8-1 Multi Model

You may have considered situations that not a single model, but multiple mobility models are present in a map. For example (Camp, Boleng, & Davis) :

- **In-Place Mobility Model:** Where the entire region is divided into several regions in which only nodes of a model move. Battlefield communication is a good example for this type because almost all of each side forces move in one region.

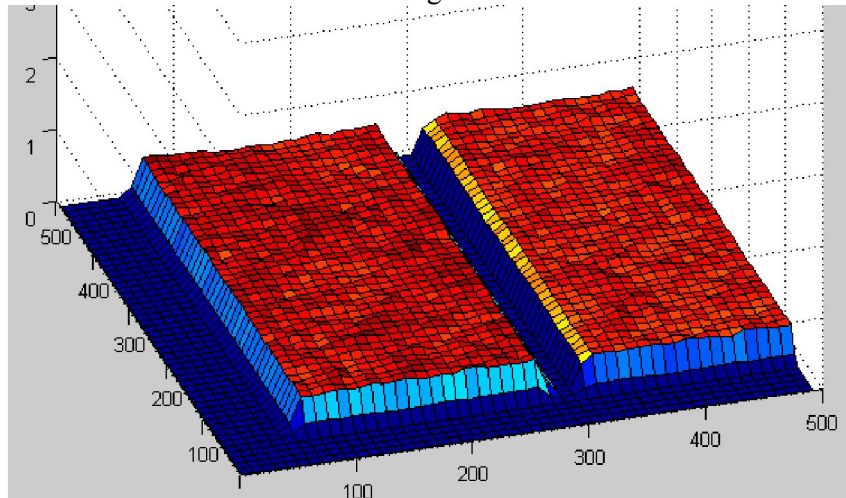


Figure 58: Multi Model In-Place Mobility Model with two Random Walk Model

- **Overlap Mobility Model:** There are situations where different groups move in a map with different models. For example, in a disaster scene, one might encounter different disaster relief teams, each of which has unique traveling patterns, speeds, and behaviors. On the other hand, there may be some injured people walking in lower pace.

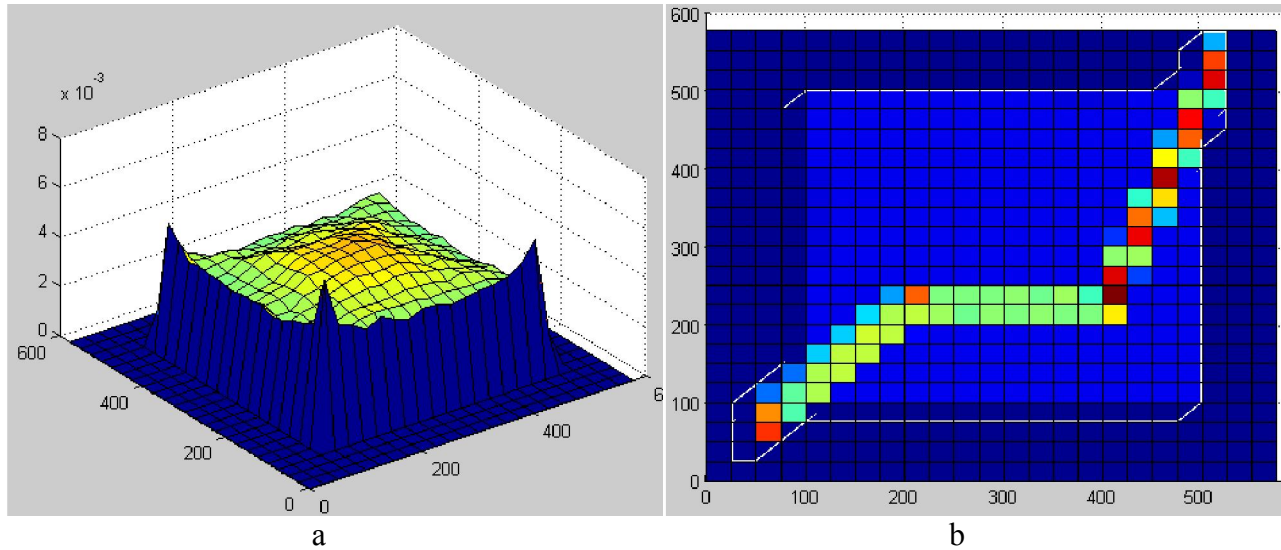


Figure 59: Multi Model Overlap Mobility Model. a) Random Direction and Random Way Point, b) Random Walk and FreeWay

- Convention Mobility Model:** In this type, simulation map is divided into regions while some nodes may cross the borders and move to other regions. A conference environment is an example for this model.

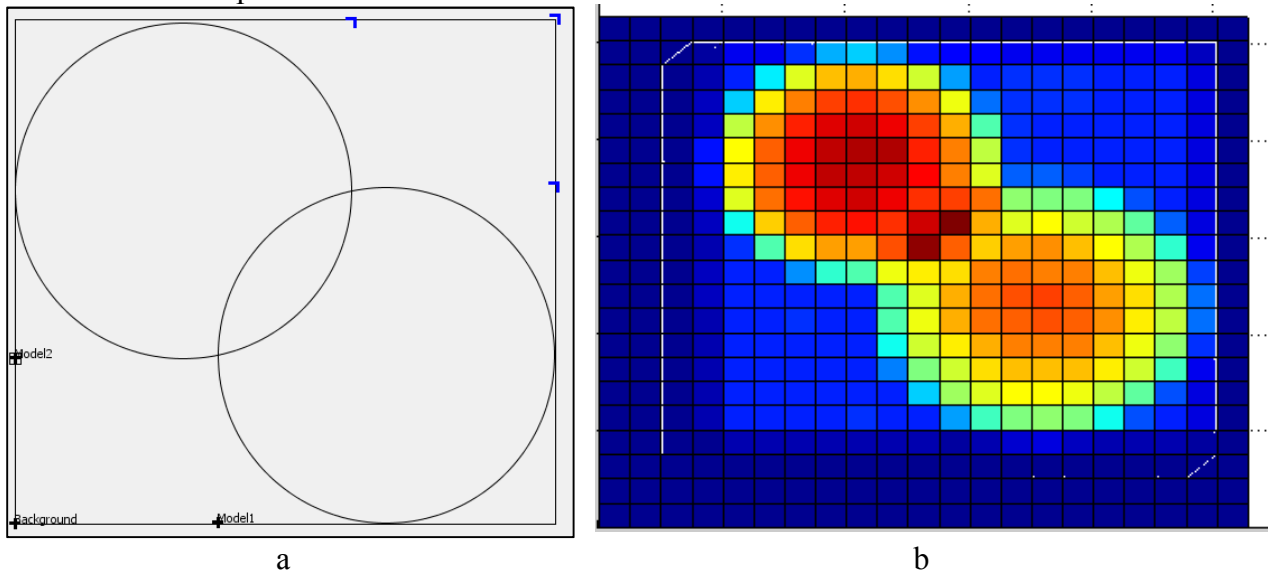



Figure 60: Multi Model Convention Mobility Model. A Random Walk model in background and two Random Way Point in Disk Reflect maps a) Map Editor window b) Node Distribution

This model is implemented as a complex model which means it has internal models. In order to add, remove, or configure internal models, you should click on the config button in front of “internal models” parameter. There is a template model in the bottom of the presented dialog. Configure the template and click on add button. Note that each internal model has an offset parameter which determines the offset of the internal model against the base map. Nodes will be distributed among the internal models based on the relative “*internalnodes*” weight of them.

Map editor of this model will show all the handles of its internal models beside their offset . There is a label for each map of internal models to help you follow the maps. Note that if the “*extend map*” parameter has been checked, its map will extend its size to cover all the internal models; otherwise, its size will restrict its internal models.

#### 4-8-2 ThreeDizer Model

As you may noticed, all the mentioned models run in a 2D environment. However, nodes move in a 3D space in the real world. This concernment becomes more important in large maps where there is a network considerable altitude. This model, as a complex model, gets a model and a 3D map as its parameters and tries to find the altitude of a node in the map. 3D map as the base map in 3D model has its own parameters:



- *gridsize* specifies maximum width and height of grids of the 3D map. Note that grid points will be always on the map and have zero altitude by default. You can change their Z value like user specified points. See the *points* section.
- *CachePrecision*: The 3D map does not calculate the Z position of a node if it was calculated before. CachePrecision parameter configures the process of finding the previous node’s Z position. If a point in the map with CachePrecision distance from current node’s position has a calculated Z factor, the 3D map will return its value; otherwise, it will calculate the Z factor and adds it to the cache.
- *Points*: Each point defines a point in the 3D space which has x, y, and z values. As the grid points will be always present in the map, you can only change their z parameter’s value by defining a point just on the grid point coordination. Non-grid points are also supported.

Note that in order to resimulate the simulation scenario, there is a file parameter, “mapsetting output”, which tries to export the 3D map settings. Set the output file address if want to use 3D resimulation.

The following paragraphs will describe how this model works. You may skip these paragraphs if you do not want to know the details.

In the initialization step, after loading the points and setting the grid points, the 3D map creates blocks using each four grid points. In order to find the Z value of the position of a node, the following steps will run:

1. Check if the current node is on a predefined point;
2. Check if the current node is near enough to a cached position;
3. Find the triangle that surrounded the current point (The grid points helps restrict the computation overhead of this process.)
  - a. Check if all the triangle vertices have equal Z, use it
  - b. Otherwise, create a 3D plane using the three points and find out the z value

In the current step, I use the 2D map editor for this model. Each point in the map has its own map handle in the map editor. While the grid points  could not be dragged in the map, you can add, remove, or move additional points . Each 3D point has a Z parameter which can be configured using the handle configuration panel. Note that the size of the 3D map is determined by the size of 2D map in the internal model and the offset handle cannot be moved. Besides, neither an additional point nor a grid point with non-zero Z parameter can be out of the map.

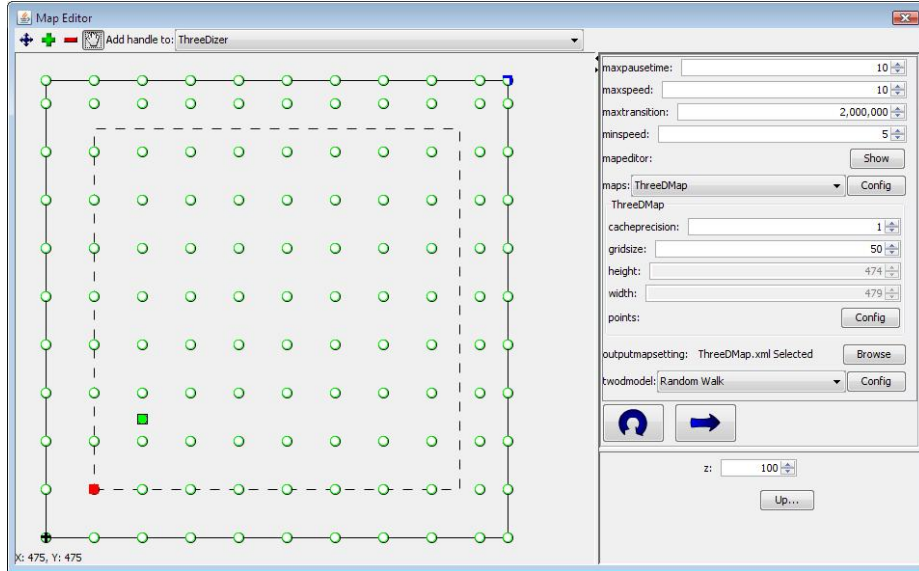


Figure 61: ThreeDizer model Map Editor window (using Random Walk)

**NOTE:** in the current implementation the horizontal speed of a node does not depend on the slope of the ground. I know that it may make threeD model useless, but it is just the start.

#### 4-9 File Model

File model is introduced to enable user to use a trace file as an internal model either for group models or complex ones. The model needs a file in its parameter list.

### 5 Evaluators

All evaluators have a parameter named “sample time”. It means how often the evaluator runs and measures the metric<sup>8</sup>. First of all, let’s define some preliminary signs:

1.  $D_{i,j}(t)$  : Euclidean distance between nodes  $i$  and  $j$  at time  $t$ ;
2.  $N$ : Number of mobile nodes;
3.  $\theta_i(t)$ : Angle made by  $\vec{V}_i(t)$  (velocity vector) at time  $t$  with the X-axis;
4.  $SR(\vec{a}(t), \vec{b}(t'))$ : Speed Ratio (SR) between the two vectors:  $\frac{\min(|\vec{a}(t)|, |\vec{b}(t')|)}{\max(|\vec{a}(t)|, |\vec{b}(t')|)}$
5.  $RD(\vec{a}(t), \vec{b}(t'))$ : Relative Direction (RD) between two vectors:  $\frac{\vec{a}(t) \cdot \vec{b}(t)}{|\vec{a}(t)| \times |\vec{b}(t')|}$

#### 5-1 Average lifetime

Average lifetime evaluator calculates average duration when a node in the network is active. It uses the “active” field in the traces.

#### 5-2 Average power/range

The evaluator calculates the average range and power of the nodes in the network.

#### 5-3 Average distance

It is a very simple evaluator that calculates the average distance between a pair of nodes.

<sup>8</sup> Mobility and Network evaluators have been merged from the version 3

#### 5-4 Clustering coefficient

This measure is proposed to evaluate small-world properties. It is calculated by the number of links among the neighbors of a node normalized by the number of them. The value of this evaluator shows redundancy in network as while there are communication links among neighbors of a node it cannot play a big role in the communication channel between them.

#### 5-5 Disconnection ratio

This evaluator calculates the ratio of number of seconds that the network is not connected to the simulation time.

#### 5-6 Interference

Although having big ranges in wireless networks improves the connectivity of the network, it increases the interference. If we suppose that the transmitting node uses RTS packets and receiving node uses CTS ones to tell their neighbors to be silent during communication, we can measure interference as the total number of neighbors between two communication parties. So I have measured interference by finding the maximum number of neighbors between two nodes in the network and averaging it over the simulation period.

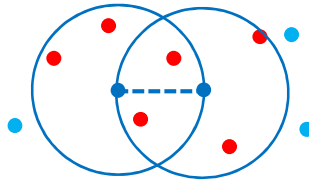


Figure 62: How much interface a link creates in the network: Count the number of nodes in the two circles with radius equal to the distance between the two nodes

#### 5-7 Link evaluator

This evaluator produces:

1. **Average Link Duration:** This evaluates how long a link is connected before it disconnects. To formulate it, we can see it as the total lifetime of each link between two nodes over the number of disconnection in that link + 1.
2. **Neighborhood instability:** Instability in the neighbors of a node can cause unreliability in network services in some protocols. I measure it by the number of link disconnection for each node. If you divide it by the simulation time, you can get Topology Change Ratio (TCR) measure that shows the dynamics of the topology of network.
3. **Intermeeting Time:** This evaluator measures the average time that two nodes should wait to see each other. It is somehow the complement of the link duration value.

#### 5-8 Network diameter

This evaluator has been defined in (Theoleyre, Tout, & Valois, Feb. 2007). It calculates the longest path between two nodes in the network using Floyd-Warshal algorithm. The network diameter is representative of the average length of the paths of the network. Thus, a mobility model which creates a low diameter will for example improve routing performances, minimizing interferences. In particular, we can verify that the network diameter represents a stable value, not changing importantly when the nodes are moving (It can be a new evaluator to be implemented).

### 5-9 Node degree

This evaluator calculates two values: 1-way node degree and 2-way node degree. 1-way node degree presents average number of nodes that each node can see whether they can see it too or not. However, the second one calculates the average of number of 2-way connections between nodes.

### 5-10 Number of disconnection

This evaluator shows the number of disconnections of the whole network graph in the simulation.

### 5-11 Relative speed

In fact, it is average relative speed according to (Bai, Sadagopan, & Helmy, 2003). I used their definition for this evaluator. Relative speed is

$$RS(i, j, t) = \begin{cases} 0 & ; D_{i,j}(t) \geq 2R \\ |\vec{V}_i(t) - \vec{V}_j(t)| & ; otherwise \end{cases}$$

Therefore the average relative speed will be the value of  $RS(i,j,t)$  averaged over node pairs and time instants. This evaluator will quantify the speed of nodes relative to each other. It may be important for selecting a protocol in a MANET.

Parameters:

- 1) R: relative speed range

### 5-12 Repetitive behavior

It is supposed (Theoleyre, Tout, & Valois, Feb. 2007) that nodes come back to a region that they were before and make a kind of repetitive behavior. One of the ways to measure this behavior is checking how much time a node moves around a point it were before. To do this, the evaluator calculates the average time that a node moves in a disk with radius equals to parameter *range* around the point captured every *initialpointselectionsampletime*. If the value of latter parameter was zero, the reference point does not change through the evaluation and will be the initial location of each node.

Parameters:

- 1) range: moving how far from the reference point is assumed repetitive behavior
- 2) initialpointselectionsampletime: how often the reference point might be changed

### 5-13 Spatial dependency

This measure (Bai, Sadagopan, & Helmy, 2003) will show how much the velocities of two nodes that are not too far are similar.

$$D_{spatial}(i, j, t) = \begin{cases} 0 & ; D_{i,j}(t) \geq 2R \\ RD(\vec{V}_i(t), \vec{V}_j(t)) * SR(\vec{V}_i(t), \vec{V}_j(t)) & ; otherwise \end{cases}$$

Similar to relative speed, I calculated the average degree of spatial dependency in this evaluator which is the value of  $D_{spatial}(i,j,t)$  averaged over node pairs and time instant. It shows how the nodes move in a group or imitate movement of each other.

Parameters:

- 1) R: Max range to have spatial relation

### 5-14 Temporal dependency

It is a measure (Bai, Sadagopan, & Helmy, 2003) describing the similarity of the velocities of a node during a period of time.

$$D_{temporal}(i, t, t') = \begin{cases} 0 & ; |t - t'| \geq T \\ RD(\vec{V}_i(t), \vec{V}_i(t')) * SR(\vec{V}_i(t), \vec{V}_i(t')) & ; otherwise \end{cases}$$

The evaluator calculates the value of  $D_{temporal}(i, t, t')$  averaged over nodes and time instants. Thus, by means of that, we can find out the correlation of node's movement in different time instants.

Parameters:

- 1) T: Maximum time gap between two related movements.

### 5-15 Transition evaluator

We can see most of the mobility models as a transition model in which each model move in one direction with a fixed speed for a specified duration. This evaluator tries to find the average values for:

1. Transition Time: duration of the transition
2. Transition Length: the distance each node travels in each transition
3. Transition Direction Change: direction change from the last transition
4. Transition Speed
5. Number of Transition

While these are only the average values, you can edit its class to extract other statistical values. However, the important point here is how this evaluator recognizes a transition:

This tries to detect a transition by the change in the speed and direction. The evaluator has two additional parameters which define the threshold for changes in one transition. If a node has changes in its speed or direction more than this threshold, the algorithm considers that it started a new transition. You can set a threshold to zero to ignore the changes in speed or direction of a node.

## 6 File formats

### 6-1 Mobility traces

Trace formats are also a pluggable module! Everyone can add some new formats to the program by implementing two classes; See 7-4 . I have already implemented three trace types.

#### 6-1-1 Plain text

The plane text contains a parameter line, a column header line, and values sorted based on time and node number. Only have I written Time, Node number, Position, Speed, and Direction, but there are many other mobility characteristics calculated in the program. A tab character separates the columns.

tracewriter=text models=RPGM selected=RandomWalk walkingtime=20 . . .						
Time	Node	PositionX	PositionY	Speed	Direction	Angle
1	0	276	27	9.45	1.75	
1	1	263	37	10.99	1.46	
1	2	262	42	7.96	1.53	
1	3	270	33	11.27	1.5	
1	4	305	39	9.94	2.03	



1	5	289	167	6.43	1.76
1	6	292	175	5.35	1.76
1	7	305	152	7.71	2

Code 1: a plain text mobility trace sample

### 6-1-2 NS2

As NS2 only reads next time, position, and speed, all other information have been indicated as a comment text in each line. This format is not mature now and needs feedbacks to be revised.

```
# models=ThreeDizer      selected=Random Walk      walkingtime=20
# Time      Node      PositionX      PositionY      Speed      Direction Angle      PositionZ
$ns_ at 1 "$node0 setdest 134 192 9.14" ;# 1.23      0
$ns_ at 1 "$node1 setdest 317 210 8.17" ;# 5.52      0
$ns_ at 1 "$node2 setdest 77 189 6.44" ;# 0.97      0
$ns_ at 1 "$node3 setdest 349 216 5.11" ;# 5.42      0
```

### 6-1-3 XML

The XML format also includes the information presented in the plain text, but in another format. “Header” tag contains the parameters in “Parameters” child tag and column headers in “Attributes” one. See Figure 63

```
<?xml version="1.0" encoding="utf-8"?>
<Simulation>
  <Header>
    <Parameters>
      <Parameter name="title" value="untitled" />
      <Parameter name="filenamegenerator" value="simple" />
      <Parameter name="seed" value="randomDirection" />
      <Parameter name="numberiterate" value="true" />
    </Parameters>
    <Attributes>
      <Attribute name="Time" />
      <Attribute name="Node" />
      <Attribute name="PositionX" />
      <Attribute name="PositionY" />
      <Attribute name="Speed" />
      <Attribute name="Direction Angle" />
    </Attributes>
  </Header>
  <Data>
    <t>1,0,385,317,8.83,3.36</t>
    <t>1,1,297,159,6.98,6</t>
    <t>1,2,336,302,8.57,2.96</t>
    <t>1.3.317.172.7.93.5.03</t>
```

Figure 63: XML mobility trace sample

## 6-2 Power traces

Power traces can be in plain text or XML format. Nevertheless, it should have two additional values: Range and Active. Active column values show if the node in that moment was active or not. Note that node’s activity impacts on some evaluators output. Figure 64 shows a plain text trace sample.

```

height=500  init=10.0  maps=SquarReflectMap  maxpausetime=10  maxsimulationtime=1000  maxspeed=10
maxtransition=2000000  memoryfactor=0.8  minspeed=5  models=Markov  width=500
Time  Node  PositionX  PositionY  Speed  Direction  Angle  Range  Active
1  0  326 89  9.47  3.01  100 true
1  1  319 78  7.75  2.48  100 true
1  2  58 365 9.78  6.03  100 true
1  3  302 86  6.72  3.63  100 true
1  4  58 389 8.14  5.81  100 true
1  5  31 82  5.72  0.27  100 true
1  6  246 367 6.43  3.68  100 true
1  7  147 187 5.3 2.17  100 true

```

Figure 64: Plain text power trace

### 6-3 Evaluation file

Evaluation files are similar to the mobility traces, but I have not implemented the first parameter row yet. They are in a well-formed format to import in other programs such as Excel easily.

```

FileName  group  Temporal  Dependency  Spatial  Dependency  RelativeSpeed  maxspeed  minspeed  models
0-10,0-5,0 Markov1.txt  1  0.28  0.03  1.76  10  5  Markov
0-10,0-5,0 outputtrace1.txt  2  0.22  -0.01  4.38  10  5  Manhattan
0-10,0-5,0 RandomDirection1.txt  3  0.48  -0.01  2.27  10  5  RandomDirection
0-10,0-5,0 RandomWaypoint1.txt  4  0.39  0  2.52  10  5  RandomWayPoint
0-10,0-5,0 RPGM1.txt  0.14  5  0.77  0.54  10  5  RPGM
0-20,0-15,0 Markov3.txt  0.14  1  0.04  1.59  20  15  Markov
0-20,0-15,0 outputtrace3.txt  2  0.09  0.01  7.9  20  15  Manhattan
0-20,0-15,0 RandomDirection3.txt  3  0.05  -0.01  5.57  20  15  RandomDirection
0-20,0-15,0 RandomWaypoint3.txt  4  0  0.02  6.2  20  15  RandomWayPoint

```

Figure 65: Evaluation file format

## 7 How to extend?

The user may want to extend the program in the following directions:

1. New mobility model
2. New map for current models
3. New file format for trace output
4. New evaluators (Mobility based or Network based)
5. New model classifier algorithm
6. New process

Before talking about the specific procedure to do each of the preceding cases, it is necessary to know about the common structure among these sections.

### 7-1 Parameterable Hierarchy

I named an object that can be parameterized as “Parameterable”. So each parameterable object can have various parameters. For example, an evaluator can have a double parameter and a Boolean one. I have implemented various simple parameters such as double, integer, Boolean, integer array and so on. Each implementation helps us not be worry about the validity of input configuration file or the representation of the parameter in the forms.

The next step is trying to have a parameterable as a parameter of another (Remember Composite design pattern (Gamma, Helm, Johnson, & Vlissides, 1994)). For example, suppose the map of a model: Each model can have a map and both are parameterable. I have defined four relationships between parameterables:

1. One-to-One
2. One-to-One (selecting from some options)
3. One-to-Many (selecting from some options)
4. One-to-Many (constructing from templates (Prototype design pattern (Gamma, Helm, Johnson, & Vlissides, 1994)))

The first relation is mostly used when a parameterable needs a file parameter and is implemented by *ParameterableParameter* class. In the second one, the user can select the parameterable parameter from a list of options. For example, the relation between models and their maps is like this one. There are two implementations for this relation. *SelectOneParameterable* which loads and creates all the choices when is loaded and *LazySelectOneParameterable* which do not loads its choices until it is needed. The latter is used when there is a loop in the hierarchy. For instance, when a model has a model parameter!

One-to-Many relation defines the relation when a parameterable object needs many instances of a kind of parameterable. For instance, an evaluation can have different evaluators active or a complex model can have various internal models. The former forces the user to select from choices while they are configurable, but the latter provides the user with some prototypes which can be instantiated and are configurable. Both cases are implemented by *MultipleSelectParameter* class and are recognized by the value of *variablenumber* parameter. Note that while these classes can have parameters they are not parameterable.

Now you can find out that there is a hierarchy of parameterables in the software. Figure 66 shows how this hierarchy looks in the Mobility Generator module.

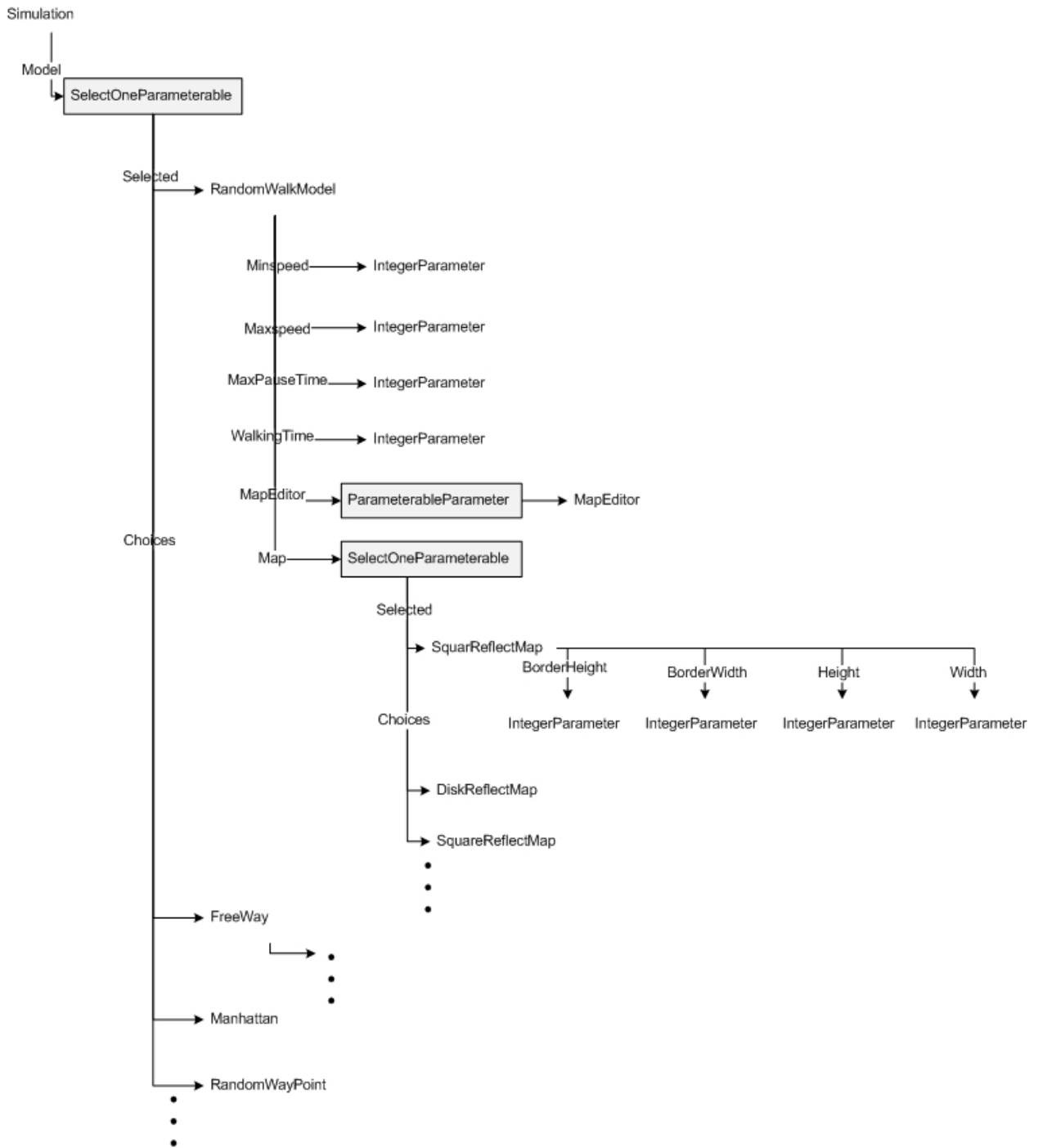


Figure 66: A real parameterable hierarchy in the software

In the form generator a default UI will be created for a parameterable object. However, a parameterable can have its own customized UI like *VariableParameter* if it implements *GraphicalStandAloneObject* interface. This interface connects logic section to the UI part which is itself a hierarchy of *NewUIParameter* objects.

The last part is talking about the data section to complete the three tier model. Each hierarchy is defined in an XML file, structure of which is defined by the following DTD text. This DTD

file says that the root element of the XML file is a `rootcomposite` tag which can have one default internal tag and multiple `parameterable` tags. Each `parameterable` tag represents a `parameterable` object or one of the relation types. A `parameterable` tag can have internal `parameter`, `composite` and `parameterable` tags. A `parameter` tag is used for initializing simple parameters. Note that the class attribute is the complete address of the class of parameters and parameterables that are in the CLASSPATH. `Composite` tags define a collection of parameterables and used mostly in relations. It is noteworthy that a composite tag can reference to a file and import its internal tags. This mechanism lets you to break a large file. At last, the `default` tag prevents us from repeating a common configuration among parameterables of a composite list. However, you can override the default configuration by redefining the configuration. For example, it is not necessary to define the map choices for all mobility models by defining the default choices in a `default` tag. Figure 67 presents the file dependency of each module. These configuration files are located in the resource folder.

```

<!ELEMENT rootcomposite (default?,parameterable*)>
<!ATTLIST rootcomposite
    name CDATA #REQUIRED
    file CDATA #IMPLIED
    lazyLoadableID CDATA #IMPLIED>
<!ELEMENT composite (default?,(parameterable*|parameter*|composite*))>
<!ATTLIST composite
    name CDATA #REQUIRED
    lazyReference CDATA #IMPLIED
    lazyLoadableID CDATA #IMPLIED
    import CDATA #IMPLIED>
<!ELEMENT parameterable (parameter|composite|parameterable)*>
<!ATTLIST parameterable
    name CDATA #REQUIRED
    class CDATA #REQUIRED>
<!ELEMENT default (parameter|composite|parameterable)*>
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
    name CDATA #REQUIRED
    value CDATA #REQUIRED
    class CDATA #IMPLIED>

```

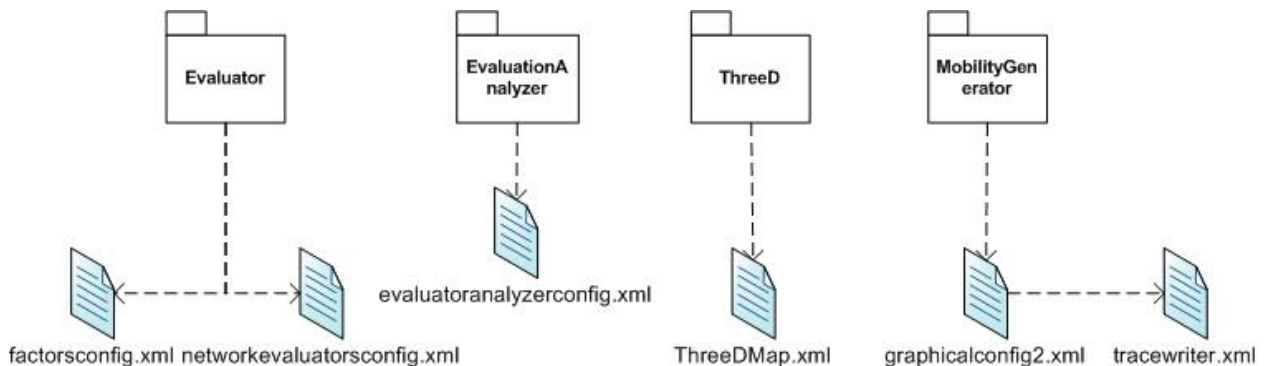


Figure 67: Configuration files of modules

Note that the following sections are very abstract and for a true description about the functionality of each class and method, see the javadoc documentations.

## 7-2 Add a new mobility model

As you may know by now, each mobility model is a parameterable object. So if you want to add any parameter to your model you must override the following methods:

```
public void setParameters(java.util.Map<String, Parameter> parameters) throws
InvalidParameterInputException
public java.util.Map<String, Parameter> getParameters()
```

By default a model has a Map and a MapEditor internal parameter. Don't forget to override the toString() method, too. Two basic methods that define all the functionality of a model are:

```
void initNode(GeneratorNode node) throws ModelInitializationException
void getNextStep(double timeStep, GeneratorNode node)
```

Each model must define the initial position, direction, and speed of a node. Then, on each simulation tick, it must update the location of that node. Note that it is the responsibility of the model itself to keep the relation between the nodes if there is any (for example in group models). If the order of initializing and updating nodes are important override initnodes() and updateNodes() methods.

If your model needs to write extra traces for each node, you must override print() and getLabels() methods. If your model needs to handle the handles in the MapEditor itself, it must implement MapHandleSupport interface. For an example, see RandomGroupModel class. If your model needs to change the way nodes are painted override getNodeNodePainter method and return your NodePainter object for each node.

To register your new model in the system, just open the Mobility Generator config file and in the parameterable tag with name attribute "model" add the following line:

```
<parameterable name="MODELNAME (TOSTRING() METHOD OUTPUT)" class="CLASS ADDRESS">
</parameterable>
```

## 7-3 Add a new map

What is a map? Roughly, a map is only a painting on a canvas! It is a parameterable that controls this painting and has some handles to be drawn in the MapEditor (so implements MapHandleSupport interface). If a map implements PassiveMap interface, it can validate the position of a node and the destination of its current transition. However, if it implements ReflectiveMap interface it can find out if the node has hit the border, so the model can decide upon this event. There are some general maps like DiskReflectiveMap or SquareReflectiveMap while Geographical models such as Manhattan has their own specific map. If a map can include another map it must implement IncludingMap interface and it only can include maps implementing IncludableMap interface. Note that the painting procedure is called through the Model class so if in any case you need to change the appearance of a map based on the state of the model you must override the paintBackground method in your model.

To register the map for a specific model you must add the following code in the config file inside the model tag:

```
<parameterable name="maps"
class="edu.sharif.ce.dml.common.parameters.logic.complex.SelectOneParameterable">
    <parameter name="selected" value="THE MAP NAME"/>
    <composite name="choices">
        <parameterable name="MAPNAME (TOSTRING() METHOD
OUTPUT)" class="CLASS ADDRESS">
            <parameter name="PARAM1" value="VALUE1"/>
        </parameterable>
    </composite>
```

```
</parameterable>
```

#### 7-4 Add a new trace output/input

Current trace formats are defined in the Common module in `edu.sharif.ce.dml.common.data.trace` package. To add an output trace format you must create a class that extends `TraceWriter` class. It is a parameterable object so it can have parameters. Its most important methods are `init` and `writeTrace` methods functionality of which is obvious. Add a new `FileFilter` in `edu.sharif.ce.dml.common.data.trace.filter.FileFilter` class to be used in file dialogs and register the new parameterable object in the `tracewriter.xml` file like this:

In a parameterable tag with `name="tracewriter"` under its composite tag add this:

```
<parameterable name="NAME" class="TRACEWRITER CLASS"/>
```

If the new file format is a type of mobility traces, the new defined `FileFilter` class must be a child of `TraceFilter` class. As a result, you must define how the program can parse this file format. All parsers must extend `LoadingHandler` class which has a type that specifies the type of objects that they produce by parsing. Notice that there are two types of `LoadingHandler` in the system: `BulkLoadingHandler` which loads all the file at once and `StreamLoadingHandler` which parse the input file object by object. For Example, XML file format, because they need to be parsed, are of bulk one. However, text format is of stream one and needs less memory.

#### 7-5 Add a new evaluator

Evaluators are also parameterable objects and you must implement `evaluate()`, `reset()`, `toString()`, `getLabels()`, and `print()` methods for a new one. An evaluator can create multiple columns in the output file. Each evaluator has a type that defines which object it can evaluate. I have implemented three types of evaluators: Mobility evaluators that do not need communication range in the traces, network evaluator that need to know the range, and evaluators that evaluate a classification. Register your new evaluator in the appropriate file like preceding parameterables.

#### 7-6 Add a new model classifier algorithm

I think it is not necessary any more to tell you that classifier algorithms are parameterable objects and can be registered in the `evaluationanalyzerconfig.xml` file. To define a new classifier algorithm you must extend `EvaluationAnalyzer` class if it needs learning data it must implement `LearnableAnalyzer` interface, too<sup>9</sup>.

The most important methods are `setEvaluationData()` and `getEvaluationGroups()`. There is also an additional method, `learn()`, for learning algorithms. These algorithms work with `Evaluation` objects containing `EvaluationRecords` and `EvaluationRecordGroup` objects presenting a class of records. Each `EvaluationRecord` object is a line in the evaluation file.

#### 7-7 Add a new process

What is a process? I define a process as a time-consuming procedure that needs to load a file, do something, and usually write the output in a file. First you must have a `LoadingHandler` to load the trace files. Then, write a `UsingHandler` (it has `Stream` and `Bulk` type) which defines what to do when an object or the whole file has been loaded. At the end, you connect them using an appropriate `User` and `FileLoader` objects (see Figure 68). The following code segment shows an example of creating and running a `User` object:

---

<sup>9</sup> Classifying algorithms which do not use learning data are usually called clustering algorithms.

```
User<EvaluationRecord> user = new BulkUser<EvaluationRecord>(dataHandler, file,
(BulkLoadingHandler<EvaluationRecord>) trackLoadingHandler);
user.run();
```

The run method, runs loading the file until it ends and calls the corresponding methods of the UsingHandler object. So you should write the code you want to do base on the loaded data in the UsingHandler.

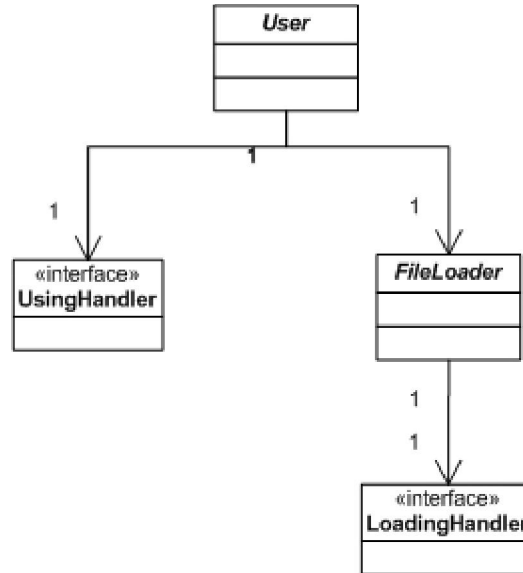


Figure 68: The structure of a process

To separate the swing (UI) thread from the process thread when the process is time-consuming, you can create a subclass of `MultiTaskSwingWorker` class that helps you present the progress form and let the UI be responsive during processing. The class is a subclass of `SwingWorker` that has `execute()` method for running and `publish()` method for publishing the result of each step. You must implement `doWork()` method in it. I have implemented most of these worker classes in the UI packages as an inner class of the forms. See an example in the code for more information.

See `edu.sharif.ce.dml.mobisim.evaluator.ui.MobilityEvaluationFrame` for an easy example.

## 8 References

- Bai, F., Sadagopan, N., & Helmy, A. (2003). The Important framework for analyzing the Impact of mobility on performance Of Routing Protocols for Adhoc Networks. 383-403.
- Bettstetter, C., & Wagner, C. (Mar 25-26, 2002). The Spatial Node Distribution of the Random Waypoint Mobility Model. *P-11*, pp. 41-58. Ulm, Germany: in Proc. German Workshop on Mobile Ad-Hoc Networks (WMAN), GI Lecture Notes in Informatics.
- Broch, J., Maltz, D. A., Johnson, D. B., Hu, Y.-C., & Jetcheva, J. (October 1998). A performance comparison of multi-hop wireless ad hoc network routing protocols. in Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking(Mobicom98), ACM, .
- Camp, T., Boleng, J., & Davis, V. (n.d.). A Survey of Mobility Models for Ad Hoc Network Research. 2(5), 483-502, 2.



- Chiang, C. (1998). *Wireless Network Multicasting*. Los Angeles: PhD thesis, University of California.
- Chong, I. R. (2008). On the Levy-Walk Nature of Human Mobility. (pp. 924-932). INFOCOM.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.
- Hong, X., Gerla, M., Pei, G., & Chiang, C.-C. (August 1999). A group mobility model for ad hoc wireless networks, in Proc. ACM Intern. Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM).  
<http://www.jsc.nildram.co.uk/>. (n.d.).
- Khaledi, M., Rabiee, H. R., & Khaledi, M. (2010). Fuzzy Mobility Analyzer: A Framework for Evaluating Mobility Models in Mobile Ad-Hoc Networks. IEEE Wireless Communications and Networking Conference (WCNC).
- L. Breslau, D. E. (2000). Advances in network simulation. 33(5), 59--67.
- Liang, B., & Haas, Z. (March 1999). Predictive distance-based mobility management for PCS networks. In Proceedings of the Joint Conference of the IEEE Computer and Communications Societies (INFOCOM).
- Mousavi, S. M., Rabiee, H. R., Moshref, M., & Dabirmoghaddam, A. (2007). Mobility Pattern Recognition in Mobile Ad-Hoc Networks. *ACM Mobility Conference*, (pp. 302--309). Singapore.
- Royer, E. M., Melliar-Smith, P. M., & Moser, L. E. (June 2001). An Analysis of the Optimum Node Density for Ad hoc Mobile Networks. Helsinki, Finland: in Proceedings of the IEEE International Conference on Communications(ICC).
- Theoleyre, F., Tout, R., & Valois, F. (Feb. 2007). New metrics to evaluate mobility models properties. 2nd International Symposium on Wireless Pervasive Computing, 2007. ISWPC 07. .
- Wang, N.-C., & Chang, S.-W. (2005). A reliable on-demand routing protocol for mobile ad hoc networks with mobility prediction. 29, pp. 123-128.

## 9 Appendix 1: Prerequisite installation

### 9-1 Java

In order to run the program you need JRE/JDK while compiling the codes needs JDK.

#### 9-1-1 Windows

After installing appropriate package for example in “C:\program files\Java\jdk1.6.0\_01”, you should set your JAVA\_HOME environment variable and update PATH variable.

1. Right click on “My computer” icon and select the properties item
2. Click on “Advanced” tab (in Windows Vista/7 first click on “Advanced system settings”.
3. Click on “Environment Variables” button
4. On the “System variables” click on the new button and define a new variable have its name “JAVA\_HOME” and value “C:\program files\Java\jdk1.6.0\_01”.

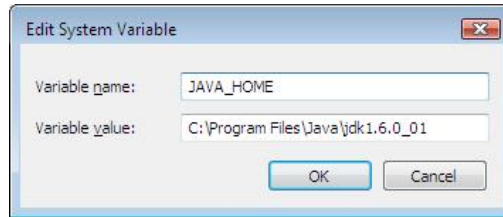


Figure 69: Defining JAVA\_HOME environment variable

5. Edit the “path” variable and add “;%JAVA\_HOME%\bin” at the end of it. (pay attention to the semi-colon sign)

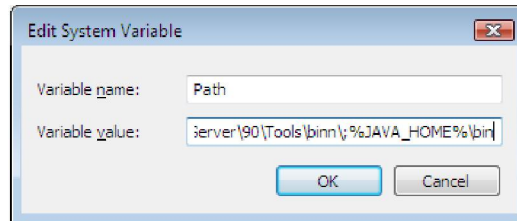


Figure 70: Redefining PATH environment variable

6. Click on OK button in all the opened windows as the final step. Note that if you have a command prompt window open, you should close and reopen it to have its variables updated

Open a command line window and type “java”. If it does not complain about not finding the command, it is all right.

```
Usage: java [-options] class [args...]
           (to execute a class)
 or java [-options] -jar jarfile [args...]
           (to execute a jar file)

where options include:
 -client  to select the "client" VM
 -server  to select the "server" VM
 -hotspot is a synonym for the "client" VM [deprecated]
           The default VM is client.
```

Figure 71: java program output, showing that it is configured correctly

### 9-1-2 Linux

1. If you are using Ubuntu or Debian use synaptic package manager in order to install a java 1.5 or 1.6 implementation (OpenJDK for example<sup>10</sup>). For other Linux flavors, find its package manager or the install the .rpm file.
2. Use locate command to find the java program location

```
locate /rt.jar
```

The command output should be something like:

```
/usr/lib/jvm/java-6-sun-1.6.0.0.7/jre/lib/rt.jar
```

Therefore, your java home path is “/usr/lib/jvm/java-6-sun-1.6.0.0.7/”

3. Edit “.bashrc” file in your home folder and add the following line to its end. Replace <JAVA\_PATH> with what you found in previous step. Close and open again the bash command line window or type bash.

```
export JDK_HOME=<JAVA_PATH>
```

<sup>10</sup> Oracle java is not in the repository any more. Use this link to install it (<http://www.wikihow.com/Install-Oracle-Java-on-Ubuntu-Linux>)

## 9-2 Ant

Ant does not need installing. In windows, open its archive package in a folder for example “c:\ant”. We need to add its bin directory to the “path” variable. So

1. Right click on “My computer” icon and select the properties item
2. Click on “Advanced” tab (in Windows Vista first click on “Advanced system settings”).
3. Click on “Environment Variables” button
4. On the “System variables” click on the new button, define a new variable, and have its name “ANT\_HOME” and its value “C:\program files\Java\jdk1.6.0\_01”.
5. Edit the “path” variable and add “;%ANT\_HOME%\bin” at the end of it. (pay attention to the semi-colon sign)
6. Click on OK button in all the opened windows as the final step. Note that if you have a command prompt window open, you should close and reopen it to have its variables updated

In Linux, just install the ant package using the package manager.

## 9-3 Java3D

You need java3D if you want to compile Mobisim3D or use 3D resimulate module. Download java3D module from <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138252.html> (I did not redistribute the library due to some license restrictions).

### 9-3-1 Windows

Just run the install file and follow the wizards.

1. Suppose that you installed Java3D in “C:\Program Files\Java\Java3D”, and Java JDK in “C:\Program Files\Java\jdk1.6.0\_01”.
2. Find ext folder in its subfolders “C:\Program Files\Java\Java3D\1.5.1\ext” and copy its contents (jar files) in the “C:\Program Files\Java\jdk1.6.0\_01\jre\lib\ext”. Also copy the contents of “C:\Program Files\Java\Java3D\1.5.1\bin” (dll files) into “C:\Program Files\Java\jdk1.6.0\_01\jre\bin”.
3. If you only installed Java JRE “C:\Program Files\Java\jre1.6.0\_01” not JDK: Find ext folder in its subfolders “C:\Program Files\Java\Java3D\1.5.1\ext” and copy its contents (jar files) in the “C:\Program Files\Java\jre1.6.0\_01\lib\ext”. Also copy the contents of “C:\Program Files\Java\Java3D\1.5.1\bin” (dll files) into “C:\Program Files\Java\jre1.6.0\_01\bin”.

### 9-3-2 Linux

1. Change the downloaded file permissions:

```
chmod 755 java3d-1_5_1_01-linux-i586.bin
```

2. Use locate command to find the java program location

```
locate /rt.jar
```

The command output should be something like:

```
/usr/lib/jvm/java-6-sun-1.6.0.0.7/jre/lib/rt.jar
```

Therefore, your java home path is “/usr/lib/jvm/java-6-sun-1.6.0.0.7/”

3. Change directory to the jre directory in the java home path:

```
cd /usr/lib/jvm/java-6-sun-1.6.0.0.7/jre
```

4. open the downloaded package in the current directory (note that you should have root access to do so):

```
/path-to-download-files/java3d-1_5_1_01-linux-i586.bin
```

You can also do it in another way. Run the last command (open archive) in its downloaded folder and copy the .jar and .so files like what has been done for windows.