City Simulator V 2.0

Contents:

Overview

What is City Simulator?
City Simulator is a scalable, three-dimensional model city that enables creation of dynamic spatial data simulating the motion of up to 1 million people. City Simulator is written in **Java** and is designed to generate realistic data for evaluation of database algorithms for indexing and storing dynamic location data. City Simulator is offered on alphaWorks so that developing systems for indexing spatial data can be tested for scalability and performance against a standardized data set.

How does it work?
Control parameters allow easy creation of realistic events such as daily commutes. A graphical user interface allows observation of the simulation. City Simulator can generate a simulation with over 10^6 individuals moving along streets, buildings, and between building floors in three dimensions. Advanced settings allow exploration of efficiency of indexing algorithms over daily commute cycles. The output of the simulation can be viewed in real time; output is also produced as a comma-separated variable (CSV) text file, which contains a unique person ID, a time stamp, and x,y,z coordinates. This data can be imported into any database to study various spatial indexing and spatial query technologies.

Platform Requirements for City Simulator

Platform
    Windows® 95 or above
Java Tools
    JRE
Computer
    Pentium II or above
    Minimum of 128 MB of RAM

What's New

Version 2.0 of the city simulator includes many new features.

1) City plans can now be created from an xml document. A sample city plan comes with the package. Look at the sample file city_plan.xml for an eample of the city plan document type definition (dtd). To create your own city plan you may write your own filter to, for example, import GML data or define your own buildings and roads. As long as you conform to the city_plan dtd, just name your xml file city_plan.xml and City Simulator will run with your city plan.
2) The CityPlan.java class now has an interface CityPlanInterface.java and abstract clas CityPlanAbstract.java. If you wish you may write your own CityPlan class to generate a plan. See the sample code XML_CityPlan.java for an example of how this is done.
3) The simulator now comes with a properties file (CitySimulator.properties) that can be used to configure program settings. The command line interface still exists as does the GUI (and supercedes values in the properties file if used). The properties file allows control of many more parameters including the name of the CityPlan class the program should use (eg. XML_CityPlan.java or your own class).
4) Parameters affecting motion rules can now be changed in real time if you run using the GUI. Simply click on a place (Road, Building, Intersection, etc.) and you can change up down probabilities for building floors, cause traffic delays on roads, invoke a realistic traffic flow model for roads and intersections, etc. Many parameters are non-static class variables so one can empty one building while filling another.
5) A Boolean parameter conservePopulation has been added to the property file. When set to false, this parameter will cause the population do decrease (ie as people leave the city they are not replaced). Removal of people from the city after the relaxation is complete (determined by relax moves). This feature allows one to investigate the rate at which a region of a city can be evacuated. When set to false, population vs time will be displayed in a new graphical window. Poputation vs time data will also be output to a file "pop_"+filename.txt where filename is the datafile name you specify.

6) Places now have color. A public method is available to color buildings (grayscale) based on their height.
7) GrassyField(s) (extending place) have been added.
8) A runme.bat file is included in the package (for windows&#174)
9) CitySimulator now works on Linux based systems.

Installation Instructions for City Simulator

1. Create a directory on your machine (such as *City Simulator*).
2. Download the zip file and extract it into that directory.
3. From a command prompt, *cd* to your *City Simulator* directory.
4. Type "java -classpath CitySimulator.jar CitySimulator.CitySimulator".

City Simulator requires that you specify a data directory for the output file. By default, this is *C:\Temp*. You can specify any data directory you wish, but it must already exist when you run the simulation.

FAQ

1. What applications can use the data generated by City Simulator?

The output is simply a text file (with comma-separated variables). The text file contains a unique person ID, a time stamp, and *x,y,z* coordinates. This data can be imported into any database to study various spatial indexing and spatial query technologies.

2. How do I run City Simulator?

From a command line, type "java -classpath CitySimulator.jar CitySimulator.CitySimulator" or add the following command line options (see ReadMe.txt for details):

java -classpath CitySimulator.jar CitySimulator.CitySimulator true/false filename.ext drive:\datadirectory\ numpeople maxRelax_Steps finalSteps

3. After I launch City Simulator and press *Start*, why do I get a message that reads "String entered is not a valid directory"?

City Simulator requires that you specify a data directory for the output's text data file. You may specify any directory, but it must already exist when you press *Start*. The default data directory is *C:\Temp*.

Simulation Details

CitySimulator is a Java city simulation designed to generate realistic data to evaluate database algorithms for indexing and storing dynamic location data. We made this simulator available on alpha works so that individuals developing systems to index location data can test scalability and performance against a standardized dataset. The CitySimulator can generate a simulation with over 10^6 individuals moving along streets, buildings, and between building floors in three dimensions. Advanced settings allow one to explore efficiency of indexing algorithms over daily commute cycles. The output of the simulation can be watched in real time and/or output as a comma separated variable (csv) text file.

Execution Modes:

The simplest way to run CitySimulator is to use the runme.bat file provided. You can also invoke the simulator with runtime arguments listed below.

CitySimulator may be invoked either in batch mode using a command line interface (CLI), or using the CitySimulator graphical user interface (GUI). The command line interface allows execution with the ability to modify many of the simulation run parameters. The GUI interface contains advanced dialogues to adjust the full range of simulation parameters. In either case the user may enable or disable runtime graphics to watch the simulation in progress or to realize maximum performance for large simulations.

Getting Started:

java -classpath CitySimulator.jar CitySimulator.CitySimulator

Execution with graphics, loading runtime defaults. Parameters may be changed with the "Input Run Parameters Dialogue".

java -classpath CitySimulator.jar CitySimulator.CitySimulator *true/false*

Execution with runtime defaults and graphics enabled (true) or disabled (false).

java -classpath CitySimulator.jar CitySimulator.CitySimulator *true/false filename.ext*

Execution with runtime defaults and graphics enabled (true) or disabled (false).
Replacing the default file name with *filename.ext*

java -classpath CitySimulator.jar CitySimulator.CitySimulator *true/false filename.ext drive:\datadirectory*

Execution with runtime defaults and graphics enabled (true) or disabled (false).
Replacing the default file name with *filename.ext*
Replacing the default data directory with *drive:\datadirectory*. This directory must exist.

java -classpath CitySimulator.jar CitySimulator.CitySimulator *true/false filename.ext drive:\datadirectory\ numpeople maxRelax_Steps finalSteps*

Same as above with addition of the following parameters:

NumPeople	The number of people simulated (added to the city)
MaxRelax	Maximum number of relaxation steps before start of simulation. The simulator iterates until *maxRelax* cycles or until the startThreshold is reached (see startThreshold below). By default this is 2000.
finalSteps	The number of simulation cycles actually stored in the datafile.

java -classpath CitySimulator.jar CitySimulator.CitySimulator *true/false filename.ext drive:\datadirectory\ numpeople max_Relax_Steps final_steps startThreshold fillThreshold emptyThreshold*

Same as above with addition of the following parameters:

startThreshold The simulation begins with people added to the city on roads at various (ten) entry points or "bridges" into the city. Over time the buildings act as sinks for the people on the roads and the road or ground floor population decreases as the buildings fill. The start threshold represents the fraction of people that should be on the ground level when the simulation should "start" (i.e. when the relaxation is finished). At a *startThreshold* of 15% (0.15, the default value) the histogram of people vs. floor number matches the histogram of number of places (floors) vs. floor number hence the city is considered to be at or near steady state.

fillThreshold This threshold represents the minimum fraction of people that should ever be left on the ground level. When the ground level population falls below the *fillThreshold* fraction, the building enter and exit probabilities and the floor up and down probabilities are changed to trigger a commute out of the buildings. By default the *fillThreshold* is 9% (0.09). The current version of the code does not allow adjustment of the enter/exit or up/down probabilities – these have been tuned to generated reasonable transport of the simulated people. Future version may allow them to be changed.

emptyThreshold This threshold represents the maximum fraction of people that should ever be left on the ground level after a commute out of the buildings has been triggered. When the ground level population exceeds the *emptyThreshold* fraction, the building enter and exit probabilities and the floor up and down probabilities are changed to trigger a commute back into the buildings (i.e. they are restored to the default values at the beginning of the simulation). By default the *emptyThreshold* is 50% (0.50). The current version of the code does not allow adjustment of the enter/exit or up/down probabilities – these have been tuned to generated reasonable transport of the simulated people. Future version may allow them to be changed.

Note: If graphics mode is set to "true", users may still override or revise any and all parameters entered on the command line. Changes are made using the Run Parameter dialogues in the GUI.

Default Run Parameters:

If not otherwise specified, the default run parameters are:

F*ileName:*	test.txt	
DataDirectory:	c:\Temp\	***(the specified directory must exist!)***
NumPeople:	100	(try 100-1000000)
MaxRelax:	2000	(need not be any larger for *startThreshold* >= 0.15)
FinalSteps:	200	
StartThreshold:	0.15	
FillThreshold:	*0.09*	
EmptyThreshold:	0.50	

Fixed (Internal) Probabilities *(not changeable in the current version*):

// in and out probabilities		
static double exitHigh	0.9;	// commute out
static double exitLow	0.1;	// commute in
static double enterProb	0.1;	
static double exitProb	exitLow;	// default at start
// up and down probabilities		
static double downHigh	0.06;	// commute out
static double downLow	0.03;	// commute in
static double upProb	0.03;	
static double downProb	downLow;	// default at start

CitySimulator.properties

The properties file allows control of many more parameters including the name of the CityPlan class the program should use (eg. XML_CityPlan.java or your own class). Any property that can be specified in the command line or GUI may also be set in the properties file. If no command line arguments are used, the values specified in the properties file are used by default and will appear in the GUI dialogues (see below). Simply edit CitySimulator.properties to customize the simulation.

City Plan DTD

City plans can now be created from an xml document. A sample city plan comes with the package. Look at the sample file city_plan.xml for an example. To create your own city plan you may write your own filter to, for example, import GML data or define your own buildings and roads. As long as you conform to the city_plan dtd, just name your xml file city_plan.xml and City Simulator will run with your city plan. You can also specify the name of the city plan xml file by changing the xmlPlanFile property in CitySimulator.properties.

For simplicity, in this version we use a common spatial definition for all objects. In version 2.0, all places are rectangles or rotated rectangles. Future versions may support

other interfaces to place objects and other shapes. For now, colors are assigned in the private City.java class and not customizable.

To create any object, the xml element must specify the object class (eg Road), the orientation of the object (angle in radians from +PI to –PI), the length and width of the object. Two child nodes must specify the EndPoints of the centerline of the object. The distance between these endpoints must correspond to the Length attribute set in the parent element. The Length attribute is used only to validate your city plan (it is tested against the length computed from the two endpoints). In V2.0, all places are rectangular and specified with the following dtd.

```
<Road Angle="0.0" Length="528.0" Width="100.0">
        <EndPoint1 x="471" y="1050"/>
        <EndPoint2 x="999" y="1050"/>
        <MotionRules EnterProb="0.1" ExitProb='0.1" VelGradient="2.4"/>
</Road>
```

In the sample city plan, roads are divided into short segments to take advantage of the realtime ability to create local obstructions and resistance to traffic flow on a road segment.

Note that the x,y values may be either integers or floats. The Length and Width must be specified as floats. You may use any units in x and y (Lat, Long, meters, etc.). The simulator will rescale the xy coordinates to internal units using the CityScale, Xmin, and Ymin attributes defined in the root element.

```
<CityPlan CityScale="scale" Xmin="0.0" Ymin="0.0">
```

You must define these three attributes to match your city size to the internal scale used in the simulation. To accomplish this, Xmin and Ymin should be set to the minimum X and Y values defined in your plan (in whatever units you chose to use). The CityScale parameter must be selected such that for any point x,y in your city

x**scale* <= CITYSIZE

CITYSIZE is defined in the properties file and has a default value of 6300.

In the sample city plan, The *scale* factor is set to 1.0 and the city is defined in internal units. To determine your scale factor, calculate (xmax-xmin) and (ymax-ymin) for your city in your units. The larger of these two is your (*city extent).* Set the CityScale attribute to:

Scale = CITYSIZE/(*city extent*) for your city.

Note that the altitude (of floors) is not affected by the scale factor.

There is a second scale factor in the properties file called MAPSCALE. This determines the screen size or MAPSIZE displayed in pixels. With CITYSIZE=6300 and MAPSCALE = 10 the screen size MAPSIZE=630 pixels. These are the recommended values. If you change MAPSCALE or CITYSIZE in the properties file, you MUST also change MAPSIZE.

```
MAPSCALE        = 10
CITYSIZE        = 6300
MAPSIZE         = 630   # MAPSIZE=CITYSIZE/MAPSCALE
```

The xml city plan must also specify motion rules. We suggest that initially you use the rules provided in the sample file at least initially. The motion parameter attributes are place dependent (see sample). They may be changed in real time during the simulation by clicking on a place on the map.

City Plan Interface

The CityPlan.java class now has an interface CityPlanInterface.java and abstract clas CityPlanAbstract.java. If you wish you may write your own CityPlan class to generate a plan.

Any class to create a city plan must extend CityPlanAbstract. For example:

```
public class XML_CityPlan extends CityPlanAbstract
```

Your class must also define a buildPlan method as specified in the public CityPlanInterface.

```
public void buildPlan(City acity);
```

See the sample code XML_CityPlan.java for an example of how this is done.

Advanced Dialogue: "Advanced Options" (settings):

With default run parameters the period of the commuter cycle is approximately XXX program cycles.

Gif File Name (not used)

Start Threshold .15

Equilibration complete when 15% of people are on ground level and the remainder are on building floors. At this value, the histogram of people vs.

building floors approximate the density of floors vs. floor height). See discussion of buildings and floors.

Fill Threshold .09

When 9% of people are left on ground level, toggle to commute-from-work Parameters.

Empty Threshold .5

When 50% of people are on roads toggle to commute-to-work parameters.

Advanced Dialogue: "Set Interactions"

With default run parameters the period of the commuter cycle is approximately XXX program cycles.

Max Exit Prob 0.9

Probability of Exiting a Building if in a building and near a door in commute-from-work state.

Min Exit Prob 0.1

Probability of Exiting a Building if in a building near a door in commute-to-work state.

MaxDown Prob 0.06

Probability of moving to a lower floor if on a building floor in commute-from-work state.

Min Down Prob 0.03

Probability of moving to a lower floor if on a building floor in commute-to-work state.

Enter Prob 0.1

Probability of entering a building if just outside a building near a door.

Up Prob 0.03

Probability of moving up to a higher building floor in on a building floor (below the top floor).

Output Data Format:

The text file output produced by the simulator appears as follows:

```
# Num People  = 1000 moves between additions = 200 final Moves = 20   output filename: =:\Temp\test3.txt
# random number seed = 17
# data format is:
# index, time,  x,y,z

732311514, 0.027093, 218.2,241.8,0.0    # cycle = 1
732311515, 0.047522, 602.6,208.2,0.0    # cycle = 1
732311516, 0.072045, 639.8,246.2,0.0    # cycle = 1
732311517, 0.106028, 560.0,672.2,0.0    # cycle = 1
732311518, 0.135268, 215.6,213.6,0.0    # cycle = 1
                    .
                    .
```

.

The first four lines contain header information.
Header Lines 1,2 summarize some of the run parameters.
Lines 3 and 4 describe the data format.
Following the header is the run data. In each row following the header are 5 columns of data.

Column 1, an integer, is a primary key unique to each simulated "person" being tracked. To speed the simulation, we do not generate a new random key for each person. Rather we pick on random key and then increment to generate a new unique key for each person. There is no other significance to the number.

Columns 2-5 contain doubles precision numbers. The second column is the time (in seconds) at which the simulated location report was "received".
Columns 3-5 represent position (x,y,z) in METERS for that location report. Initially, the z value for all people is 0.0 as people are introduced to the city at entry points on roads which all have altitude zero (we could add height variations to the roads and ground floors of buildings but have not done so as of yet). As time progresses and people move in to and up building floors, the reported z value will correspond to the altitude of the floor the person is on.
Finally, after the data each row contains a comment indicating cycle number. One complete cycle takes place each time every simulated person's location report is updated. So if the population is 10,000 a single cycle will contain 10,000 updates or reports. This number may vary only slightly as people leave the city. Anytime a person leaves, a new person is introduced to the city so the total population is held constant. We view the expiration of location data (a person leaving the zone) as a fact to be handled by any indexing scheme.

Run Time GUI

At the start of the simulation, data may be input via the “Input Run Parameters” dialogue. After the start button is pressed, three new windows will appear:

1) A console window containing run time messages and information
2) A map of the city
3) A “Floor Population Histogram” window.

The graphics will be static for some time while the simulator executes the “relaxation” phase described above. After relaxation, moving people are visible and represented as points displayed in one of three colors. White points are people on the roads or on the ground floor of a building. Red points are above the ground floor but not at the top floor. Yellow points are people at the top floor of a building. The total number of floors varies building to building.

The dialogue labeled Floor Population Histogram shows the histogram of building floors (constant) as well as the distribution of people (dynamic) as a function of building floor.

Object Model

The city is constructed with collections of various objects. The high level objects in the simulation are:

Place
Person

The following objects extend place:

Building

Road
Intersection
Floor

Places have several attributes. Coordinates, extents, altitude or floor number, and pointers to neighboring places. Places also contain enter and exit probabilities, up down probabilities, drift probabilities (on roads), scatter probabilities (on intersections), etc. People move according to rules based on the place they are in. A person on a building floor does a random walk. At specific points (stairways) they may, move up or down (if not at the top floor or ground floor respectively) , leave a building (if near a door) etc. A person on a road moves with a linear combination of:
Velocity = (random walk component) + (Drift Velocity).
Magnitude of the drift velocity increases as a person moves closer to the center of a road. Direction of the drift velocity changes sign at the center (so on one side of a road people move North or East, on the other side they move South or West). Road objects also have orientations that determine if the drift velocity is North/South or East West.

The implementation of this object model is quite efficient and simulations of up to one million people may be fully relaxed and iterated for several hundred cycles in a run time of just a few hours on a PC.

The package CitySimulator.jar contains the following files

Java Class Files:

Building.class
ChartPanel.class
City.class
CityPlan.class
CityPlanAbstract.class
CityPlanInterface.class
CitySimulator.class
CitySimulator.properties
CityWindowAdapter.class
Console.class
Constants.class
DocumentOutputStream$1.class
DocumentOutputStream.class
ExitPoint.class
Floor.class
GrassyField.class
GridBagHelper.class
Input_data.class
Intersection.class
LocationChartFrame.class
matrixDisplay.class
MyOutput.class

Person.class
Place.class
PointVelocity.class
Road.class
RunSimulation.class
TransPoint.class
VFlowLayout.class
XML_CityPlan.class
XML_CityWriter.class

Gif Icon File:

city2.gif

Author Bios:

James Kaufman
IBM Almaden Research Center

Dr. James H. Kaufman is a Research Staff Member in the Web Technologies Department at the Almaden Research Center, IBM Research Division. Dr. Kaufman received his B.A. in Physics from Cornell University and his PhD in Physics from U.C.S.B.. Dr. Kaufman has worked in several areas of research at IBM including micro-magnetic modeling, magneto-electronic devices, dynamical systems, pattern formation, critical phenomena, and diamond-like carbon for protective thin film disk overcoats. He has also served as a technical manager, and led a department in Science and Technology responsible for development new materials for thin film recording heads and non-volatile magnetic memory (MRAM). He recently moved to computer science where his current research interests included management of dynamic location data and technology for privacy protection. James can be reached through email at kaufman@almaden.ibm.com.

Jussi Myllymaki
IBM Almaden Research Center

Dr. Jussi Myllymaki is a Research Staff Member in the Web Technologies Department at the Almaden Research Center, IBM Research Division. He received his MS degree in Industrial Management from Helsinki University of Technology, Finland, and his MS and PhD degrees in Computer Science from the University of Wisconsin at Madison. His early work focused on performance evaluation of tertiary storage devices and database systems. Later, he worked in Web search engine technology and Web data extraction (see the IBM developerWorks article at

http://www-106.ibm.com/developerworks/library/wa-wbdm/). Dr. Myllymaki's current interests include location-based services and management of dynamic location data. His IBM Research web page is at http://www.research.ibm.com/people/j/jussi/.
Jussi can be reached through email at jussi@almaden.ibm.com.

Jared Jackson
IBM Almaden Research Center

Jared Jackson is a Research Associate at IBM's Almaden Research Center and a recent graduate from Harvey Mudd College with a degree in Computer Science. Jared can be reached through email at jjared@almaden.ibm.com.