Network design of a bicycle-sharing system

Silvana de Leeuw (370701)

July 6, 2015

Abstract

In this report we will be analysing a bicycle-sharing system. If we want to stimulate people to use these systems, we first need to make sure that people have trust in the system. We will be validating two different models that estimate the performance of a bicycle-sharing system with a simulation. We will also use the simulation program Anylogic to gain insight into the system. After the validation we will implement a case study and conclude that the performance of the Equilibrium State Model seems to improve when the system contains more stations and bicycles while the Stochastic Network Flow model is not capable of estimating all cases.

1 Introduction

Bicycle-sharing systems are commonly used in big cities. In our view there are two main reasons why the these systems should be used more. Firstly transportation by bicycle is a green way of transportation as it does not emission CO_2 and if the bicycles are shared, less have to be made within a factory. Secondly it is also healthier for humans to use a bicycle instead of a car or public transport. If we want more people to use the system, we need to make sure that the bicycles are at the right station at the right time such that the service level of the system is high enough to pay for the bicycles, the redistribution, and the docking stations.

If the number of bicycles, the redistribution, and the renting price of a bicycles are chosen optimal, this system could generate profits. If we want to know which redistribution should take place, we first need to know more about the bicycle movements in the system. Besides, if we want to make it profitable, we need to be able to set a requirement utilization for each bicycle.

In a bicycle-sharing system, bicycles are shared and each bicycle can be docked at one of the stations in the system. People who take a bicycle from a docking station need to dock it at another station and pay for their trip. Now someone else can use the same bicycle from the station it was docked. In a bicycle-sharing system the number of trips made needs to be maximized since these trips generate revenues. A trip can only be made if there is a bicycle available at a station when a customer arrives.

We would like to verify some results from Shu, Chou, Liu, Teo and Wang [1]. In this article several models related to bicycle-sharing systems, are constructed and verified with a simulation. These models aim to estimate the performance of bicycle-sharing system. They estimate the number of trips made over the total time period and the bicycle utilization with a optimal initial bicycle distribution in a Stochastic Network Flow Model. They also provide an Equilibrium State Model to determine where the bicycles will be when time goes to infinity. With this model, we can gather information that can be useful for determining the redistribution of the bicycles after some time periods.

In this report we will firstly verifying the Stochastic Network Flow Model and the Equilibrium State Model constructed in Shu et al [1] with a numerical example and a case study. We will use two simulations to compare the models with. Secondly we will compare the performance of the models with the small numerical example with the performance of the model with the data from the case study. Are the Stochastic Network Flow Model and the Equilibrium State Model more accurate when the bicycle-sharing system contains more docking stations and bicycles?

In section 2 the Stochastic Network Flow and Equilibrium State Model are described with the necessary lemma's and proofs. Then in section 3 the numerical example is introduced and we explain how we verify the models. In section 4 we will see the performance of both models with a bicycle-sharing systems that contains more stations and bicycles. In section 5 we will conclude the report and finally in section 6 some suggestions for future research and discuss the findings.

2 Model Description

In this section we will describe two models for bicycle-sharing systems constructed in Shu et al [1]. To analyse these models, we will first look at the connections between the customer arrival rates and the expected number of trips that will take place.

2.1 Relationship between customer arrivals and trips made

In order to construct a model to estimate the performance of a bicycle-sharing system, Shu et al [1] firstly describe connections between the expected number of trips made and the customer arrival rate for a certain origin-destination trip. They used three lemma's to construct their models. To describe these lemma's we will firstly define some sets, parameters and variables.

- **Set** ζ : the set of all stations in the bicycle-sharing system.
- **Set** T : the set of all K time periods (1...K).
- **Parameter** $r_{ij}(t)$: the customer arrival rate at station i of customers with destination station j during time period t.
- **Parameter** $D_{ij}(t)$: the number of customer arrivals traveling from station i to station j during time period t.
- **Parameter** $D_i(t)$: the number of customer arrivals at station i during time period t.
- **Variable** $y_{ij}(t)$: the expected number of bicycles moving from station i to station j during time period t. Note that y_{ii} denotes the number of bicycles that stay at station i during time period t.
- **Variable** $y_i(t)$: the expected number of available bicycles at station i at the beginning of time period t.
- **Parameter** $x_i(t)$: the number of bicycles available at station i at the beginning of time period t.

Using these definitions we can define the number of bicycles that leaves station i during time period t as $min(x_i(t), D_i(t))$. If $x_i(t)$ is smaller than $D_i(t)$ we assume that the first arriving customers will use the available bicycles. If a customer arrives, but there is no bicycle available, then the customer will be declined and leave the system.
Let $Q_i(p)$ be a sequence of independent Bernoulli random variables with mean p while $0 \leq p \leq 1$, then $D_i(t)[p] = \sum_{k=1}^{D_i(t)} Q_k(p)$ This means that $D_i(t)[p]$ could be seen as the number of arriving customers that is tagged when there is a chance of p to be tagged.
With the Poisson Thinning Lemma as described in Ross [2], we can conclude that $D_i(t)[p]$ is Poisson distributed with rate $p \times \sum_{j\in\zeta:j\neq i} r_{ij}(t)$. If we define $p_{ij}(t) \equiv r_{ij}(t)/\sum_{k\in\zeta:k\neq i} r_{ik}(t)$ we can state

$$D_{ij}(t) \sim D_i(t)[p_{ij}(t)]. \tag{1}$$

For a given $x_i(t)$, let

$$min(x_i(t), D_i(t)[p]) = \sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p). \tag{2}$$

According to the equation (2) combined with the Poisson Thinning Lemma, the number of bicycles leaving station i with destination station j follows the distribution of $min(D_i(t), x_i(t))[p_{ij}(t)]$.

Now we can define the number of bicycles available at station i at the beginning of the next time periods. This is equal to the number of bicycles available at station i at the beginning of this time period t minus the number of bicycles that depart from station i during time period t, plus the number of bicycles that arrive station i during time period t.

$$x_i(t+1) = x_i(t) - \sum_{j\in\zeta:j\neq i} min(D_i(t), x_i(t))[p_{ij}(t)] + \sum_{j\in\zeta:j\neq i} min(D_j(t), x_j(t))[p_{ji}(t)]. \tag{3}$$

Since we do not know the number of trips that are going to be made on each origin-destination link beforehand, we have to use expectations in the bicycle-system models. The expected number of bicycle trips made during the modelling time is given by

$$\sum_{t=0}^{K} \sum_{i \in \zeta} \sum_{j \in \zeta : j \neq i} E(min(D_i(t), x_i(t))[p_{ij}(t)]). \tag{4}$$

Let us define the following variables

$$y_i(t) = E(x_i(t)) \tag{5}$$

$$y_{ij}(t) = E(min(D_i(t), x_i(t))[p_{ij}(t)]) \tag{6}$$

$$y_{ii}(t) = y_i(t) - \sum_{j \in \zeta : j \neq i} y_{ij}(t) \tag{7}$$

With these definitions we will describe some structural properties of $y_{ij}(t)$ by defining and proving three lemma's.

The first lemma defines the maximum of the expected number of bicycles moving from station i to station j during time period t.

Lemma 1. $y_{ij}(t) \leq r_{ij}(t)$

Proof. This lemma can be proven with the use of some definitions

$$y_{ij}(t) = E(min(D_i(t), x_i(t))[p_{ij}(t)]) \tag{8}$$

$$r_{ij}(t) = E(D_i(t)[p_{ij}(t)]) \tag{9}$$

We also know that

$$E(min(D_i(t), x_i(t))[p_{ij}(t)]) \leq E(D_i(t)[p_{ij}(t)]) \tag{10}$$

To see why this is true, lets say we have two random variable, U and O. If we know that $U(\omega) \leq O(\omega)$ for all events ω, then we can conclude that $E(U) \leq E(O)$. Now both sides of equation (10) are expectations of random variables and for every demand realization $d_i(t)$ it holds that $min(d_i(t), x_i(t)) \leq d_i(t)$. Therefore we can conclude that equation (10) holds. This leads to the conclusion that $y_{ij}(t) \leq r_{ij}(t)$. □

The next lemma describes a relationship between the expected number of bicycles at a station at the beginning of the next time period by using the expectations of this time period.

Lemma 2.

$$y_i(t+1) = y_i(t) - \sum_{j \in \zeta : j \neq i} y_{ij}(t) + \sum_{j \in \zeta : j \neq i} y_{ji}(t)$$

Proof. This lemma follows from the relation described in equation (3). If we take the expectation of all parts of that equation and combine them with the definitions of $y_i(t)$ and $y_{ij}(t)$, we get this lemma. This means that the expected number of bicycles at station i at the beginning of time period *t+1* is equal to the expected number of bicycles at station i at the beginning of time period t minus the expected departures from station i during time period t plus the expected number of arrivals at station i during time period t. □

The last lemma describes the relationship between the ratio of two expected number of trips from one station with different destinations and passenger arrival rates.

Lemma 3.

$$\frac{y_{ij}(t)}{y_{il}(t)} = \frac{r_{ij}(t)}{r_{il}(t)}$$

Proof. To proof this we will have to use some definitions

$$y_{ij}(t) = E(min(D_i(t), x_i(t))[p_{ij}(t)]) = E(\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{ij}(t))) \tag{11}$$

Similarly,

$$y_{il}(t) = E(min(D_i(t), x_i(t))[p_{il}(t)]) = E(\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{il}(t))) \tag{12}$$

Next we can use the fact that $E(Q_k(p_{ij}(t))) = p_{ij}(t)$, and $E(Q_k(p_{il}(t))) = p_{il}(t)$ as this follows the definition of the expectation of a Bernoulli sequence. We will combine these definitions and condition on $min(x_i(t), D_i(t))$.

$$\begin{aligned}\frac{y_{ij}(t)}{y_{il}(t)} &= E(\frac{\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{ij}(t))}{\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{il}(t))}) = E(E(\frac{\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{ij}(t))}{\sum_{k=1}^{min(x_i(t),D_i(t))} Q_k(p_{il}(t))}|min(x_i(t), D_i(t)))) \\ &= E(E(\frac{min(x_i(t), D_i(t))p_{ij}(t)}{min(x_i(t), D_i(t))p_{il}(t)})) = \frac{p_{ij}(t)}{p_{il}(t)} \end{aligned} \tag{13}$$

Now we will use the definitions of $p_{ij}(t)$ and $p_{il}(t)$.

$$\frac{y_{ij}(t)}{y_{il}(t)} = \frac{\frac{r_{ij}(t)}{\sum_{m\in\zeta:m\neq i} r_{im}(t)}}{\frac{r_{il}(t)}{\sum_{m\in\zeta:m\neq i} r_{im}(t)}} = \frac{r_{ij}(t)}{r_{il}(t)} \tag{14}$$

□

2.2 Stochastic Network Flow model

The first model Shu et al [1] constructs is a Stochastic Network Flow Model. This model maximizes the total amount of trips by determining the best initial allocations of the bicycles with a given bicycle utilization requirement. Some of the sets, parameters and variables are already described in section 2.1. Only the utilization requirement needs to be introduced in this section.

- **Parameter** β : the bicycle utilization requirement over the modelling time. This is equal to the average number of time periods each bicycle needs to be utilized.

The Stochastic Network Flow Model is now defined as follows.

$$Z^*(\beta) = max_{x_i(0),y_{ij}(t)}(\sum_{t=0}^{K}\sum_{i\in\zeta}\sum_{j\in\zeta:j\neq i} y_{ij}(t)) \tag{15}$$

subject to

$$y_i(t+1) = y_i(t) - \sum_{j\in\zeta:j\neg i} y_{ij}(t) + \sum_{j\in\zeta:j\neg i} y_{ji}(t) \qquad \forall i\in\zeta, t\in T \tag{16}$$

$$\sum_{t=0}^{K}\sum_{i\in\zeta}\sum_{j\in\zeta:j\neq i} y_{ij}(t) \geq \beta \sum_{i\in\zeta} x_i(0) \tag{17}$$

$$y_i(t) = y_{ii}(t) + \sum_{j\in\zeta:j\neg i} y_{ij}(t) \qquad \forall i\in\zeta, t\in T \tag{18}$$

$$\frac{y_{ij}(t)}{y_{il(t)}} = \frac{r_{ij}(t)}{r_{il}(t)} \qquad \forall i,j,l\in\zeta \tag{19}$$

$$y_i(0) = x_i(0) \qquad \forall i\in\zeta \tag{20}$$

$$0 \leq y_{ij}(t) \leq r_{ij}(t) \qquad \forall i,j\in\zeta, i\neq j, t\in T \tag{21}$$

In this problem $Z^*(\beta)$ is the optimal objective value for a given β. In equation (15) the total number of trips made in the analysed time periods $1...K$ is maximized.
The first set of constraints in equation (16) states that the amount of bicycles at the beginning of time period $t+1$ is equal to the amount of bicycles at the beginning of time period t minus the departures from station i during time period t plus the arrivals at station i during time period t. These constraints follow from lemma 1.
The second set of constraints in equation (17) makes sure that the bicycle utilization requirement β is met.
The third set of constraints in equation (18) states that the number of bicycles at station i at the beginning of time period t should always equal to the number of bicycles that will stay at station i during time period t plus the number of bicycles that will leave station i during time period t.
The fourth set of constraints in equation (19) follows from lemma 3. The fifth set of constraints in equation (20) state that the number of bicycles at station i at the initial time period should be equal to the number of bicycles assigned to station i during the initial allocation of the bicycles.
Then finally the last set of constraints in equation (21) make sure that the number of trips made between two stations can never be negative and can never become more than the customer arrival rate of trips between the same two stations during all time periods t. These constraints follow from lemma 2.
It is also possible to determine the optimal initial bicycle allocation for a given number of bicycles with the Stochastic Network Flow Model. Lets say that the total number of bicycles in the system is equal to N.

$$\sum_{i \in \zeta} x_i(0) = N \tag{22}$$

If we add equation (22) to the Stochastic Network Flow Model, it will determine the optimal initial bicycle allocation for a given number of bicycles in the system (N).

2.2.1 Upper bound

The Stochastic Network Flow Model described above is very effective in providing an estimation of the performance of a bicycle-sharing system. However, it remains an estimation which means that it is not exact. The optimal solution of the model can suppress certain trips if that generates a higher objective value. For example, take a three node bicycle-sharing system with two bicycles located at station 3. Now suppose $r_{31}(0) = r_{32}(0) = 1$, $r_{23}(t) = r_{32}(t) = 1$ for all $t > 1$, and $r_{ij}(t) = 0$ otherwise. If a bicycles moves from station 3 to station 1 during time period 0, then the bicycle will be stuck there for the remaining time periods. In real time and in the simulation this will happen eventually. However, the number of trips made over the whole day will be maximized if both bicycles move between station 2 and 3. The Stochastic Network Flow Model thus suppresses the bicycles from moving to station 1 in order to maximize the objective.

2.3 Equilibrium State Model

To determine which bicycle reallocation should take place, we have to know the equilibrium state of the bicycle-sharing system. As $t \to \infty$ we expect $y_i(t+1) = y_i(t)$ in the equilibrium state. If we let $y_{ij} = lim_{t\to\infty} y_{ij}(t)$ and define N as the total number of bicycles in the system, then the following model can be used to determine the equilibrium state of the bicycle-sharing system.

$$Z^* = max \sum_{i,j \in \zeta : j \neq i} y_{ij} \tag{23}$$

subject to

$$\sum_{j \in \zeta : j \neq i} y_{ij} = \sum_{j \in \zeta : j \neq i} y_{ji} \qquad \forall i \in \zeta \tag{24}$$

$$\frac{y_{ij}}{y_{il}} = \frac{r_{ij}}{r_{il}} \qquad \forall i, j, l \in \zeta \tag{25}$$

$$0 \leq y_{ij} \leq r_{ij} \qquad \forall i, j \in \zeta \tag{26}$$

$$\sum_{i \in \zeta} (y_{ii} + \sum_{j \in \zeta : j \neq i} y_{ij}) = N \tag{27}$$

The objective in equation (23) maximizes the total number of trips made in the equilibrium state. The first set of constraints in equation (24) states that the number of trips arriving at station i during time period t should be equal to the number of bicycles departing from station i during time period t. The set of constraints in equations (25) and (26) are almost the same as the constraints in equation (19) and (20) respectively, except that these constraints do not depend on the time periods.
The last constraint in equation (27) states that each of the N bicycles should either be used in a trip or must stay at the same station in the equilibrium state.

2.3.1 Adjustment of the solution

This model will determine the movements of the bicycles in the equilibrium state. It does not however show where the bicycles that are not used will be when time goes to infinity. Shu et al [1] suggests moving the bicycles to the sink node. These are the nodes with a larger inflow than outflow of bicycles. However, they do not mention how the bicycles should be divided over several sink nodes. In this report we will assume that these bicycles will 'sink' in the node with the largest inflow of bicycles compared to the outflow.

3 Numerical Experiments

In this section we will verify both models described in section 2 by comparing them to the results of a simulation. To do so we will firstly introduce the numerical example. Then we will describe the simulation and lastly we will compare some results of both models with the results of the simulation.

3.1 Numerical example

This numerical example consists of three nodes and arrival rates that are also used in Shu et al [1]. The customer arrival rates are assumed to be Poisson distributed. All nodes are connected and each node represents a station with docks for the bicycles. It is assumed that the duration of each trip is always shorter than one time period of fifteen minutes. However, if for example a bicycle from station 1 arrives at station 2 before the end of a time period, it is assumed that the bicycle can not be used until the start of the new time period. The arrival rates are time-invariant, which means that the customer arrival rates at each node will be the same for each time period.

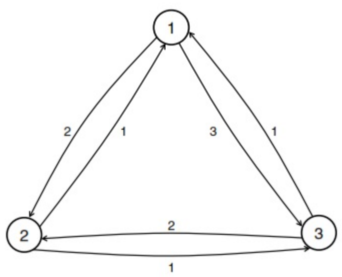

Figure 1: Three Node Example

Figure 1 displays the three node example with the customer arrival rates on the edges. For example, the customer arrival rate of station 1 who want to travel to station 2 is two.

3.2 CPLEX

Both models are implemented in IBM ILOG CPLEX Optimization Studio 12.6.1, or briefly referred to as CPLEX. This is a optimization software package that can be connected to several programming languages. For this report we imported CPLEX into Eclipse and use the Java programming language. Eclipse is a open source framework that allows you to install new plug-ins.

3.3 Simulation

To verify the models described in section 2 we used two forms of simulations. The first program w used is Anylogic. This program gives a nice visualization of the system and improves the intuition. The second is a simulation in Eclipse in Java language. The most important profit of this program and programming language is the fact that it allows to get numerical results in a short time period.

3.3.1 Anylogic

In this program, several simulation tools are optional. For this simulation an agent based simulation. In such simulations the agents can behave certain ways and they can transit between state. Persons can for example be infected, sick or healthy and they can transition between the three states. These states and their possible transition are visualised and connected in a state chart. The state chart of the numerical example is shown in figure 2.

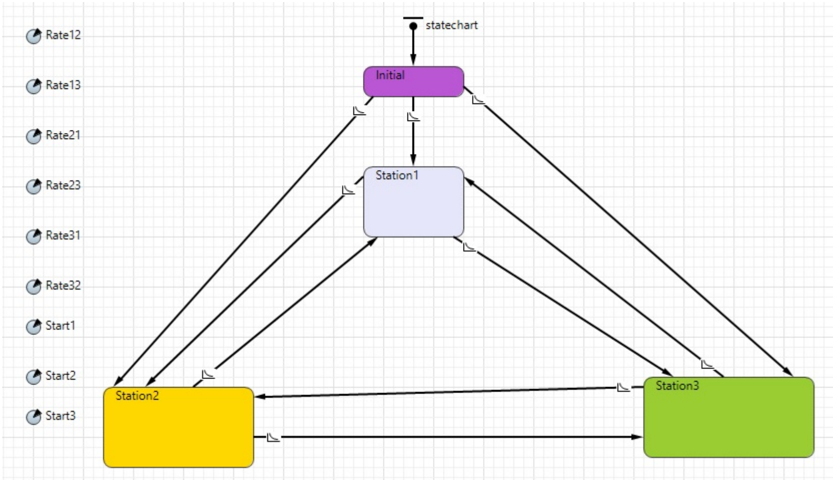

Figure 2: Anylogic state chart

In this simulation, the bicycles are the agents and the stations are their states. A bicycle can transit between the stations. The transitions are triggered by the arrivals of customers. The Poisson arrival process is build into Anylogic and we only have to define the rates. Now we can define variables and statistics that will keep track of what happens during the simulation. For example in this simulation a diagram showing the number of bicycles at each station appears on the screen.
When an agent enters a state, it can perform a entry action. Also when an agent leaves a state, it can perform a exit action. These actions can, for example, update variables or change the colour of the agent depicted on the screen. A screenshot of a simulation run of this example is shown in figure 3.

The bicycles (depicted as persons since the program does not have bicycles shapes) are changing colour during the simulation run, depending on in which state the bicycles are. During a simulation run the statistics and the diagram containing the number of bicycles at each station are automatically updated. If you set a model time as ending time of one simulation run, you can repeat the simulation run and generate results. You can even change parameters for each simulation, which means that you can see what happens when the parameters change. These are just some things that are possible using Anylogic but of course there are many more possibilities within this program.

3.3.2 Java

To describe this simulation, we need to define the following sets, parameters and variables that are used in the simulation.

- **Set** ζ : the set of all stations in the bicycle-sharing system.
- **Set** L : the set of number of simulations (1...H).
- **Set** T : the set of all K time periods (1...K).

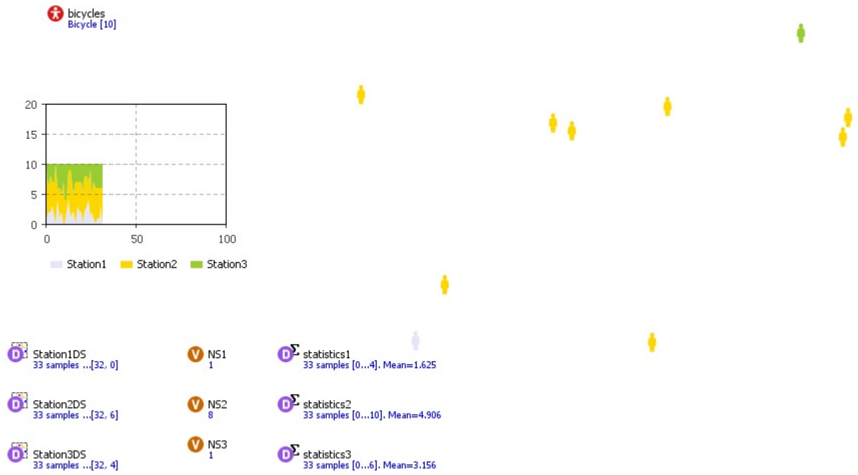

Figure 3: Screenshot of running Anylogic

- **Parameter** $r_{ij}(t)$: the customer arrival rate at station i with destination station j during time period t.
- **Variable** $c_{ij}(t)$: the number of trips made from station i to station j during time period t.
- **Variable** $b_i(t)$: the number of bicycles available at station i at the beginning of time period t.
- **Variable** m_j : the movements made from the current station i to station j during the current time period t.
- **Variable** M_{ij} : movements from station i to station j during current time period t.
- **Variable** $o(h)$: the total number of trips made during simulation run h
- **Variable** $u(h)$: the bicycle utilization during simulation run h.
- **Variable** $L_{ij}(h)$: the last bicycle movements of simulation run h.

With these definitions, we can now describe the main steps of the simulation. Some steps are later explained in more detail.

1. Initialization: $H = 2000$, $K = 50$, o, u, L
2. Read data, fill $r_{ij}(t)$ and $b_i(0)$
3. **For** all $h \in L$
4. Initialization: c, b
5. **For** all $t \in T$
6. Initialization M
7. **For** all $i \in \zeta$
8. **if** $b_i(t) \geq 0$
9. Initialization m
10. $minimum = min(b_i(t), \sum_j r_{ij}(t))$
11. **For** $1 \dots minimum$
12. Determine destination this bicycle
13. Save the movement in m

14. **Next**
15. **End if**
16. Save movements of bicycles from station i in M
17. **Next**
18. Process all movements in $c_{ij}(t)$ and $b_i(t)$
19. **Next**
20. Calculate $o(h)$, $u(h)$ and $L_{ij}(h)$
21. **Next**

During step 2 in the overview above, the customer arrival rates and the initial bicycle distribution are imported into the program. The data is stored in text-files which are imported and read during this step. In this step we can also tell the program which customer arrival rates should be saved for which time periods. This allows us to simulate with time-varying customer arrival rates. Of course it is also possible to work with time-invariant customer arrival rates. In this case $r_{ij}(t)$ will be the same for all t.
In step 4 the variables c and b are initialized. After each simulation run, these variables will be initialized in order to make them useful for the next simulation run.
The next initialization, the initialization of M, occurs in step 6. For each time period t of this simulation run h we have to create M because this variable will contain all the movements of this time period t. When we move to the next time period $t+1$, this variable we be initialized again such that we can use them for the next time period $t+1$.

Now that we are in simulation run h during time period t, we have to determine which movements will take place. This will be done by iterating through the stations and determining how many bicycles will leave each station to each destination. So now we will determine which customers will take a bicycle station i during time period t in simulation run h.
In step 8 we will firstly check whether there are bicycles available at station i before we start assigning customers to bicycles. If there are bicycles available, we will initialize m for this station i in which the number of bicycles that leave this station with destination j during this time period t is saved for all j. Next we decide in step 10 how many bicycles can leave this station. As explained in section 2 we have to take the minimum of the number of bicycles available and the sum of all customers arrival rates of station i.

Now that we know how many bicycles will leave station i, we have to decide what their destinations will be. This is done with a transformation of the Poisson arrival rates. For each bicycle that will be moved we will generate the transformation numbers f_j for each possible destination station j. We start by initializing a new random number generator and then draw a number v_j from the uniform distribution between 0.0 and 1.0 for each $j \in \zeta$ and $j \neq i$. With this v_j and $r_{ij}(t)$ we can now get to f_j with the following formula:

$$f_j = \frac{-log(1.0 - v_j)}{r_{ij}(t)} \tag{28}$$

Next we decide which station got the lowest f_j as this will be the destination of this bicycle. Once we decided the destination of this bicycle, we will save this movement in m in step 12 and continue to the next bicycle.
We will do this for all bicycles that will leave this station and then save the movements from this station i in M in step 16. If we have decided what all movements will be from all stations $i \in \zeta$ during this time period, we will process these movements by updating $c_{ij}(t)$ and $b_i(t)$ in step 18.
When we arrive at step 20 we will determine how many trips are made during this simulation run and save that value for this simulation h in $o(h)$. We will also determine the bicycle utilization of this simulation h and save it in $u(h)$. Then we will save the last movements M_{ij} of this simulation run h in $L_{ij}(h)$.
These steps are repeated until all H simulation runs are completed. Now we can calculate the needed numbers with the help of $o(h)$ and $u(h)$. For the validation of the Stochastic Network Flow Model we will use the average of $o(h)$ and $u(h)$ to compare to the model results. For the Equilibrium State Model we will use the averages of the last movements of each simulation run by taking the average of $L_{ij}(h)$.

3.4 Comparison

In this section we will compare the results of the Stochastic Network Flow Model and the Equilibrium State Model with the results of the simulation.

3.4.1 Stochastic Network Flow Model

To verify the Stochastic Network Flow Model we first implemented the numerical example in CPLEX with constant arrival rates during 50 time periods. This model allows you to set a utilization requirement(β) as described in section 2.2. We solved the model for different utilization requirements. This model then determines the initial bicycle distribution such that the number of trips made will be maximized (note that this also means that the model determines the number of bicycles in the system). Next we used this bicycle distribution as input for the simulation. This simulation the simulates the trips made during 50 time periods and calculates the total number of trips made and the bicycle utilization. This is repeated 2000 times and then we took the average of both the number of trips made and the bicycle utilization.

Table 1: Comparison of the Stochastic Network Flow Model and the simulation

Deterministic			*Simulation*	
% time periods a bicycle is used	**Objective (number of trips)**	**Number of bicycles in the system**	**Average % time periods a bicycle is used**	**Average objective (number of trips)**
0%	500.00	157.00	6.22%	487.97
10%	428.75	85.75	9.79%	419.83
20%	340.83	34.08	19.58%	333.58
30%	314.62	20.97	29.10%	305.08
40%	302.96	15.15	39.06%	295.85
50%	296.38	11.86	48.48%	287.48
60%	292.14	9.74	58.15%	283.18
70%	289.19	8.26	67.81%	280.05
80%	287.02	7.18	76.00%	272.84
90%	285.00	6.33	81.17%	256.90
100%	283.33	5.67	80.68%	228.74

The results of this comparison are shown in table 1. Firstly we look at the result of the deterministic model. An increase in the bicycle utilization requirement leads to a decrease in the objective. We also see a decrease in the optimal number of bicycles in the system. As a bicycle now needs to be used during more time periods, according to the bicycles utilization requirement, there can be less bicycles in the system. If less bicycles are available this will also decrease the total number of trips made as more customers are denied.

If we compare the bicycle utilization requirements in the first column of table 1 with the bicycle utilization in the simulation in the fourth column of table 1 we see that the model always overvalues the utilization. This can be explained by the fact that the model is a estimation and gives a upper bound of the number of trips made as described in section 2.2.1. We see that the difference between both values increases when the requirement increases. When we look at the objectives of the deterministic model and the simulation, we see the same pattern.

If we look at the last two rows of table 1 we see that the bicycle utilization in the simulation actually decreases. This suggests that the Stochastic Network Flow Model is not always capable of estimations with a bicycle requirement of 100%. The model suppresses more trips to satisfy the bicycle utilization requirement which leads to a bigger gap between the model and the simulation. However, the simulation will not suppress trips and therefore it could happen that the bicycle utilization actually decreases slightly.

3.4.2 Equilibrium State Model

For the comparison of this model with the simulation, we assumed that there are 10 bicycles in the system. The results of this model are stated in table 2

Table 2: Equilibrium State Model with numerical example

Origin \Destination	**1**	**2**	**3**
1	0.00	0.67	1.00
2	1.00	4.33	1.00
3	0.67	1.33	0.00

Number of trips made in the equilibrium state

As we see in table 2 only 5.66 bicycles move in the equilibrium state while 4.33 are staying at station 2. When comparing the expected number of bicycles at station 2 to the customer arrival rates in figure 1, it can be seen that station 2 is a sink node. This means that bicycles will sink into this station when time increases. This station will have more bicycles than customers arriving. Each arriving customer can use a bicycle and the other bicycles will stay at the station.

The results of this model are compared with a simulation run of 50 time periods. The simulation ran 2000 times and after each simulation the last movements were saved. After that we took the average of these movements. The simulation needs an initial bicycle distribution as input in order to simulate the trips that the bicycles will make. This distribution can be chosen randomly but in this case we decided to use the same initial bicycle distribution as Shu et al [1]. This means that there are five bicycles at station 1, two bicycles at station 2 and, three bicycles at station 3 at the beginning of each simulation run. The result of these simulation are stated in table 3

Table 3: Simulation with the numerical example

Origin \Destination	**1**	**2**	**3**
1	0.00	0.67	1.00
2	1.00	4.19	1.00
3	0.67	1.33	0.14

Average number of trips made during the last time period

Most values in table 3 are the same as in table 2. The main difference however, is the fact that some of the not moving bicycles stay at station 3 in the equilibrium state. When we look at the customers arrival rates in figure 1 you can see that station 3 also has a overflow of arriving bicycles compared to arriving customers. However, for station 2 this overflow is larger. So in some cases, there might be bicycles staying in station 3 during the last of the 50 time periods. We see, nonetheless, that this only happens on rare occasions, thus that in most cases the bicycles in station three will have moved to station 2 already.
In table 4 the number of bicycles at each station in the equilibrium state are stated for the model and the simulation.

Table 4: Equilibrium State Model compared to simulation

	Deterministic	**Simulation**	**Abs. Diff.**	**% Diff. deterministic**
Avg no. of bicycles at station 1	1.67	1.67	0.00	0.00%
Avg no. of bicycles at station 2	6.33	6.19	0.14	2.21%
Avg no. of bicycles at station 3	2.00	2.14	0.14	7.00%

To compare the deterministic model and the simulation we take their absolute difference as a percentage of the number of bicycles in the equilibrium state according to the deterministic model. From table 4 we can conclude that the pattern of the number of bicycles at each station in the equilibrium state is the same for the deterministic model and the simulation. However, on average there is still a difference of 3.07% per station.

4 Case studies

In this case study we will implement both models described in section 2 with data from the metro stations in Rio De Janeiro. Firstly we will describe how we use such data to estimate the customer arrival rates. Then we will describe the data we received and and how we used it. Lastly we will look at the results of both models and the simulation.

4.1 Estimation arrival rates

For many cities a lot of data is available about people travelling. They know how many people use the bus, metro or train. However, in many cases there is no information available about how many people travel on some origin-destination route. Often we only know how many people get into a system at each station and we know how many people leave the system at each station. Often this data is also collected per time period.
For a good working system, we need to know how many people travel on a certain origin-destination route. If this is not known, then the origin-destination demand can be estimated using only the number of people getting into the system at each station and the number of people leaving the system at each station during each time period. These estimates can be used to determine the customer arrival rates of a certain origin-destination combination.
In the article by Marujo [3] a algorithm is developed to estimate these customer arrival rate. These estimations will then be implemented into the models and simulations as the $r_{ij}(t)$ parameter.

4.2 Rio De Janeiro

The data set we used in this case study, is estimated with the algorithm described in Marujo [3]. This data set contains the origin-destination estimated customer arrival rates for 35 metro stations in Rio De Janeiro. The names and codes we used for these station can be found in the Appendix. The data set contains estimations for each link on three different moments of a day, and an estimation of an average weekday. The three moments are the morning peak, lunch time and, the evening peak. Each customer arrival rates is measured over a 3 hour period. However, with the use of the Poisson Thinning lemma (see Ross [2]), we can use these rates for smaller time periods. In this case we decided to use time periods of half hours as all the trips should take no longer than one time period. The bicycle-sharing system in Rio De Janeiro possesses 600 bicycles in total.

4.3 Stochastic Network Flow Model

With this data from the metro stations in Rio De Janeiro, we would like to verify the Stochastic Network Flow Model with time varying customer arrival rates. We choose to look at one day and used the four different customer arrival rate sets described in section 4.2. The day starts with a morning peak from 06:30 till 09:30, the lunch break from 11:00 till 14:00 and, a evening peak from 17:00 till 20:00. The time in-between is filled with the regular weekdays data. All customer arrival rates were scaled to half hour periods. This means that there are 33 time periods in the model and a simulation run. A simulation run is repeated 2000 times and then the results were collected. The comparison values are shown in table 5.

Just like in the numerical experiment we see the same pattern when the bicycle utilization requirement increases. Also both the bicycle utilization and the number of trips made is higher in the model results than the simulation results. However, in this case we see the same values for a bicycle utilization requirement of 0%, 10% and, 20%. This means that the maximum number of trips possible with these customer arrival rates can already be achieved when there are 314235.18 bicycles in the system. Also the bicycle utilization will be at least 20% of the time periods. If we calculate the overall performance as the percentile difference between the bicycle utilization requirement, we will see that the model seemed to perform better for the numerical example. This model is not capable of capturing this case.
In the optimal objective without a bicycle utilization requirement, the bicycles will be used for some percentage of the time periods. If this percentage is higher than your bicycle utilization requirement, then the estimation will differ more than with requirements larger than this percentage.

Table 5: Comparison of the Stochastic Network Flow Model and the simulation with data from Rio De Janeiro

Deterministic			*Simulation*	
% time periods a bicycle is used	**Objective (number of trips)**	**Number of bicycles in the system**	**Average % time periods a bicycle is used**	**Average objective (number of trips)**
0%	2244744.93	314235.18	20.13%	208736.47
10%	2244744.93	314235.18	20.13%	208736.47
20%	2244744.93	314235.18	20.13%	2087837.91
30%	2077944.73	209893.41	27.94%	1934994.39
40%	1850759.35	140209.04	37.19%	1720781.37
50%	1598014.43	96849.36	46.52%	1486667.44
60%	982836.63	49638.21	56.79%	926791.44
70%	567563.19	24569.84	66.34%	537874.38
80%	381509.79	14451.13	76.18%	363269.53
90%	274260.24	9234.35	86.17%	262577.05
100%	96525.05	2925.00	96.53%	93177.45

4.4 Equilibrium State Model

We also used the data sets described in section 4.2 in the Equilibrium State Model and again compared the results with the results of a simulation. We did this for all four data sets and the whole comparison tables can be found in the appendix. A simulation run was again 50 time periods and was repeated 2000 times. In table 6 the average differences between the model and the simulation of the number of bicycles at each station in the equilibrium state are shown.

Table 6: Average difference between the Equilibrium State Model and the simulation

Data Set	**average % difference**
Weekday	0.56%
Morning	0.88%
Lunch	0.58%
Evening	0.64%

We see that these numbers are rather smaller than the differences in the numerical experiment. These data sets contain more stations and also more bicycles. So more stations and bicycles in the system will lead to a higher accuracy of the Equilibrium State Model.

5 Conclusion

The Equilibrium State Model seems to get close to bicycles distribution of the moving bicycles when t goes to infinity. However, it does not take into account the distribution of the bicycles that do not move in the equilibrium state.
From the comparison of the Equilibrium State Model with the simulation we can conclude that the model tempts to be closer to the simulated values when more bicycles are available and when there are more stations in the system.
With the Stochastic Network Flow Model the number of bicycles needed to obey a utilization requirement can be determined. However, it does not seem to be capable of estimating all cases. In the optimal objective without a bicycle utilization requirement, the bicycles will be used for some percentage of the time periods. If this percentage is higher than your bicycle utilization requirement, then the estimation will differ more than with requirements larger than this percentage.

6 Discussion

First of all we would like to mention that the output using the data sets from Rio De Janeiro should be downscaled. The data set represents the arrival rate of customer who want to travel with the metro. It is not realistic to assume that all metro passengers will choose to use the bicycles if they are distributed correctly.

In future research we would firstly like to extend the Anylogic model such that it can handle more nodes in a system. It would also be a good step to make a moving picture of a bigger simulation to see where the bicycles are moving. This could give some insight into improvements of the system.
Secondly we saw that Shu et all [1] did not mention how the bicycles that are not moving should be distributed over the stations in the equilibrium state. Since most of these bicycles will 'sink' into one or more stations, we think it would be interesting to see which stations they 'sink' into with what numbers as this can give us insight into which bicycle redistribution should take place if we want to redistribute the bicycles after one day for example. And lastly we would like to analyse the bicycle redistribution model constructed by Shu et al [1] because we think redistribution of the bicycles is the main reason for looking into a Equilibrium State Model. Redistribution can also make the whole system more profitable and trustworthy which are both important for the reputation of bicycle-sharing systems.

References

[1] Shu, J., Chou, M., Liu, Q., Teo, C.-P., Wang, I.-L., (2013). Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research. Vol. 61, No.6, pp. 1346-1359.*

[2] Ross, S.M., (2014). Introduction to probability models 11th edition.

[3] Marujo, L.G., (2015). OD matrix estimation and the mode choice function.

7 Appendix

7.1 Stations in Rio De Janeiro

This table contains all the stations we used in this report with the codes used in the tables.

Table 7: Station names with codes

Name Station			**Code Station**
Ipanema/General	Osório		IGO
Cantagalo			CTG
Siqueira	Campos		SCP
Cardeal	Arcoverde		CAV
Botafogo			BTF
Flamengo			FLA
Largo	do	Machado	LMC
Catete			CTT
Glória			GLR
Cinelândia			CNL
Carioca			CRC
Uruguaiana			URG
Presidente	Vargas		PVG
Central			CTR
Praça	Onze		POZ
Estácio			ESA
Afonso	Pena		AFP
São	Francisco	Xavier	SFX
Saens	Peña		SPN
Cidade	Nova		CNV
São	Cristóvão		SCR
Maracanã			MRC
Triagem			TRG
Maria	da	Graça	MGR
Nova	América/Del	Castilho	DCT
Inhaúma			INH
Engenho	da	Rainha	ERN
Thomaz	Coelho		TCL
Vicente	de	Carvalho	VCV
Irajá			IRJ
Colégio			CLG
Coelho	Neto		CNT
Acari/Fazenda	Botafogo		AFB
Engenheiro	Rubens	Paiva	ERP
Pavuna			PVN

7.2 Equilibrium State Model - Simulation

The next table is a comparison of the Equilibrium State Model with the simulation with the weekday data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 8: Equilibrium State Model and Simulation with weekday data in Rio De Janeiro

	Deterministic	**Simulation**	**Absolute Difference**	**%Difference from deterministic**
SPN	41.36	41.47	0.11	0.27
SFX	7.10	7.06	0.04	0.56
AFP	7.00	6.93	0.07	1.00
ESA	7.81	7.86	0.05	0.64
POZ	4.39	4.4	0.01	0.23
CTR	79.54	79.72	0.18	0.23
PVG	8.12	8.2	0.08	0.99
URG	38.92	39.14	0.22	0.57
CRC	55.24	54.92	0.32	0.58
CNL	31.74	31.68	0.06	0.19
GLR	9.02	9	0.02	0.22
CTT	10.57	10.59	0.02	0.19
LMC	22.23	22.3	0.07	0.31
FLA	15.01	14.87	0.14	0.93
BTF	58.15	58.18	0.03	0.05
CAV	10.11	10.14	0.03	0.30
SCP	17.90	17.8	0.1	0.56
CTG	10.77	10.73	0.04	0.37
IGO	25.27	25.12	0.15	0.59
CNV	4.06	4.09	0.03	0.74
SCR	8.02	8.05	0.03	0.37
MRC	4.84	4.81	0.03	0.62
TRG	4.06	4.05	0.01	0.25
MGR	6.62	6.63	0.01	0.15
DCT	14.08	14.18	0.1	0.71
INH	5.03	5.06	0.03	0.60
ERN	4.17	4.13	0.04	0.96
TCL	2.26	2.23	0.03	1.33
VCV	12.78	12.84	0.06	0.47
IRJ	10.16	10.23	0.07	0.69
CLG	4.98	4.98	1.78E-15	0.00
CNT	8.95	9	0.05	0.56
AFB	2.41	2.43	0.02	0.83
ERP	2.35	2.4	0.05	2.13
PVN	45.05	45.2	0.15	0.33

The next table is a comparison of the Equilibrium State Model with the simulation with the morning data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 9: Equilibrium State Model and Simulation with morning data in Rio De Janeiro

	Deterministic	**Simulation**	**Absolute Difference**	**%Difference from deterministic**
SPN	33.48	33.41	0.07	0.21
SFX	7.00	7.11	0.11	1.57
AFP	7.30	7.2	0.1	1.37
ESA	11.02	11.08	0.06	0.54
POZ	10.53	10.59	0.06	0.57
CTR	46.88	47.03	0.15	0.32
PVG	13.54	13.64	0.1	0.74
URG	53.44	53.32	0.12	0.22
CRC	82.67	82.7	0.03	0.04
CNL	44.65	44.52	0.13	0.29
GLR	15.93	15.86	0.07	0.44
CTT	12.66	12.73	0.07	0.55
LMC	28.59	28.5	0.09	0.31
FLA	18.59	18.51	0.08	0.43
BTF	66.61	66.66	0.05	0.08
CAV	9.76	9.73	0.03	0.31
SCP	20.08	20.21	0.13	0.65
CTG	11.76	11.92	0.16	1.36
IGO	34.98	34.92	0.06	0.17
CNV	4.24	4.25	0.01	0.24
SCR	8.92	9.05	0.13	1.46
MRC	4.23	4.2	0.03	0.71
TRG	3.32	3.32	8.88178E-16	0.00
MGR	3.50	3.51	0.01	0.29
DCT	8.79	8.9	0.11	1.25
INH	1.61	1.63	0.02	1.24
ERN	1.52	1.5	0.02	1.32
TCL	0.67	0.69	0.02	2.99
VCV	4.84	4.84	8.88178E-16	0.00
IRJ	4.79	4.87	0.08	1.67
CLG	2.04	2.01	0.03	1.47
CNT	3.08	3.1	0.02	0.65
AFB	0.89	0.91	0.02	2.25
ERP	0.88	0.84	0.04	4.55
PVN	16.96	17.05	0.09	0.53

The next table is a comparison of the Equilibrium State Model with the simulation with the lunch data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 10: Equilibrium State Model and Simulation with lunch data in Rio De Janeiro

	Deterministic	**Simulation**	**Absolute Difference**	**%Difference from deterministic**
SPN	42.95	42.95	7.10543E-15	0.00
SFX	7.97	7.87	0.1	1.25
AFP	7.24	7.27	0.03	0.41
ESA	12.80	12.83	0.03	0.23
POZ	6.10	6.13	0.03	0.49
CTR	59.25	59.32	0.07	0.12
PVG	12.70	12.79	0.09	0.71
URG	52.03	52.34	0.31	0.60
CRC	66.39	66.12	0.27	0.41
CNL	38.87	39.07	0.2	0.51
GLR	10.10	10.17	0.07	0.69
CTT	11.41	11.28	0.13	1.14
LMC	24.96	25	0.04	0.16
FLA	13.84	13.82	0.02	0.14
BTF	55.12	55.06	0.06	0.11
CAV	12.04	12.1	0.06	0.50
SCP	19.77	19.64	0.13	0.66
CTG	11.17	11.2	0.03	0.27
IGO	26.19	26.06	0.13	0.50
CNV	5.96	5.95	0.01	0.17
SCR	8.52	8.48	0.04	0.47
MRC	5.48	5.49	0.01	0.18
TRG	2.85	2.82	0.03	1.05
MGR	4.81	4.81	1.77636E-15	0.00
DCT	12.71	12.82	0.11	0.87
INH	3.16	3.19	0.03	0.95
ERN	2.99	2.93	0.06	2.01
TCL	1.74	1.71	0.03	1.72
VCV	9.31	9.38	0.07	0.75
IRJ	7.51	7.47	0.04	0.53
CLG	3.61	3.58	0.03	0.83
CNT	5.42	5.41	0.01	0.18
AFB	1.62	1.63	0.01	0.62
ERP	0.98	0.99	0.01	1.02
PVN	32.50	32.49	0.01	0.03

The next table is a comparison of the Equilibrium State Model with the simulation with the evening data of Rio De Janeiro. For each station the average amount of bicycles at that station is shown.

Table 11: Equilibrium State Model and Simulation with evening data in Rio De Janeiro

	Deterministic	**Simulation**	**Absolute Difference**	**%Difference from deterministic**
SPN	39.86	39.81	0.05	0.13
SFX	5.86	5.89	0.03	0.51
AFP	7.02	7.12	0.10	1.42
ESA	5.83	5.86	0.03	0.51
POZ	3.51	3.59	0.08	2.28
CTR	85.38	85.16	0.22	0.26
PVG	4.54	4.61	0.07	1.54
URG	19.84	19.76	0.08	0.40
CRC	30.62	30.66	0.04	0.13
CNL	19.46	19.28	0.18	0.92
GLR	8.21	8.25	0.04	0.49
CTT	9.11	9.08	0.03	0.33
LMC	15.15	15.14	0.01	0.07
FLA	12.41	12.5	0.09	0.73
BTF	47.28	47.44	0.16	0.34
CAV	7.28	7.4	0.12	1.65
SCP	13.50	13.54	0.04	0.30
CTG	8.31	8.33	0.02	0.24
IGO	17.18	17.29	0.11	0.64
CNV	3.83	3.75	0.08	2.09
SCR	13.91	13.87	0.04	0.29
MRC	8.33	8.36	0.03	0.36
TRG	5.23	5.24	0.01	0.19
MGR	11.02	11.06	0.04	0.36
DCT	26.15	26.19	0.04	0.15
INH	8.44	8.33	0.11	1.30
ERN	7.20	7.23	0.03	0.42
TCL	4.96	4.95	0.01	0.20
VCV	20.92	20.81	0.11	0.53
IRJ	16.66	16.54	0.12	0.72
CLG	8.60	8.67	0.07	0.81
CNT	16.85	16.82	0.03	0.18
AFB	5.13	5.15	0.02	0.39
ERP	5.20	5.27	0.07	1.35
PVN	77.25	77.17	0.08	0.10

7.3 Appendix: Java Codes

7.3.1 Simulation with output for the Stochastic Network Flow Model

Listing 1: Simulation Stochastic Network Flow Model with input data Rio De Janeiro

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;


```

```
public class MainSimulationRio {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        int nd=1; //Denote number of days
        int np=33; //Denote number of periods on a day
        int NT=nd*np;
        double NTD = (double) NT;
        int H=2000; // Number of simulation runs
        //int NB = 600; // Number of bicycles
        int m=35; //number of nodes in the system
        Double[][] rm = DataSet("ODMetroRioMorningt.txt", m);
        Double[][] rl = DataSet("ODMetroRioLuncht.txt", m);
        Double[][] re = DataSet("ODMetroRioEveningt.txt", m);
        Double[][] rw = DataSet("ODMetroRioWeekdayt.txt", m);
        Double[] Util = new Double[H];
        Double[] Obj = new Double[H];
        Double[][] x = new Double[m][NT]; //Create matrix that will hold all number of
            bicycles at each station
        Double[][][] yj = new Double[m][m][H];
        Double[][][] r = new Double[m][m][NT];
        //Distribute all bicycles at the initialization

        // Fill r
        for (int i=0;i<m;i++){
            for(int j=0;j<m;j++){
                for(int d=0;d<nd ; d++){
                    for (int t= 33*d; t<(33*d)+6;t++){
                        r[i][j][t]=(1.0/6.0)*rm[i][j]; // Fills the morning part on each day
                    }
                    for (int t= (33*d)+6; t<(33*d)+9;t++){
                        r[i][j][t]=(3.0/28.0)*rw[i][j]; // Fills part between morning and
                            lunch on each day
                    }
                    for (int t= (33*d)+9; t<(33*d)+15;t++){
                        r[i][j][t]=(1.0/6.0)*rl[i][j]; // Fills lunch part on each day
                    }
                    for (int t= (33*d)+15; t<(33*d)+21;t++){
                        r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part between lunch and
                            evening on each day
                    }
                    for (int t= (33*d)+21; t<(33*d)+27;t++){
                        r[i][j][t]=(1.0/6.0)*re[i][j]; // Fills evening part on each day
                    }
                    for (int t= (33*d)+27; t<(33*d)+33;t++){
                        r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part after evening on each
                            day
                    }
                }
            }
        }
        Double[] xSNF = new Double[m];
        for(int run=0;run<10;run++){

            if(run==0){
                xSNF = readX("disxR33.0.txt",m);
            }
            else if(run==1){
                xSNF = readX("disxR29.7.txt",m);
            }
            else if(run==2){
                xSNF = readX("disxR26.4.txt",m);
            }else if(run==3){
                xSNF = readX("disxR23.1.txt",m);
            }else if(run==4){
                xSNF = readX("disxR19.8.txt",m);
            }else if(run==5){
                xSNF = readX("disxR16.5.txt",m);
            }else if (run==6){
                xSNF = readX("disxR13.2.txt",m);
            }else if(run==7){
                xSNF = readX("disxR9.9.txt",m);
            }else if (run==8){
```

```java
            xSNF = readX("disxR6.6.txt",m);
        }else{
            xSNF = readX("disxR3.3.txt",m);
        }

        double value=  getSum(xSNF);
        int NB = (int) value;
        for (int i=0;i<m;i++){
            x[i][0] = xSNF[i] ;
        }

        // Execute the simulation H times
        for (int h=0;h<H;h++){
            Double[][][] yh = new Double[m][m][NT];
            yh=Simulation(NT,m,NB,r,x);
            // Save the objective
            Double[][] tempO1 = new Double[m][m];
            Double[] tempO2 = new Double[m];
            for(int i =0; i<m; i++){
                for (int j =0; j<m ; j++){
                    yj[i][j][h] = yh[i][j][NT-1];  //Save all last movements
                    if(i!=j){
                        tempO1[i][j] = getSum(yh[i][j]); // summation over t
                    }else if (i==j){
                        tempO1[i][j]=0.0;
                    }
                }
                tempO2[i] = getSum(tempO1[i]); // summation over t and over j
            }
            Obj[h] = getSum(tempO2); // summation over t , i and j

            // Save the utilization percentage
            Util[h] = (Obj[h] / NB)/NTD;
        }

        //Get the numbers, average of each 2000
        Double[][] ya = new Double[m][m];
        for (int i=0; i<m;i++){
            for (int j =0;j<m;j++){
                Double[] temp = new Double[H];
                for(int h=0;h<H;h++){
                    temp[h] = yj[i][j][h];
                }
                ya[i][j] = round(getAverage(temp),2);
            }
        }

        writeMatrix("SimRioWeekBeta.txt",  ya, m);

        System.out.println("Utilization: "+ getAverage(Util));
        System.out.println("Objective: "+ getAverage(Obj));
    }
}

public static Double[][][] Simulation (int NT, int m,int NB, Double[][][] r,Double[][] x) throws FileNotFoundException, IOException{
    //Initialization
    Double[][][] y = new Double[m][m][NT]; //Create the matrix that will hold all movements
    //Fill x and y with 0.0
    for(int i =0;i<m;i++){
        for (int t=0;t<NT;t++){
            if(t!=0){
                x[i][t]=0.0;
            }
            for (int j=0;j<m;j++){
                y[i][j][t]=0.0;
            }
        }
    }


```

```java

    // Enter the for-loop of the actual simulation
    for(int t=0;t<NT-1;t++){
      Double[][] allMove = new Double[m][m];
      for (int i=0;i<m;i++){
        for (int j=0; j<m ; j++){
          allMove[i][j]=0.0;
        }
      }
      for (int i=0;i<m;i++){
        if(x[i][t]!=0.0){
          x[i][t+1]=x[i][t]; // First get the previous value before adjusting it
          Double[] mi= new Double[m];
          mi=move(i,m,t,r,y,x); //What is the movement
          allMove[i]=mi; //Save the movements
          for (int a=0;a<m;a++){
            if (a==i){
              y[i][a][t+1]= x[i][t] -getSum(mi); // Determine the number of bicycles that stayed at the station
            }else{
              //We can already save the departures. The arrivals need to be saved after all movements are determined
              x[i][t+1] = x[i][t+1] - mi[a];
              y[i][a][t+1] = mi[a];
            }
          }
        }
      }
      for(int i=0;i<m;i++){
        for(int j=0;j<m;j++){
          x[i][t+1] = x[i][t+1] + allMove[j][i];
        }
      }
    }
    return y;
  }

  public static double getSum(Double[] x){
    double som=0.0;
    for (int i=0;i<x.length;i++){
      som = som + x[i];
    }
    return som;
  }

  public static double getAverage(Double[] x){
    double average=0.0;
    double sum=0.0;
    for(int i=0;i<x.length;i++){
      sum=sum+x[i];
    }
    average = sum/x.length;
    return average;
  }

  public static Double[][] DataSet(String filename, int size) throws FileNotFoundException, IOException {
    String line = "";


    Double[][] data = new Double[size][size];

    FileReader fr= new FileReader(filename);
    BufferedReader br = new BufferedReader(fr);
    line=br.readLine();

    String[] fline=line.split(";");
    int c=fline.length;


    for(int j=0; j<size; j++){
      String[] theline=line.split(";");
```

```java
        Double[] td=new Double[size];
        for (int i=0; i<c; i++){
            td[i] = (double) ( Integer.parseInt(theline[i]));
        }
        data[j]=td;
        if(j!=size){
            line=br.readLine();
        }
    }


    br.close();


    return data;
}


public static Double[] readX(String filename, int size)throws FileNotFoundException
    , IOException{//Creating r
    Double[] data = new Double[size]; //Create new array for initial distribution
    String line = "";

    FileReader fr= new FileReader(filename);
    BufferedReader br = new BufferedReader(fr);
    line=br.readLine();

    String[] theline=line.split(";");
    for (int i=0; i<size; i++){
        data[i]=  Double.parseDouble(theline[i]);
    }
    br.close();
    return data;
}


public static Double[] move(int s, int m, int t, Double[][][] r,Double[][][] y,
     Double[][] x){
    Double[] movement=new Double[m];
    // Fill movement with 0.0
    for(int i=0;i<m;i++){
        movement[i]=0.0;
    }

    //Determine the total departure rate
    int sumR=0;

    for (int j =0;j<m;j++){
        if (j!=s){
            double thisrate= r[s][j][t];
            sumR=sumR+ (int) thisrate;
        }
    }
    //Decide how many bicycles can be moved
    int minimum=0;
    if (x[s][t]<= sumR){
        double value=x[s][t];
        minimum= (int) value;
    }else{
        minimum=sumR;
    }

    for (int k=0;k<minimum;k++){
        Double[] pois = new Double[m];
        for (int j =0;j<m;j++){
            if(j!=s){
                Random rand = new Random();
                pois[j] = -Math.log(1.0 - rand.nextDouble()) / r[s][j][t];
            }else{
                pois[j] = 10000000.0;
            }
        }
```

```
            int minPlace=0;
            double minValue=10000000.0;
            for (int h=0;h<m;h++){
                if (pois[h]<minValue){
                    minPlace=h;
                    minValue=pois[h];
                }else if (pois[h] ==minValue){
                    Integer[] ran=new Integer[2];
                    ran[0]=h;
                    ran[1]=minPlace;
                    minPlace=getRandom(ran);
                }
            }
            movement[minPlace] = movement[minPlace] +1.0;
        }

        return movement;
    }

    public static int getRandom(Integer[] array) {
        int rnd = new Random().nextInt(array.length);
        return array[rnd];
    }

    public static  void writeMatrix(String filename, Double [][] yp, int m){

        String fileName = filename;

        try {
            // New Filewriter
            FileWriter fileWriter = new FileWriter(fileName);
            BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);

            for (int i=0;i<m;i++){
                for (int j=0; j<m; j++){
                    bufferedWriter.write(String.valueOf(yp[i][j]));
                    bufferedWriter.write("  ");
                }
                bufferedWriter.newLine();
            }

            // Close files
            bufferedWriter.close();
        }
        catch(IOException ex) {
            System.out.println(
                    "Error writing to file '"
                            + fileName + "'");
        }
    }

    public static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();

        long factor = (long) Math.pow(10, places);
        value = value * factor;
        long tmp = Math.round(value);
        return (double) tmp / factor;
    }

}
```

7.4 Simulation without for the Equilibrium State Model

Listing 2: Simulation Equilibrium State Model with input data Rio De Janeiro

```


import java.io.BufferedReader;
import java.io.BufferedWriter;
```

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;


public class MainSimulationRioES {

    public static void main(String[] args) throws FileNotFoundException, IOException {
        int nd=1; //Denote number of days
        int np=50; //Denote number of periods on a day
        int NT=nd*np;
        double NTD = (double) NT;
        int H=2000; // Number of simulation runs
        int NB = 600; // Number of bicycles
        int m=35; //number of nodes in the system
        Double[][] rm = DataSet("ODMetroRioEveningt.txt", m);
        Double[] Util = new Double[H];
        Double[] Obj = new Double[H];
        Double[][] x = new Double[m][NT]; //Create matrix that will hold all number of bicycles at each station
        Double[][][] yj = new Double[m][m][H];
        Double[][][] r = new Double[m][m][NT];
        //Distribute all bicycles at the initialization

        // Fill r
        for (int i=0;i<m;i++){
            x[i][0] = 0.0;
            for(int j=0;j<m;j++){
                for (int t=0; t<NT; t++){
                    r[i][j][t] = rm[i][j];
                }
            }
        }

        //Randomly distribute bicycles over the stations
        for(int nb=0;nb<NB;nb++){
            Random rand = new Random();
            int place=rand.nextInt(m);
            x[place][0] = x[place][0] + 1.0;
        }


        // Execute the simulation H times
        for (int h=0;h<H;h++){
            Double[][][] yh = new Double[m][m][NT];
            yh=Simulation(NT,m,NB,r,x);
            // Save the objective
            Double[][] tempO1 = new Double[m][m];
            Double[] tempO2 = new Double[m];
            for(int i =0; i<m; i++){
                for (int j =0; j<m ; j++){
                    yj[i][j][h] = yh[i][j][NT-1];  //Save all last movements
                    if(i!=j){
                        tempO1[i][j] = getSum(yh[i][j]); // summation over t
                    }else if (i==j){
                        tempO1[i][j]=0.0;
                    }
                }
                tempO2[i] = getSum(tempO1[i]); // summation over t and over j
            }
            Obj[h] = getSum(tempO2); // summation over t , i and j

            // Save the utilization percentage
            Util[h] = (Obj[h] / NB)/NTD;
        }

        //Get the numbers, average of each 2000
        Double[][] ya = new Double[m][m];
        for (int i=0; i<m;i++){
            for (int j =0;j<m;j++){
```

```
            Double[] temp = new Double[H];
            for(int h=0;h<H;h++){
                temp[h] = yj[i][j][h];
            }
            ya[i][j] = round(getAverage(temp),2);
        }
    }

    writeMatrix("SimRioWeekBeta.txt", ya, m);

    System.out.println("Utilization: "+ getAverage(Util));
    System.out.println("Objective: "+ getAverage(Obj));
}


public static Double[][][] Simulation (int NT, int m,int NB, Double[][][] r,Double[][] x) throws FileNotFoundException, IOException{
    //Initialization
    Double[][][] y = new Double[m][m][NT]; //Create the matrix that will hold all movements
    //Fill x and y with 0.0
    for(int i =0;i<m;i++){
        for (int t=0;t<NT;t++){
            if(t!=0){
                x[i][t]=0.0;
            }
            for (int j=0;j<m;j++){
                y[i][j][t]=0.0;
            }
        }
    }



    // Enter the for-loop of the actual simulation
    for(int t=0;t<NT-1;t++){
        Double[][] allMove = new Double[m][m];
        for (int i=0;i<m;i++){
            for (int j=0; j<m ; j++){
                allMove[i][j]=0.0;
            }
        }
        for (int i=0;i<m;i++){
            if(x[i][t]!=0.0){
                x[i][t+1]=x[i][t]; // First get the previous value before adjusting it
                Double[] mi= new Double[m];
                mi=move(i,m,t,r,y,x); //What is the movement
                allMove[i]=mi; //Save the movements
                for (int a=0;a<m;a++){
                    if (a==i){
                        y[i][a][t+1]= x[i][t] -getSum(mi); // Determine the number of bicycles that stayed at the station
                    }else{
                        //We can already save the departures. The arrivals need to be saved after all movements are determined
                        x[i][t+1] = x[i][t+1] - mi[a];
                        y[i][a][t+1] = mi[a];
                    }
                }
            }
        }
        for(int i=0;i<m;i++){
            for(int j=0;j<m;j++){
                x[i][t+1] = x[i][t+1] + allMove[j][i];
            }
        }
    }
    return y;
}

public static double getSum(Double[] x){
    double som=0.0;
```

```
        for (int i=0;i<x.length;i++){
            som = som + x[i];
        }
        return som;
    }

    public static double getAverage(Double[] x){
        double average=0.0;
        double sum=0.0;
        for(int i=0;i<x.length;i++){
            sum=sum+x[i];
        }
        average = sum/x.length;
        return average;
    }

    public static Double[][] DataSet(String filename, int size) throws
         FileNotFoundException, IOException {
        String line = "";


        Double[][] data = new Double[size][size];

        FileReader fr= new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        line=br.readLine();

        String[] fline=line.split(";");
        int c=fline.length;


        for(int j =0; j<size; j++){
            String[] theline=line.split(";");
            Double[] td=new Double[size];
            for (int i=0; i<c; i++){
                td[i] = (double) ( Integer.parseInt(theline[i]));
            }
            data[j]=td;
            if(j!=size){
                line=br.readLine();
            }
        }


        br.close();


        return data;
    }


    public static Double[] readX(String filename, int size)throws FileNotFoundException
         , IOException{//Creating r
        Double[] data = new Double[size]; //Create new array for initial distribution
        String line = "";

        FileReader fr= new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        line=br.readLine();

        String[] theline=line.split(";");
        for (int i=0; i<size; i++){
            data[i]= Double.parseDouble(theline[i]);
        }
        br.close();
        return data;
    }


    public static Double[] move(int s, int m, int t, Double[][][] r,Double[][][] y,
         Double[][] x){
        Double[] movement=new Double[m];
```

```java
        // Fill movement with 0.0
        for(int i=0;i<m;i++){
            movement[i]=0.0;
        }

        //Determine the total departure rate
        int sumR=0;

        for (int j =0;j<m;j++){
            if (j!=s){
                double thisrate= r[s][j][t];
                sumR=sumR+ (int) thisrate;
            }
        }
        //Decide how many bicycles can be moved
        int minimum=0;
        if (x[s][t]<= sumR){
            double value=x[s][t];
            minimum= (int) value;
        }else{
            minimum=sumR;
        }

        for (int k=0;k<minimum;k++){
            Double[] pois = new Double[m];
            for (int j =0;j<m;j++){
                if(j!=s){
                    Random rand = new Random();
                    pois[j] = -Math.log(1.0 - rand.nextDouble()) / r[s][j][t];
                }else{
                    pois[j] = 10000000.0;
                }
            }
            int minPlace=0;
            double minValue=10000000.0;
            for (int h=0;h<m;h++){
                if (pois[h]<minValue){
                    minPlace=h;
                    minValue=pois[h];
                }else if (pois[h] ==minValue){
                    Integer[] ran=new Integer[2];
                    ran[0]=h;
                    ran[1]=minPlace;
                    minPlace=getRandom(ran);
                }
            }
            movement[minPlace] = movement[minPlace] +1.0;
        }

        return movement;
    }

    public static int getRandom(Integer[] array) {
        int rnd = new Random().nextInt(array.length);
        return array[rnd];
    }

    public static  void writeMatrix(String filename, Double [][] yp, int m){

        String fileName = filename;

        try {
            // New Filewriter
            FileWriter fileWriter = new FileWriter(fileName);
            BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);

            for (int i=0;i<m;i++){
                for (int j=0; j<m; j++){
                    bufferedWriter.write(String.valueOf(yp[i][j]));
                    bufferedWriter.write("  ");
                }
                bufferedWriter.newLine();
```

```
            }

            // Close files
            bufferedWriter.close();
        }
        catch(IOException ex) {
            System.out.println(
                    "Error writing to file '"
                            + fileName + "'");
        }
    }

    public static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();

        long factor = (long) Math.pow(10, places);
        value = value * factor;
        long tmp = Math.round(value);
        return (double) tmp / factor;
    }

}
```

7.5 CPLEX code for the Stochastic Network Flow Model

Listing 3: CPLEX code Stochastic Network Flow Model with input data Rio De Janeiro

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import ilog.concert.*;
import ilog.cplex.*;


public class ModelSNFRio {
    public static void solveMe() throws FileNotFoundException, IOException{
        //Initialization
        double B=0.0;
        int nd=1; //Denote number of days
        int np=33; //Denote number of periods on a day
        int NT=nd*np;
        double NTD=(double) NT;
        int m=35; // Number of nodes in the system
        Double[][][] r = new Double[m][m][NT]; //Create new 'matrix' for the arrival
            rates
        //Because there are only four matrices, it is faster to read all matrices into
            the program and then getting values from the arrays[][]
        Double[][] rm = DataSet("ODMetroRioMorningt.txt", m);
        Double[][] rl = DataSet("ODMetroRioLuncht.txt", m);
        Double[][] re = DataSet("ODMetroRioEveningt.txt", m);
        Double[][] rw = DataSet("ODMetroRioWeekdayt.txt", m);

        for (int i=0;i<m;i++){
            for(int j=0;j<m;j++){
                for(int d=0;d<nd ; d++){
                    for (int t= 33*d; t<(33*d)+6;t++){
                        r[i][j][t]=(1.0/6.0)*rm[i][j]; // Fills the morning part on each day
                    }
                    for (int t= (33*d)+6; t<(33*d)+9;t++){
                        r[i][j][t]=(3.0/28.0)*rw[i][j]; // Fills part between morning and
                            lunch on each day
                    }
                    for (int t= (33*d)+9; t<(33*d)+15;t++){
                        r[i][j][t]=(1.0/6.0)*rl[i][j]; // Fills lunch part on each day
                    }
                    for (int t= (33*d)+15; t<(33*d)+21;t++){
```

```
                    r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part between lunch and evening on each day
                }
                for (int t= (33*d)+21; t<(33*d)+27;t++){
                    r[i][j][t]=(1.0/6.0)*re[i][j]; // Fills evening part on each day
                }
                for (int t= (33*d)+27; t<(33*d)+33;t++){
                    r[i][j][t]=(6.0/28.0)*rw[i][j]; // Fills part after evening on each day
                }
            }
        }
    }


    try{

        //Define new model
        IloCplex cplex = new IloCplex();

        // Decision variables
        IloNumVar[][][] y = new IloNumVar[m][m][NT];
        IloNumVar[][] yi = new IloNumVar[m][NT];
        for(int i=0;i<m;i++){
            yi[i]=cplex.numVarArray(NT, 0.0, Double.MAX_VALUE);
            for (int j=0;j<m;j++){
                y[i][j]=cplex.numVarArray(NT, 0.0, Double.MAX_VALUE);
            }
        }
        IloNumVar[] x = new IloNumVar[m];
        x=cplex.numVarArray(m, 0.0, Double.MAX_VALUE);




        //Expression for the summations and the objective
        IloLinearNumExpr[][] typeDepartures = new IloLinearNumExpr[m][NT]; // Initialization for all the departures from station i, separately for all destinations
        IloLinearNumExpr[][] typeArrivals = new IloLinearNumExpr[m][NT];// Initialization for all the arrivals at station i, separately from all origins
        IloLinearNumExpr[][] C1r= new IloLinearNumExpr[m][NT];// Initialization for the expression on the right side of constraint 1
        IloLinearNumExpr[][] C3r= new IloLinearNumExpr[m][NT];// Initialization for the expression on the right side of constraint 3
        IloLinearNumExpr objective=cplex.linearNumExpr();// Initialization of the objective expression
        for (int i=0; i<m; i++){
            IloLinearNumExpr[] sumArrivals = new IloLinearNumExpr[NT];
            IloLinearNumExpr[] sumDepartures = new IloLinearNumExpr[NT];
            IloLinearNumExpr[] sumC1r = new IloLinearNumExpr[NT];
            IloLinearNumExpr[] sumC3r = new IloLinearNumExpr[NT];
            for(int t=0;t<NT;t++){
                sumDepartures[t]=cplex.linearNumExpr();
                sumArrivals[t]=cplex.linearNumExpr();
                sumC1r[t]=cplex.linearNumExpr();
                sumC3r[t]=cplex.linearNumExpr();
                for (int j=0; j<m; j++){
                    if(i!=j){
                        sumDepartures[t].addTerm(1.0, y[i][j][t]);
                        sumArrivals[t].addTerm(1.0, y[j][i][t]);
                        sumC1r[t].addTerm(-1.0,y[i][j][t]);
                        sumC1r[t].addTerm(1.0,y[j][i][t]);
                        objective.addTerm(1.0, y[i][j][t]);
                    }

                    sumC3r[t].addTerm(1.0,y[i][j][t]);
                }
                sumC1r[t].addTerm(1.0,yi[i][t]);
            }
            typeArrivals[i]=sumArrivals;
```

```
            typeDepartures[i]=sumDepartures;
            C1r[i]=sumC1r;
            C3r[i]=sumC3r;
        }



        // Define objective
        cplex.addMaximize(objective);

        //Constraint 1
        for (int i=0;i<m;i++){
            for (int t=0;t<NT-1;t++){
                cplex.addEq(yi[i][t+1], C1r[i][t]);
            }
        }

        //Constraint 2
        cplex.addGe(objective, cplex.prod(B, cplex.sum(x)));



        //Constraint 3
        for(int i=0;i<m;i++){
            for (int t=0;t<NT;t++){
                cplex.addEq(yi[i][t],C3r[i][t]);
            }
        }

        //Constraint 4
        for(int i=0; i<m; i++){
            for (int j=0;j<m;j++){
                for (int l=0;l<m;l++){
                    for(int t=0;t<NT;t++){
                        if(i!=j && i!=l && j!=l)
                            cplex.addEq(cplex.prod(y[i][j][t], r[i][l][t]), cplex.prod(y[i
                                ][l][t],r[i][j][t]));
                    }
                }
            }
        }

        //Constraint 5
        for(int i=0;i<m;i++){
            cplex.addEq(yi[i][0],x[i]);
        }

        //Constraint 6
        for(int i=0; i<m; i++){
            for (int j=0;j<m;j++){
                for(int t=0;t<NT;t++){
                    if(j!=i){
                        cplex.addLe(y[i][j][t], r[i][j][t]);
                    }
                    cplex.addGe(y[i][j][t],0.0);
                }

            }
        }

        //Don't print what it wants to print
        cplex.setParam(IloCplex.Param.Simplex.Display, 0);

        // write model to file
        cplex.exportModel("lpSNF.lp");



        //Solve the problem and what to print if it can be solved
        if(cplex.solve()){
            Double[][] ya = new Double[m][m];
            Double[] xp = new Double[m];
```

```
            for (int i=0; i<m; i++){
               xp[i]=cplex.getValue(x[i]);
               for (int j =0;j<m; j++){
                  Double[] temp = new Double[NT];
                  for(int k=0;k<NT; k++){
                     temp[k] = cplex.getValue(y[i][j][k]);
                  }
                  ya[i][j] = round(getAverage(temp),2);
               }
            }

            writeMatrix("SNFRio.txt",  ya, m,xp);
            System.out.println("objective = " + cplex.getObjValue());


         }
         else{
            System.out.println("Problem not solved");
         }
         //End the problem
         cplex.end();

      }
      catch (IloException e) {
         System.err.println("Concert exception '" + e + "' caught");
      }
   }

   public static double getAverage(Double[] x){
      double average=0.0;
      double sum=0.0;
      for(int i=0;i<x.length;i++){
         sum=sum+x[i];
      }
      average = sum/x.length;
      return average;
   }
   public static Double[][] DataSet(String filename, int size) throws
        FileNotFoundException, IOException {
      String line = "";


      Double[][] data = new Double[size][size];

      FileReader fr= new FileReader(filename);
      BufferedReader br = new BufferedReader(fr);
      line=br.readLine();

      String[] fline=line.split(";");
      int c=fline.length;


      for(int j =0; j<size; j++){
         String[] theline=line.split(";");
         Double[] td=new Double[size];
         for (int i=0; i<c; i++){
            td[i] = (double) ( Integer.parseInt(theline[i]));
         }
         data[j]=td;
         if(j!=size){
            line=br.readLine();
         }
      }


      br.close();


      return data;


   }
```

```
    public static  void writeMatrix(String filename, Double [][] yp, int m,Double[] xp)
        {

        String fileName = filename;

        try {
            // New Filewriter
            FileWriter fileWriter = new FileWriter(fileName);
            BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);

            for (int i=0;i<m;i++){
                for (int j=0; j<m; j++){
                    bufferedWriter.write(String.valueOf(yp[i][j]));
                    bufferedWriter.write("  ");
                }
                bufferedWriter.newLine();
            }
            for(int i=0; i<m; i++){
                bufferedWriter.write(String.valueOf(xp[i]));
                bufferedWriter.write("  ");
            }

            // Close files
            bufferedWriter.close();
        }
        catch(IOException ex) {
            System.out.println(
                    "Error writing to file '"
                            + fileName + "'");
        }
    }

    public static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();

        long factor = (long) Math.pow(10, places);
        value = value * factor;
        long tmp = Math.round(value);
        return (double) tmp / factor;
    }
}
```

7.6 CPLEX code for the Equilibrium State Model

Listing 4: CPLEX code Equilibrium State Model with input data Rio De Janeiro

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;

import ilog.concert.*;
import ilog.cplex.*;

public class ModelRio {
    public static void solveMe() throws FileNotFoundException, IOException{
        //Initialization
        int N =600; // Number of bicycles in the system
        int m=35; // Number of nodes in the system
        Double[][] r = new Double[m][m]; //Create new 'matrix' for the arrival rates
        r=DataSet("ODMetroRioEveningt.txt",35);
        // Enter numerical example


        try{

            //Define new model
```

```
        IloCplex cplex = new IloCplex();

        // Decision variables
        IloNumVar[][] y = new IloNumVar[m][m];
        for(int i=0;i<m;i++){
            y[i]=cplex.numVarArray(m, 0.0, Double.MAX_VALUE);
        }

        //Expression for the summations and the objective
        IloLinearNumExpr[] typeDepartures = new IloLinearNumExpr[m]; //Initialization for all the departures from station i, separately for all destinations
        IloLinearNumExpr[] typeArrivals = new IloLinearNumExpr[m];//Initialization for all the arrivals at station i, separately from all origins
        IloLinearNumExpr[] allBicycles = new IloLinearNumExpr[m];//Initialization of the numbers needed for constraint 4. For each i y_ii plus typeDepartures
        IloLinearNumExpr objective=cplex.linearNumExpr();//Initialization of the objective expression
        for (int i=0; i<m; i++){
            typeDepartures[i]=cplex.linearNumExpr();
            typeArrivals[i]=cplex.linearNumExpr();
            allBicycles[i]=cplex.linearNumExpr();
            for (int j=0; j<m; j++){
                if(i!=j){
                    typeDepartures[i].addTerm(1.0, y[i][j]);
                    typeArrivals[i].addTerm(1.0, y[j][i]);
                    objective.addTerm(1.0, y[i][j]);
                }
                allBicycles[i].addTerm(1.0, y[i][j]);
            }
        }


        // Define objective
        cplex.addMaximize(objective);


        //Constraint 1
        for(int i=0;i<m;i++){
            cplex.addEq(typeDepartures[i], typeArrivals[i]);
        }

        //Constraint 2
        for(int i=0; i<m; i++){
            for (int j=0;j<m;j++){
                for (int l=0;l<m;l++){
                    if(i!=j && i!=l ){
                        cplex.addEq(cplex.prod(y[i][j], r[i][l]), cplex.prod(y[i][l],r[i][j]));
                    }
                }
            }
        }

        //Constraint 3
        for(int i=0; i<m; i++){
            for (int j=0;j<m;j++){
                if(i!=j){
                    cplex.addLe(y[i][j], r[i][j]);
                }
                cplex.addGe(y[i][j],0.0);
            }
        }

        //Constraint 4
        cplex.addEq(cplex.sum(allBicycles),N);

        //Don't print what it wants to print
        cplex.setParam(IloCplex.Param.Simplex.Display, 0);

        // write model to file
        cplex.exportModel("lpES.lp");

```

```java
            //Solve the problem and what to print if it can be solved
            if(cplex.solve()){
                Double[][] yp = new Double[m][m];
                for(int i=0;i<m;i++){
                    for (int j =0; j<m; j++){
                        yp[i][j]=round(cplex.getValue(y[i][j]),2);
                    }
                }
                writeMatrix("ESRio.txt", yp, m);
                System.out.println("Objective =" +cplex.getObjValue());
                System.out.println("Bicycles station 1 = "+ cplex.getValue(cplex.sum(y[0])));
                System.out.println("Bicycles station 2 = "+ cplex.getValue(cplex.sum(y[1])));
                System.out.println("Bicycles station 3 = "+ cplex.getValue(cplex.sum(y[2])));

            }
            else{
                System.out.println("Problem not solved");
            }

            //End the problem
            cplex.end();

        }
        catch (IloException e) {
            System.err.println("Concert exception '" + e + " ' caught");
        }
    }

    //Read in the data



    public static Double[][] DataSet(String filename, int size) throws FileNotFoundException, IOException {
        String line = "";


        Double[][] data = new Double[size][size];

        FileReader fr= new FileReader(filename);
        BufferedReader br = new BufferedReader(fr);
        line=br.readLine();

        String[] fline=line.split(";");
        int c=fline.length;


        for(int j =0; j<size; j++){
            String[] theline=line.split(";");
            Double[] td=new Double[size];
            for (int i=0; i<c; i++){
                td[i] = (double) ( Integer.parseInt(theline[i]));
            }
            data[j]=td;
            if(j!=size){
                line=br.readLine();
            }
        }


        br.close();


        return data;


    }

    public static void writeMatrix(String filename, Double [][] yp, int m){
```

```
        String fileName = filename;

        try {
           // New Filewriter
           FileWriter fileWriter = new FileWriter(fileName);
           BufferedWriter bufferedWriter =new BufferedWriter(fileWriter);

           for (int i=0;i<m;i++){
              for (int j=0; j<m; j++){
                 bufferedWriter.write(String.valueOf(yp[i][j]));
                 bufferedWriter.write("  ");
              }
              bufferedWriter.newLine();
           }

           // Close files
           bufferedWriter.close();
        }
        catch(IOException ex) {
           System.out.println(
                 "Error writing to file '"
                       + fileName + "'");
        }
    }

    public static double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();

        long factor = (long) Math.pow(10, places);
        value = value * factor;
        long tmp = Math.round(value);
        return (double) tmp / factor;
    }
}
```

7.7 Simulation with Anylogic

Listing 5: The Java source code for the Anylogic simulation

```

public class Anylogic {
    package bbs_constaint_rates;

    import java.io.Serializable;
    import java.sql.Connection;
    import java.sql.SQLException;
    import java.util.ArrayDeque;
    import java.util.ArrayList;
    import java.util.Arrays;
    import java.util.Calendar;
    import java.util.Collection;
    import java.util.Collections;
    import java.util.Comparator;
    import java.util.Currency;
    import java.util.Date;
    import java.util.Enumeration;
    import java.util.HashMap;
    import java.util.HashSet;
    import java.util.Hashtable;
    import java.util.Iterator;
    import java.util.LinkedHashMap;
    import java.util.LinkedHashSet;
    import java.util.LinkedList;
    import java.util.List;
    import java.util.ListIterator;
    import java.util.Locale;
    import java.util.Map;
    import java.util.PriorityQueue;
    import java.util.Random;
```

```
import java.util.Set;
import java.util.SortedMap;
import java.util.SortedSet;
import java.util.Stack;
import java.util.Timer;
import java.util.TreeMap;
import java.util.TreeSet;
import java.util.Vector;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.geom.AffineTransform;
import com.anylogic.engine.connectivity.ResultSet;
import com.anylogic.engine.connectivity.Statement;
import com.anylogic.engine.elements.IElementDescriptor;
import com.anylogic.engine.markup.Network;
import com.anylogic.engine.Position;
import com.anylogic.engine.markup.PedFlowStatistics;
import com.anylogic.engine.markup.DensityMap;

import static java.lang.Math.*;
import static com.anylogic.engine.UtilitiesArray.*;
import static com.anylogic.engine.UtilitiesCollection.*;
import static com.anylogic.engine.presentation.UtilitiesColor.*;
import static com.anylogic.engine.presentation.UtilitiesDrawing.*;
import static com.anylogic.engine.HyperArray.*;

import com.anylogic.engine.*;
import com.anylogic.engine.analysis.*;
import com.anylogic.engine.connectivity.*;
import com.anylogic.engine.gis.*;
import com.anylogic.engine.markup.*;
import com.anylogic.engine.presentation.*;

import com.anylogic.libraries.processmodeling.*;

import java.awt.geom.Arc2D;

import java.io.*;

public class Main extends Agent
{
  // Parameters
  // Plain Variables

  public
double
 NS1;
  public
double
 NS2;
  public
double
 NS3;
  @AnyLogicInternalCodegenAPI
  private static Map<String, IElementDescriptor> elementDesciptors_xjal = null;

  @AnyLogicInternalCodegenAPI
  @Override
  public Map<String, IElementDescriptor> getElementDesciptors() {
    if (elementDesciptors_xjal == null) {
      elementDesciptors_xjal = createElementDescriptors(super.getElementDesciptors
          (), Main.class);
    }
    return elementDesciptors_xjal;
  }
  @AnyLogicCustomProposalPriority(type = AnyLogicCustomProposalPriority.Type.
      STATIC_ELEMENT)
  public static final Scale scale = new Scale( 10.0 );

  @Override
  public Scale getScale() {
```

```java
      return scale;
    }
    // Events

    @AnyLogicInternalCodegenAPI
    public EventTimeout _Station1DS_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _Station2DS_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _Station3DS_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _statistics1_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _statistics2_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _statistics3_autoUpdateEvent_xjal = new EventTimeout(this);
    @AnyLogicInternalCodegenAPI
    public EventTimeout _chart_autoUpdateEvent_xjal = new EventTimeout(this);

    @Override
    @AnyLogicInternalCodegenAPI
    public String getNameOf( EventTimeout _e ) {
       if( _e == _Station1DS_autoUpdateEvent_xjal ) return "Station1DS auto update event";
       if( _e == _Station2DS_autoUpdateEvent_xjal ) return "Station2DS auto update event";
       if( _e == _Station3DS_autoUpdateEvent_xjal ) return "Station3DS auto update event";
       if( _e == _statistics1_autoUpdateEvent_xjal ) return "statistics1 auto update event";
       if( _e == _statistics2_autoUpdateEvent_xjal ) return "statistics2 auto update event";
       if( _e == _statistics3_autoUpdateEvent_xjal ) return "statistics3 auto update event";
       if( _e == _chart_autoUpdateEvent_xjal ) return "chart auto update event";
      return super.getNameOf( _e );
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public EventTimeout.Mode getModeOf( EventTimeout _e ) {
      if ( _e == _Station1DS_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _Station2DS_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _Station3DS_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _statistics1_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _statistics2_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _statistics3_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      if ( _e == _chart_autoUpdateEvent_xjal ) return EVENT_TIMEOUT_MODE_CYCLIC;
      return super.getModeOf( _e );
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public double getFirstOccurrenceTime( EventTimeout _e ) {
      double _t;
      if ( _e == _Station1DS_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t, SECOND );
        return _t;
      }
      if ( _e == _Station2DS_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t, SECOND );
        return _t;
      }
```

```
      if ( _e == _Station3DS_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if ( _e == _statistics1_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if ( _e == _statistics2_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if ( _e == _statistics3_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if ( _e == _chart_autoUpdateEvent_xjal ) {
        _t =
0
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      return super.getFirstOccurrenceTime( _e );
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public double evaluateTimeoutOf( EventTimeout _e ) {
      double _t;
      if( _e == _Station1DS_autoUpdateEvent_xjal) {
        _t =
1
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if( _e == _Station2DS_autoUpdateEvent_xjal) {
        _t =
1
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if( _e == _Station3DS_autoUpdateEvent_xjal) {
        _t =
1
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if( _e == _statistics1_autoUpdateEvent_xjal) {
        _t =
1
;
        _t = toModelTime( _t , SECOND );
        return _t;
      }
      if( _e == _statistics2_autoUpdateEvent_xjal) {
        _t =
```

```
1
;
    _t = toModelTime( _t, SECOND );
    return _t;
  }
  if( _e == _statistics3_autoUpdateEvent_xjal) {
    _t =
1
;
    _t = toModelTime( _t, SECOND );
    return _t;
  }
  if( _e == _chart_autoUpdateEvent_xjal) {
    _t =
1
;
    _t = toModelTime( _t, SECOND );
    return _t;
  }
  return super.evaluateTimeoutOf( _e );
}

@Override
@AnyLogicInternalCodegenAPI
public void executeActionOf( EventTimeout _e ) {
  if ( _e == _Station1DS_autoUpdateEvent_xjal ) {
    Station1DS.update();
    return;
  }
  if ( _e == _Station2DS_autoUpdateEvent_xjal ) {
    Station2DS.update();
    return;
  }
  if ( _e == _Station3DS_autoUpdateEvent_xjal ) {
    Station3DS.update();
    return;
  }
  if ( _e == _statistics1_autoUpdateEvent_xjal ) {
    statistics1.update();
    return;
  }
  if ( _e == _statistics2_autoUpdateEvent_xjal ) {
    statistics2.update();
    return;
  }
  if ( _e == _statistics3_autoUpdateEvent_xjal ) {
    statistics3.update();
    return;
  }
  if ( _e == _chart_autoUpdateEvent_xjal ) {
    chart.updateData();
    return;
  }
  super.executeActionOf( _e );
}

/** Internal constant, shouldn't be accessed by user */
@AnyLogicInternalCodegenAPI
protected static final short _STATECHART_ELEMENT_NEXT_ID_xjal = 0;
// Embedded Objects

@AnyLogicInternalCodegenAPI
private static final AgentAnimationSettings _bicycles_animationSettings_xjal =
    new AgentAnimationSettings(1000L, 1000000000L);

public String getNameOf( Agent ao ) {
  return super.getNameOf( ao );
}

public AgentAnimationSettings getAnimationSettingsOf( Agent ao ) {
  return super.getAnimationSettingsOf( ao );
}
```

```

    public class _bicycles_Population extends AgentArrayList<Bicycle> {
      _bicycles_Population( Agent owner ) {
        super( owner );
      }

      @AnyLogicInternalCodegenAPI
      public void callSetupParameters( Bicycle agent, int index ) {
        setupParameters_bicycles_xjal( agent, index );
      }

      @AnyLogicInternalCodegenAPI
      public void callCreate( Bicycle agent, int index ) {
        create_bicycles_xjal( agent, index );
      }

      @AnyLogicInternalCodegenAPI
      public boolean isPresentationEnabled() {
        return true;
      }


      public int NStation1() {
        return _bicycles_NStation1_xjal();
      }

      public int NStation2() {
        return _bicycles_NStation2_xjal();
      }

      public int NStation3() {
        return _bicycles_NStation3_xjal();
      }
    }

    @AnyLogicCustomProposalType(value = AnyLogicCustomProposalType.Label.POPULATION, customText = "Bicycle")
    public _bicycles_Population bicycles = new _bicycles_Population( this );

    public String getNameOf( AgentList<?> aolist ) {
      if( aolist == bicycles ) return "bicycles";
      return super.getNameOf( aolist );
    }

    public AgentAnimationSettings getAnimationSettingsOf( AgentList<?> aolist ) {
      if( aolist == bicycles ) return _bicycles_animationSettings_xjal;
      return super.getAnimationSettingsOf( aolist );
    }

    /**
     * This method creates and adds new embedded object in the replicated embedded object collection bicycles<br>
     * @return newly created embedded object
     */
    public Bicycle add_bicycles() {
      int index = bicycles.size();
      Bicycle _result_xjal = instantiate_bicycles_xjal( index );
      bicycles.callSetupParameters( _result_xjal, index );
      bicycles.callCreate( _result_xjal, index );
      _result_xjal.start();
      return _result_xjal;
    }

    /**
     * This method creates and adds new embedded object in the replicated embedded object collection bicycles<br>
     * This method uses given parameter values to setup created embedded object<br>
     * Index of this new embedded object instance can be obtained through calling <code>bicycles.size()</code> method <strong>before</strong> this method is called
     * @param Rate12
     * @param Rate13
```

```
     * @param Rate21
     * @param Rate23
     * @param Rate31
     * @param Rate32
     * @param Start1
     * @param Start2
     * @param Start3
     * @return newly created embedded object
     */
    public Bicycle add_bicycles( double Rate12, double Rate13, double Rate21, double Rate23, double Rate31, double Rate32, double Start1, double Start2, double Start3 ) {
      int index = bicycles.size();
      Bicycle _result_xjal = instantiate_bicycles_xjal( index );
      // Setup parameters
      _result_xjal.markParametersAreSet();
      _result_xjal.Rate12 = Rate12;
      _result_xjal.Rate13 = Rate13;
      _result_xjal.Rate21 = Rate21;
      _result_xjal.Rate23 = Rate23;
      _result_xjal.Rate31 = Rate31;
      _result_xjal.Rate32 = Rate32;
      _result_xjal.Start1 = Start1;
      _result_xjal.Start2 = Start2;
      _result_xjal.Start3 = Start3;
      // Finish embedded object creation
      bicycles.callCreate( _result_xjal, index );
      _result_xjal.start();
      return _result_xjal;
    }

    /**
     * This method removes the given embedded object from the replicated embedded object collection bicycles<br>
     * The given object is destroyed, but not immediately in common case.
     * @param object the active object - element of replicated embedded object bicycles - which should be removed
     * @return <code>true</code> if object was removed successfully, <code>false</code> if it doesn't belong to bicycles
     */
    public boolean remove_bicycles( Bicycle object ) {
      if( ! bicycles._remove( object ) ) {
        return false;
      }
      object.removeFromFlowchart();
      object.setDestroyed();
      return true;
    }

    /**
     * Creates an embedded object instance and adds it to the end of replicated embedded object list<br>
     * <i>This method should not be called by user</i>
     */
    protected Bicycle instantiate_bicycles_xjal( final int index ) {
      Bicycle _result_xjal = new Bicycle( getEngine(), this, bicycles );

      bicycles._add( _result_xjal );

      return _result_xjal;
    }

    /**
     * Setups parameters of an embedded object instance<br>
     * This method should not be called by user
     */
    private void setupParameters_bicycles_xjal( final Bicycle self, final int index ) {
      self.Rate12 =
  2.0
  ;
      self.Rate13 =
```

```
3.0
;
    self.Rate21 =
1.0
;
    self.Rate23 =
1.0
;
    self.Rate31 =
1.0
;
    self.Rate32 =
2.0
;
    self.Start1 = self._Start1_DefaultValue_xjal();
    self.Start2 = self._Start2_DefaultValue_xjal();
    self.Start3 = self._Start3_DefaultValue_xjal();
  }

  /**
   * Setups an embedded object instance<br>
   * This method should not be called by user
   */
  private void create_bicycles_xjal(Bicycle self, final int index ) {
    self.setEnvironment( this );
    self.setXYZ( 400.0, 60.0, 0.0 );
    self.create();

    // Port connections
  }

  /**
   * <i>This method should not be called by user</i>
   */
  private int _bicycles_NStation1_xjal() {
    int _value = 0;
    for ( Bicycle item : bicycles ) {
      boolean _t =
item.inState(Bicycle.Station1)
;
      if ( _t ) {
        _value++;
      }
    }
    return _value;
  }
  /**
   * <i>This method should not be called by user</i>
   */
  private int _bicycles_NStation2_xjal() {
    int _value = 0;
    for ( Bicycle item : bicycles ) {
      boolean _t =
item.inState(Bicycle.Station2)
;
      if ( _t ) {
        _value++;
      }
    }
    return _value;
  }
  /**
   * <i>This method should not be called by user</i>
   */
  private int _bicycles_NStation3_xjal() {
    int _value = 0;
    for ( Bicycle item : bicycles ) {
      boolean _t =
item.inState(Bicycle.Station3)
;
      if ( _t ) {
        _value++;
```

```
          }
        }
        return _value;
      }
      // Analysis Data Elements
      @AnyLogicInternalCodegenAPI
      public DataSet _chart_expression0_dataSet_xjal = new DataSet( 51, new
          DataUpdater_xjal() {
        double _lastUpdateX = Double.NaN;
        @Override
        public void update( DataSet _d ) {
          if ( time() == _lastUpdateX ) { return; }
          _d.add( time(), __chart_expression0_dataSet_xjal_YValue() );
          _lastUpdateX = time();
        }
      } );
      /**
       * <i>This method should not be called by user</i>
       */
      @AnyLogicInternalCodegenAPI
      private double __chart_expression0_dataSet_xjal_YValue() {
        return
    bicycles.NStation1()
    ;
      }

      @AnyLogicInternalCodegenAPI
      public DataSet _chart_expression1_dataSet_xjal = new DataSet( 51, new
          DataUpdater_xjal() {
        double _lastUpdateX = Double.NaN;
        @Override
        public void update( DataSet _d ) {
          if ( time() == _lastUpdateX ) { return; }
          _d.add( time(), __chart_expression1_dataSet_xjal_YValue() );
          _lastUpdateX = time();
        }
      } );
      /**
       * <i>This method should not be called by user</i>
       */
      @AnyLogicInternalCodegenAPI
      private double __chart_expression1_dataSet_xjal_YValue() {
        return
    bicycles.NStation2()
    ;
      }

      @AnyLogicInternalCodegenAPI
      public DataSet _chart_expression2_dataSet_xjal = new DataSet( 51, new
          DataUpdater_xjal() {
        double _lastUpdateX = Double.NaN;
        @Override
        public void update( DataSet _d ) {
          if ( time() == _lastUpdateX ) { return; }
          _d.add( time(), __chart_expression2_dataSet_xjal_YValue() );
          _lastUpdateX = time();
        }
      } );
      /**
       * <i>This method should not be called by user</i>
       */
      @AnyLogicInternalCodegenAPI
      private double __chart_expression2_dataSet_xjal_YValue() {
        return
    bicycles.NStation3()
    ;
      }

      public DataSet Station1DS = new DataSet( 50, new DataUpdater_xjal() {
        double _lastUpdateX = Double.NaN;
        @Override
        public void update( DataSet _d ) {
```

```
        if ( time() == _lastUpdateX ) { return; }
        _d.add( time(), _Station1DS_YValue() );
        _lastUpdateX = time();
      }
      @Override
      public double getDataXValue() {
        return time();
      }
    } );

    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    private double _Station1DS_YValue() {
      return
  bicycles.NStation1()
  ;
    }

    public DataSet Station2DS = new DataSet( 50, new DataUpdater_xjal() {
      double _lastUpdateX = Double.NaN;
      @Override
      public void update( DataSet _d ) {
        if ( time() == _lastUpdateX ) { return; }
        _d.add( time(), _Station2DS_YValue() );
        _lastUpdateX = time();
      }
      @Override
      public double getDataXValue() {
        return time();
      }
    } );

    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    private double _Station2DS_YValue() {
      return
  bicycles.NStation2()
  ;
    }

    public DataSet Station3DS = new DataSet( 50, new DataUpdater_xjal() {
      double _lastUpdateX = Double.NaN;
      @Override
      public void update( DataSet _d ) {
        if ( time() == _lastUpdateX ) { return; }
        _d.add( time(), _Station3DS_YValue() );
        _lastUpdateX = time();
      }
      @Override
      public double getDataXValue() {
        return time();
      }
    } );

    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    private double _Station3DS_YValue() {
      return
  bicycles.NStation3()
  ;
    }

    public StatisticsContinuous statistics1 = new StatisticsContinuous( new
        DataUpdater_xjal() {
      double _lastUpdateX = Double.NaN;
      @Override
```

```
    public void update( StatisticsContinuous _d ) {
      if ( time() == _lastUpdateX ) { return; }
      _d.add( _statistics1_Value(), time() );
      _lastUpdateX = time();
    }
  } );

  /**
   * <i>This method should not be called by user</i>
   */
  @AnyLogicInternalCodegenAPI
  private double _statistics1_Value() {
    return
NS1
;
  }

  public StatisticsContinuous statistics2 = new StatisticsContinuous( new
      DataUpdater_xjal() {
    double _lastUpdateX = Double.NaN;
    @Override
    public void update( StatisticsContinuous _d ) {
      if ( time() == _lastUpdateX ) { return; }
      _d.add( _statistics2_Value(), time() );
      _lastUpdateX = time();
    }
  } );

  /**
   * <i>This method should not be called by user</i>
   */
  @AnyLogicInternalCodegenAPI
  private double _statistics2_Value() {
    return
NS2
;
  }

  public StatisticsContinuous statistics3 = new StatisticsContinuous( new
      DataUpdater_xjal() {
    double _lastUpdateX = Double.NaN;
    @Override
    public void update( StatisticsContinuous _d ) {
      if ( time() == _lastUpdateX ) { return; }
      _d.add( _statistics3_Value(), time() );
      _lastUpdateX = time();
    }
  } );

  /**
   * <i>This method should not be called by user</i>
   */
  @AnyLogicInternalCodegenAPI
  private double _statistics3_Value() {
    return
NS3
;
  }

  // View areas
  public ViewArea _origin_VA = new ViewArea( this, "[Origin]", 0, 0, ViewArea.
      TOP_LEFT, ViewArea.SPECIFIED_ZOOM, 1, 100, 100 );
  @Override
  @AnyLogicInternalCodegenAPI
  public int getViewAreas(Map<String, ViewArea> _output) {
    if ( _output != null ) {
      _output.put( "_origin_VA", this._origin_VA );
    }
    return 1 + super.getViewAreas( _output );
  }
  @AnyLogicInternalCodegenAPI
  protected static final int _bicycles_presentation = 1;
```

```
    @AnyLogicInternalCodegenAPI
    protected static final int _chart = 2;

    /** Internal constant, shouldn't be accessed by user */
    @AnyLogicInternalCodegenAPI
    protected static final int _SHAPE_NEXT_ID_xjal = 3;


    /**
     * Top-level presentation group id
     */
    @AnyLogicInternalCodegenAPI
    protected static final int _presentation = 0;

    @AnyLogicInternalCodegenAPI
    public boolean isPublicPresentationDefined() {
      return true;
    }

    @AnyLogicInternalCodegenAPI
    public boolean isEmbeddedAgentPresentationVisible( Agent _a ) {
      return super.isEmbeddedAgentPresentationVisible( _a );
    }
    /**
     * Top-level icon group id
     */
    @AnyLogicInternalCodegenAPI
    protected static final int _icon = -1;


    protected TimeStackChart chart;

    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    private void _bicycles_presentation_SetDynamicParams_xjal(
        ShapeEmbeddedObjectPresentation shape, int index ) {
      shape.setEmbeddedObject_xjal(
  bicycles.get( index )
  );
    }


    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    protected ShapeEmbeddedObjectPresentation
        _bicycles_presentation_createShapeWithStaticProperties_xjal( final int _index
        ) {
      ShapeEmbeddedObjectPresentation shape = new ShapeEmbeddedObjectPresentation(
          Main.this, SHAPE_DRAW_2D3D, true, 400.0, 60.0, 0.0, 0.0,
        true, true, bicycles.get( _index ) );
      return shape;
    }

    /**
     * <i>This method should not be called by user</i>
     */
    @AnyLogicInternalCodegenAPI
    private int _bicycles_presentation_Replication() {
      return
  bicycles.size()
  ;
    }

    protected ReplicatedShape<ShapeEmbeddedObjectPresentation> bicycles_presentation;

    private INetwork[] _getNetworks_xjal;

    @Override
```

```
    public INetwork[] getNetworks() {
      return _getNetworks_xjal;
    }


    private com.anylogic.engine.markup.Ground[] _getGrounds_xjal;

    @Override
    public com.anylogic.engine.markup.Ground[] getGrounds() {
      return _getGrounds_xjal;
    }


    private com.anylogic.engine.markup.RailwayNetwork[] _getRailwayNetworks_xjal;

    @Override
    public com.anylogic.engine.markup.RailwayNetwork[] getRailwayNetworks() {
      return _getRailwayNetworks_xjal;
    }

    @AnyLogicInternalCodegenAPI
    private void _createPersistentElementsBP0_xjal() {
    }

    @AnyLogicInternalCodegenAPI
    private void _createPersistentElementsAP0_xjal() {
      {
      DataSet _item;
      List<DataSet> _items = new ArrayList<DataSet>( 3 );
      _items.add( _chart_expression0_dataSet_xjal );
      _items.add( _chart_expression1_dataSet_xjal );
      _items.add( _chart_expression2_dataSet_xjal );
      List<String> _titles = new ArrayList<String>( 3 );
      _titles.add( "Station1" );
      _titles.add( "Station2" );
      _titles.add( "Station3" );
      List<Color> _colors = new ArrayList<Color>( 3 );
      _colors.add( lavender );
      _colors.add( gold );
      _colors.add( yellowGreen );
      chart = new TimeStackChart(
            Main.this, true, 40.0, 120.0,
           260.0, 210.0,
              null, null,
              50.0, 30.0,
           180.0, 120.0, white, black, black,
              30.0, Chart.SOUTH,

  100
              , Chart.WINDOW_MOVES_WITH_TIME, null, Chart.SCALE_AUTO
              , 0, Chart.GRID_DEFAULT, Chart.GRID_DEFAULT,
              darkGray, darkGray, _items, _titles, _colors );
      }
      bicycles_presentation = new ReplicatedShape<ShapeEmbeddedObjectPresentation>()
          {
        @Override
        public Class<ShapeEmbeddedObjectPresentation> getShapeClass() {
          return ShapeEmbeddedObjectPresentation.class;
        }

        @Override
        public int getReplication() {
          return _bicycles_presentation_Replication();
        }

        @Override
        public ShapeEmbeddedObjectPresentation createShapeWithStaticProperties_xjal(
            int index ) {
          ShapeEmbeddedObjectPresentation _e =
              _bicycles_presentation_createShapeWithStaticProperties_xjal( index );
          return _e;
        }
```

```java

      @Override
      public void setShapeDynamicProperties_xjal( ShapeEmbeddedObjectPresentation shape, int index ) {
        _bicycles_presentation_SetDynamicParams_xjal( shape, index );
      }
    };
  }


  // Static initialization of persistent elements
  {
    _createPersistentElementsBP0_xjal();
  }
  protected ShapeTopLevelPresentationGroup presentation;
  protected ShapeGroup icon;

  @Override
  @AnyLogicInternalCodegenAPI
  public Object getPersistentShape( int _shape ) {
    switch (_shape) {
      case _presentation: return presentation;
      case _icon: return icon;
      case _chart: return chart;
      case _bicycles_presentation: return bicycles_presentation;
      default: return super.getPersistentShape( _shape );
    }
  }

  @Override
  @AnyLogicInternalCodegenAPI
  public String getNameOfShape_xjal( Object _shape ) {
    try {
      if ( _shape == null ) return null;
      String _name_xjal;
      _name_xjal = checkNameOfShape_xjal( _shape, presentation, "presentation" ); if (_name_xjal != null) return _name_xjal;
      _name_xjal = checkNameOfShape_xjal( _shape, icon, "icon" ); if (_name_xjal != null) return _name_xjal;
      _name_xjal = checkNameOfShape_xjal( _shape, chart, "chart" ); if (_name_xjal != null) return _name_xjal;
      _name_xjal = checkNameOfShape_xjal( _shape, bicycles_presentation, "bicycles_presentation" ); if (_name_xjal != null) return _name_xjal;
    } catch (Exception e) {
      return null;
    }
    return super.getNameOfShape_xjal( _shape );
  }

  @AnyLogicInternalCodegenAPI
  private void drawModelElements_PlainVariables_xjal(Panel _panel, Graphics2D _g, boolean _publicOnly, boolean _isSuperClass ) {
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 240, 400, 10, 0, "NS1", NS1, false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 240, 450, 10, 0, "NS2", NS2, false );
    }
    if (!_publicOnly) {
      drawPlainVariable( _panel, _g, 240, 490, 10, 0, "NS3", NS3, false );
    }
  }

  @AnyLogicInternalCodegenAPI
  private void drawModelElements_DataElements_xjal(Panel _panel, Graphics2D _g, boolean _publicOnly, boolean _isSuperClass ) {
    if (!_publicOnly) {
      drawDataset( _panel, _g, 60, 400, 15, 0, "Station1DS", Station1DS );
    }
    if (!_publicOnly) {
      drawDataset( _panel, _g, 60, 450, 15, 0, "Station2DS", Station2DS );
    }
```

```
      if (!_publicOnly) {
        drawDataset( _panel, _g, 60, 500, 15, 0, "Station3DS", Station3DS );
      }
      if (!_publicOnly) {
        drawStatistics( _panel, _g, 330, 400, 15, 0, "statistics1", statistics1 );
      }
      if (!_publicOnly) {
        drawStatistics( _panel, _g, 330, 450, 15, 0, "statistics2", statistics2 );
      }
      if (!_publicOnly) {
        drawStatistics( _panel, _g, 330, 500, 15, 0, "statistics3", statistics3 );
      }
    }

    @AnyLogicInternalCodegenAPI
    private void drawModelElements_EmbeddeObjects_xjal(Panel _panel, Graphics2D _g, boolean _publicOnly, boolean _isSuperClass ) {
      // Embedded object "bicycles"
      if (!_publicOnly) {
        drawEmbeddedObjectModelDefault( _panel, _g, 100 , 60 , 10, 0, "bicycles", this.bicycles );
      }
    }

    @AnyLogicInternalCodegenAPI
    private void drawModelElements_AgentLinks_xjal(Panel _panel, Graphics2D _g, boolean _publicOnly, boolean _isSuperClass ) {
      if (_publicOnly) { return; }
        drawLinkToAgent( _panel, _g, 50, -50, 15, 0, "connections", true, connections );
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public void drawModelElements( Panel _panel, Graphics2D _g, boolean _publicOnly, boolean _isSuperClass ) {
     super.drawModelElements( _panel, _g, _publicOnly, true );
      drawModelElements_PlainVariables_xjal( _panel, _g, _publicOnly, _isSuperClass );
      drawModelElements_DataElements_xjal( _panel, _g, _publicOnly, _isSuperClass );
      drawModelElements_EmbeddeObjects_xjal( _panel, _g, _publicOnly, _isSuperClass );
      drawModelElements_AgentLinks_xjal( _panel, _g, _publicOnly, _isSuperClass );
    }

    @AnyLogicInternalCodegenAPI
    private boolean onClickModelAt_EmbeddedObjects_xjal( Panel _panel, double _x, double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
      if ( !bicycles.isEmpty() && modelElementContains(_x, _y, 100, 60) ) {
        if ( _clickCount == 2 ) {
          _panel.browseAgent_xjal( 100, 60, this, "bicycles" );
        } else {
          _panel.addInspect( 100, 60, this, "bicycles" );
        }
        return true;
      }
      return false;
    }

    @AnyLogicInternalCodegenAPI
    private boolean onClickModelAt_AgentLinks_xjal( Panel _panel, double _x, double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
      if ( modelElementContains(_x, _y, 50, -50) ) {
          _panel.addInspect_xjal( 50, -50, this, "connections", Panel.INSPECT_CONNECTIONS_xjal );
        return true;
      }
      return false;
    }


    @AnyLogicInternalCodegenAPI
```

```java
    private boolean onClickModelAt_PlainVariables_xjal( Panel _panel, double _x, double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
      if( !_publicOnly && modelElementContains(_x, _y, 240, 400) ) {
        _panel.addInspect( 240, 400, this, "NS1" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 240, 450) ) {
        _panel.addInspect( 240, 450, this, "NS2" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 240, 490) ) {
        _panel.addInspect( 240, 490, this, "NS3" );
        return true;
      }
      return false;
    }

    @AnyLogicInternalCodegenAPI
    private boolean onClickModelAt_DataElements_xjal( Panel _panel, double _x, double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
      if( !_publicOnly && modelElementContains(_x, _y, 60, 400) ) {
        _panel.addInspect( 60, 400, this, "Station1DS" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 60, 450) ) {
        _panel.addInspect( 60, 450, this, "Station2DS" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 60, 500) ) {
        _panel.addInspect( 60, 500, this, "Station3DS" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 330, 400) ) {
        _panel.addInspect( 330, 400, this, "statistics1" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 330, 450) ) {
        _panel.addInspect( 330, 450, this, "statistics2" );
        return true;
      }
      if( !_publicOnly && modelElementContains(_x, _y, 330, 500) ) {
        _panel.addInspect( 330, 500, this, "statistics3" );
        return true;
      }
      return false;
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public boolean onClickModelAt( Panel _panel, double _x, double _y, int _clickCount, boolean _publicOnly, boolean _isSuperClass ) {
      if ( onClickModelAt_EmbeddedObjects_xjal( _panel, _x, _y, _clickCount, _publicOnly, _isSuperClass ) ) { return true; }
      if ( onClickModelAt_AgentLinks_xjal( _panel, _x, _y, _clickCount, _publicOnly, _isSuperClass ) ) { return true; }
      if ( onClickModelAt_PlainVariables_xjal( _panel, _x, _y, _clickCount, _publicOnly, _isSuperClass ) ) { return true; }
      if ( onClickModelAt_DataElements_xjal( _panel, _x, _y, _clickCount, _publicOnly, _isSuperClass ) ) { return true; }
      return super.onClickModelAt( _panel, _x, _y, _clickCount, _publicOnly, true );
    }



    /**
     * Constructor
     */
    public Main( Engine engine, Agent owner, AgentList<? extends Main> ownerPopulation ) {
      super( engine, owner, ownerPopulation );
      if (isTopLevelClass_xjal( Main.class )) {
        instantiateBaseStructure_xjal();
```

```
    }
  }

  @AnyLogicInternalCodegenAPI
  public void onOwnerChanged_xjal() {
    super.onOwnerChanged_xjal();
    setupReferences_xjal();
  }

  @AnyLogicInternalCodegenAPI
  public void instantiateBaseStructure_xjal() {
    super.instantiateBaseStructure_xjal();
    setupReferences_xjal();
  }

  @AnyLogicInternalCodegenAPI
  private void setupReferences_xjal() {
  }

  /**
   * Simple constructor. Please add created agent to some population by calling goToPopulation() function
   */
  public Main() {
  }

  /**
   * Creating embedded object instances
   */
  @AnyLogicInternalCodegenAPI
  private void instantiatePopulations_xjal() {
    {
      int _cnt =
10
;
      for ( int i = bicycles.size(); i < _cnt; i++ ) {
        instantiate_bicycles_xjal( i );
      }
    }
  }

  @Override
  @AnyLogicInternalCodegenAPI
  public void create() {
    super.create();
    // Creating embedded object instances
    instantiatePopulations_xjal();
    // Assigning initial values for plain variables
    setupPlainVariables_Main_xjal();
    // Dynamic initialization of persistent elements
    _createPersistentElementsAP0_xjal();
    presentation = new ShapeTopLevelPresentationGroup( Main.this, true, 0, 0, 0, 0, bicycles_presentation, chart );
    icon = new ShapeGroup( Main.this, true, 0, 0, 0  );
    // Creating contents for replicated shapes
    bicycles_presentation.createShapes();
    // Creating embedded object instances
    instantiatePopulations_xjal();
    // Environments setup
    {
      double _x_xjal =
500
;
      double _y_xjal =
500
;
      double _z_xjal =
0
;
      setupSpace( _x_xjal, _y_xjal, _z_xjal );
    }
    disableSteps();
```

```
      setNetworkUserDefined();
      setLayoutType( LAYOUT_RANDOM );
      // Port connectors with non-replicated objects
      // Creating replicated embedded objects
      bicycles.setEnvironment( this );
      for ( int i = 0; i < bicycles.size(); i++ ) {
        setupParameters_bicycles_xjal( bicycles.get(i), i );
        create_bicycles_xjal( bicycles.get(i), i );
      }
      setupInitialConditions_xjal( Main.class );
      if (isTopLevelClass_xjal( Main.class )) {
        onCreate();
      }
    }

    @AnyLogicInternalCodegenAPI
    public void setupExt_xjal(AgentExtension _ext) {
      // Agent properties setup
      if ( _ext instanceof ExtAgentWithSpatialMetrics && _ext instanceof
          ExtWithSpaceType ) {
        double _value;
        _value =
  10
  ;
        ((ExtAgentWithSpatialMetrics) _ext).setSpeed( _value, MPS );
      }
    }

    @Override
    @AnyLogicInternalCodegenAPI
    public void start() {
      super.start();
      _Station1DS_autoUpdateEvent_xjal.start();
      _Station2DS_autoUpdateEvent_xjal.start();
      _Station3DS_autoUpdateEvent_xjal.start();
      _statistics1_autoUpdateEvent_xjal.start();
      _statistics2_autoUpdateEvent_xjal.start();
      _statistics3_autoUpdateEvent_xjal.start();
      _chart_autoUpdateEvent_xjal.start();
      applyLayout();
      for (Agent embeddedObject : bicycles){
        embeddedObject.start();
      }
      if (isTopLevelClass_xjal( Main.class )) {
        onStartup();
      }
    }


    /**
     * Assigning initial values for plain variables<br>
     * <em>This method isn't designed to be called by user and may be removed in
         future releases.</em>
     */
    @AnyLogicInternalCodegenAPI
    public void setupPlainVariables_xjal() {
      setupPlainVariables_Main_xjal();
    }

    /**
     * Assigning initial values for plain variables<br>
     * <em>This method isn't designed to be called by user and may be removed in
         future releases.</em>
     */
    @AnyLogicInternalCodegenAPI
    private void setupPlainVariables_Main_xjal() {
      NS1 =
  0.0
  ;
      NS2 =
  0.0
  ;
```

```
      NS3 =
  0.0
  ;
    }

    // User API -----------------------------------------------------------
    @AnyLogicInternalCodegenAPI
    static LinkToAgentAnimationSettings _connections_commonAnimationSettings_xjal = new LinkToAgentAnimationSettingsImpl( false, black, 1.0, LINE_STYLE_SOLID, ARROW_NONE, 0.0 );

    public LinkToAgentCollection<Agent, Agent> connections = new LinkToAgentStandardImpl<Agent, Agent>(this, _connections_commonAnimationSettings_xjal);
    @Override
    public LinkToAgentCollection<? extends Agent, ? extends Agent> getLinkToAgentStandard_xjal() {
      return connections;
    }


    @AnyLogicInternalCodegenAPI
    public void drawLinksToAgents(boolean _underAgents_xjal, LinkToAgentAnimator _animator_xjal) {
      super.drawLinksToAgents(_underAgents_xjal, _animator_xjal);
      if ( _underAgents_xjal ) {
        _animator_xjal.drawLink( this, connections, true, true );
      }
    }

    public List<Object> getEmbeddedObjects() {
      List<Object> list = super.getEmbeddedObjects();
      if (list == null) {
        list = new LinkedList<Object>();
      }
      list.add( bicycles );
      return list;
    }

    public AgentList<? extends Main> getPopulation() {
      return (AgentList<? extends Main>) super.getPopulation();
    }

    public List<? extends Main> agentsInRange( double distance ) {
      return (List<? extends Main>) super.agentsInRange( distance );
    }

    @AnyLogicInternalCodegenAPI
    public void onDestroy() {
      _Station1DS_autoUpdateEvent_xjal.onDestroy();
      _Station2DS_autoUpdateEvent_xjal.onDestroy();
      _Station3DS_autoUpdateEvent_xjal.onDestroy();
      _statistics1_autoUpdateEvent_xjal.onDestroy();
      _statistics2_autoUpdateEvent_xjal.onDestroy();
      _statistics3_autoUpdateEvent_xjal.onDestroy();
      _chart_autoUpdateEvent_xjal.onDestroy();
      for (Agent _item : bicycles) {
        _item.onDestroy();
      }
    // Analysis Data Elements
      _chart_expression0_dataSet_xjal.destroyUpdater_xjal();
      _chart_expression1_dataSet_xjal.destroyUpdater_xjal();
      _chart_expression2_dataSet_xjal.destroyUpdater_xjal();
      Station1DS.destroyUpdater_xjal();
      Station2DS.destroyUpdater_xjal();
      Station3DS.destroyUpdater_xjal();
      statistics1.destroyUpdater_xjal();
      statistics2.destroyUpdater_xjal();
      statistics3.destroyUpdater_xjal();
      super.onDestroy();
    }

```

```


    }

}
```