CS 522
Assignment 5

Jean-Sébastien Bolduc
(jseb@cs.mcgill.ca)

December 16, 2002

Requirements for the assignment are described on the assignment web page. The solution consists of the two files:

- `Traffic.py`
- `Simulator.py`

The file `DEVS.py` is also required, but hasn't been modified.

Description

- **Generator DEVS**
  This DEVS generates `Car` objects according to a uniform distribution. This DEVS has a single output port, to be connected to a Queue DEVS. Since it has no input ports, no external transition will ever occur in this DEVS.
  The main challenge here is to keep track of the global simulation time, so `Car` objects can be initialized properly. In our case, we keep track of the global time in the attribute `globalTime` (it would be cleaner to use the state for this, as in our case the state is unused). A first idea would be to increment `globalTime` with `elapsed` in the internal transition function. This will not work, since in the DEVS formalism `elapsed` is reset to 0 *before* the actual internal transition.
  The solution is thus to store the last value returned by the time advance function in an attribute `timeFlag`, and use this attribute to increment `globalTime`.
- **Queue DEVS**
  This DEVS has one input port to receive `Car` objects from a Generator DEVS. It also needs both an input and output port to communicate with the Junction DEVS.
  The state of the Queue DEVS comprises a queue of cars, and an integer reflecting the progress of negotiations with the Junction DEVS. We can summarize negotiations from the Queue DEVS' side as follows:
  1. If the queue is not empty, the Queue DEVS sends an "authorization-request" message to the junction DEVS. This request consists in an integer, the aggressiveness factor of the car at the head of the queue.
  2. The Queue DEVS then enters a passive state, where it waits for an "authorization-granted" message from the Junction DEVS (this message is simply the integer 1).
  3. The head car is then dequeud ans sent to the Junction DEVS.
  4. Go back to first step.
- **Junction DEVS**
  This DEVS has a pair of input/output ports to communicate with two Queue DEVS (the two lanes). It also has an output port to send `Car` objects to the Statistics DEVS.
  The state of the Junction DEVS comprises two entries to reflect the status of the requests from both lanes, an entry to hold the car being processed, and an entry to remember the time spent so far on the current car. This last entry is necessary since the Junction DEVS is likely to be interrupted any time by either Queue DEVS.
  The Junction DEVS remains busy with a car for a time interval K. After that, the car is sent to the Statistics DEVS. We can summarize negotiations from the Junction DEVS' side as follows:
  - If the DEVS is busy with a car, it just stores any incoming request in the proper state entry.
  - If the DEVS is idle (i.e., not busy with a car), it checks the status of the requests from both lanes:
    - If there are no unanswered request, the DEVS remains idle.
    - If there is only one unanswered request, then wait for a period of time ε and then send an "authorization-granted" message to the corresponding Queue DEVS.

- If there are two unanswered requests, then choose one of the lanes, and send an "authorization-granted" message to that lane after the required period of time. The choice of the lane and the period of time are both dictated by the straqy.

A `Car` object is expected to be received immediately after an "authorization-granted" message.
A `GenericJunction` DEVS is defined. Both strategies are defined by inheritance. Both inherited classes define the methods `chooselane` and `timeAdvance`.

- **Statistics DEVS**
  This DEVS has a single input port, to receive `Car` objects from the Junction DEVS. This is a passive DEVS, i.e. it has not internal transition function (and no output function). All it does is compute the transit time of every car it receives, and use it to update its attributes:
  - `numCars` — Number of cars that passed through
  - `sumT` — Sum of the transit times
  - `sumTSQUARE` — Sum of the square of transit times

  These attributes are used to compute the average and standard deviation of transit times.
  To compute the transit times, this DEVS also needs to keep track of the global time. This time we can use the first idea proposed for the Generator DEVS, as the `elapsed` attribute is reset to 0 *after* the actual external transition.

The coupled-DEVS connects the atomic-DEVS in the obvious manner. We use the default select function.

As for the simulator (file `Simulator.py`), it has been modified as suggested so the termination condition is based on the Statistics DEVS' `numCars` attribute.

Use

To run the simulation, use the following command line:

```
> python Traffic.py -h
```

to display help information, or

```
> python Traffic.py [s:int] [n:int] [r:int | -debug]
```

where

- `s`: Strategy to be used (default is 1)
  1. Survival of the Fittest strategy
  2. Zipper strategy
- `n`: Number of cars to count (termination condition, default is 50)
- `r`: Number of experiments at each traffic density point (default is 1)

If the `-debug` flag is not used, several experiments will be performed for various traffic densities (traffic densities specified by the list `MEANS` in the code). The output then consists of three columns (mean, average transit time, std deviation of transit times).

If the `-debug` flag is used, only one experiment is performed, with a traffic density corresponding to `MEANS[0]`. The output is then a detailed description of the simulation is progress.

By default, `MEANS = [1.5, 2.0, 2.5, 3.0, ...29.5, 30.0]`

Example

Below is the output obtained by typing `python Traffic.py s:2 n:2 -debug`:

```
* * * * * * * * * *  CLOCK: 0.925550

Generator B : create < ID:1, aggr:1, created:0.925550>
Queue B : receive < ID:1, aggr:1, created:0.925550>
Queue B : send authorization request (1)
Junction : receive authorization request from Queue B (1)

* * * * * * * * * *  CLOCK: 1.025550
```

```
Junction : send authorization to Queue B
Queue B : receive authorization
Queue B : send < ID:1, aggr:1, created:0.925550>
Junction : receive < ID:1, aggr:1, created:0.925550> from Queue B

* * * * * * * * * *  CLOCK: 1.554002

Generator A : create < ID:2, aggr:7, created:1.554002>
Queue A : receive < ID:2, aggr:7, created:1.554002>
Queue A : send authorization request (7)
Junction : receive authorization request from Queue A (7)

* * * * * * * * * *  CLOCK: 1.766945

Generator B : create < ID:3, aggr:5, created:1.766945>
Queue B : receive < ID:3, aggr:5, created:1.766945>
Queue B : send authorization request (5)
Junction : receive authorization request from Queue B (5)

* * * * * * * * * *  CLOCK: 3.164434

Generator A : create < ID:4, aggr:5, created:3.164434>
Queue A : receive < ID:4, aggr:5, created:3.164434>

* * * * * * * * * *  CLOCK: 3.713996

Generator A : create < ID:5, aggr:5, created:3.713996>
Queue A : receive < ID:5, aggr:5, created:3.713996>

* * * * * * * * * *  CLOCK: 4.144591

Generator B : create < ID:6, aggr:9, created:4.144591>
Queue B : receive < ID:6, aggr:9, created:4.144591>

* * * * * * * * * *  CLOCK: 4.820522

Generator A : create < ID:7, aggr:4, created:4.820522>
Queue A : receive < ID:7, aggr:4, created:4.820522>

* * * * * * * * * *  CLOCK: 5.733151

Generator B : create < ID:8, aggr:2, created:5.733151>
Queue B : receive < ID:8, aggr:2, created:5.733151>

* * * * * * * * * *  CLOCK: 5.816533

Generator A : create < ID:9, aggr:10, created:5.816533>
Queue A : receive < ID:9, aggr:10, created:5.816533>

* * * * * * * * * *  CLOCK: 6.025550

Junction : output < ID:1, aggr:1, created:0.925550>

* * * * * * * * * *  CLOCK: 6.125550

Junction : send authorization to Queue A
```

```
Queue A : receive authorization
Queue A : send < ID:2, aggr:7, created:1.554002>
Junction : receive < ID:2, aggr:7, created:1.554002> from Queue A
Queue A : send authorization request (5)
Junction : receive authorization request from Queue A (5)

* * * * * * * * * *  CLOCK: 7.230760

Generator B : create < ID:10, aggr:5, created:7.230760>
Queue B : receive < ID:10, aggr:5, created:7.230760>

* * * * * * * * * *  CLOCK: 8.088203

Generator A : create < ID:11, aggr:8, created:8.088203>
Queue A : receive < ID:11, aggr:8, created:8.088203>

* * * * * * * * * *  CLOCK: 9.018014

Generator B : create < ID:12, aggr:9, created:9.018014>
Queue B : receive < ID:12, aggr:9, created:9.018014>

* * * * * * * * * *  CLOCK: 9.148306

Generator A : create < ID:13, aggr:4, created:9.148306>
Queue A : receive < ID:13, aggr:4, created:9.148306>

* * * * * * * * * *  CLOCK: 10.355048

Generator A : create < ID:14, aggr:1, created:10.355048>
Queue A : receive < ID:14, aggr:1, created:10.355048>

* * * * * * * * * *  CLOCK: 10.775091

Generator B : create < ID:15, aggr:3, created:10.775091>
Queue B : receive < ID:15, aggr:3, created:10.775091>

* * * * * * * * * *  CLOCK: 11.125550

Junction : output < ID:2, aggr:7, created:1.554002>

Mean Transit Time : 7.335774
Std Deviation Transit Time : 3.161862
```

Results

We plot the average transit time in function of the mean μ (uniform distributions of Inter-arrival times for cars on both lanes are uniformly distributed in [μ-1, μ+1]).

In figure 1, we counted 100 cars, and we did 10 experiments at every point. We clearly see that when the traffic gets more important (i.e., when the mean μ gets smaller) the Zipper strategy is more interesting.

To evaluate the impact of the number of cars and the number of experiments at each point, we did two extra experiment. In figure 2, we counted 200 cars and did 10 experiments at every point, whereas in figure 3 we counted 100 cars and did 5 experiments at every point.

We note that even if the number of cars and the number of experiments at each point vary, the results are qualitatively the same.

For reference, we plot in figure 4 the standard deviation associated with the transit time as a function of mean, for the same experiment as in figure 1.

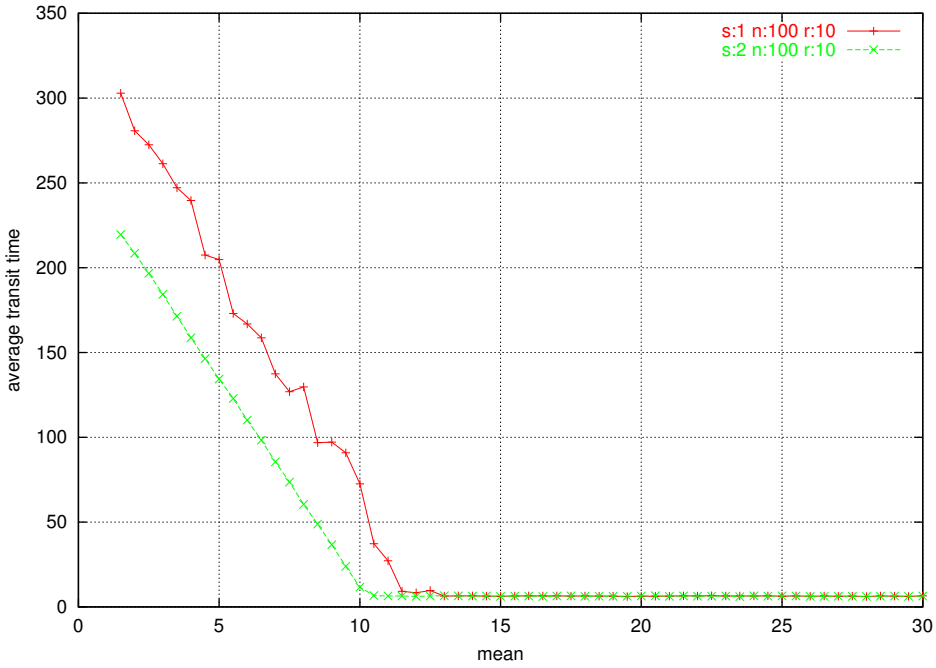

Figure 1: 100 cars, 10 experiments

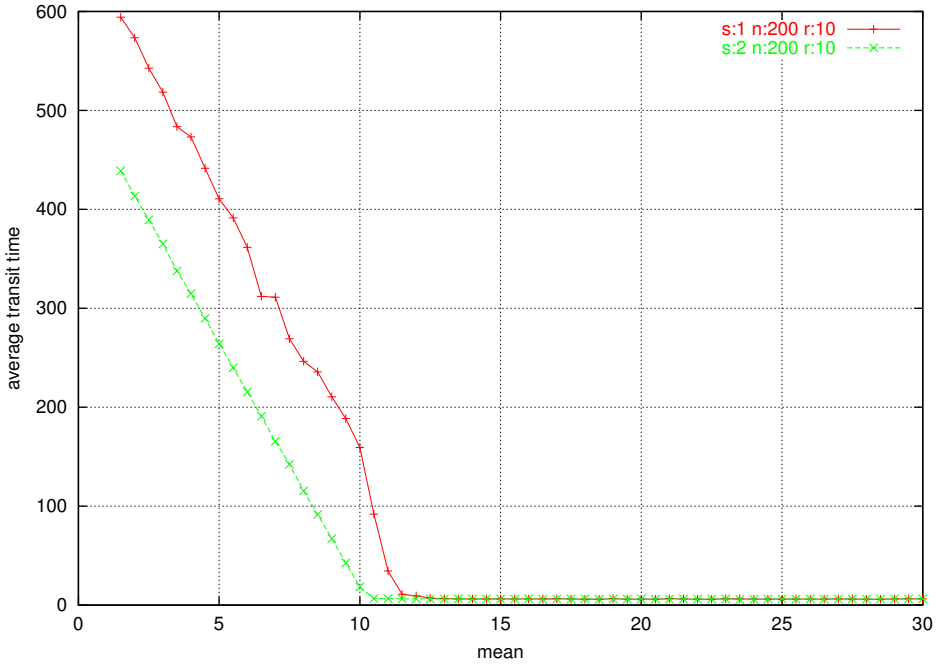

Figure 2: 200 cars, 10 experiments

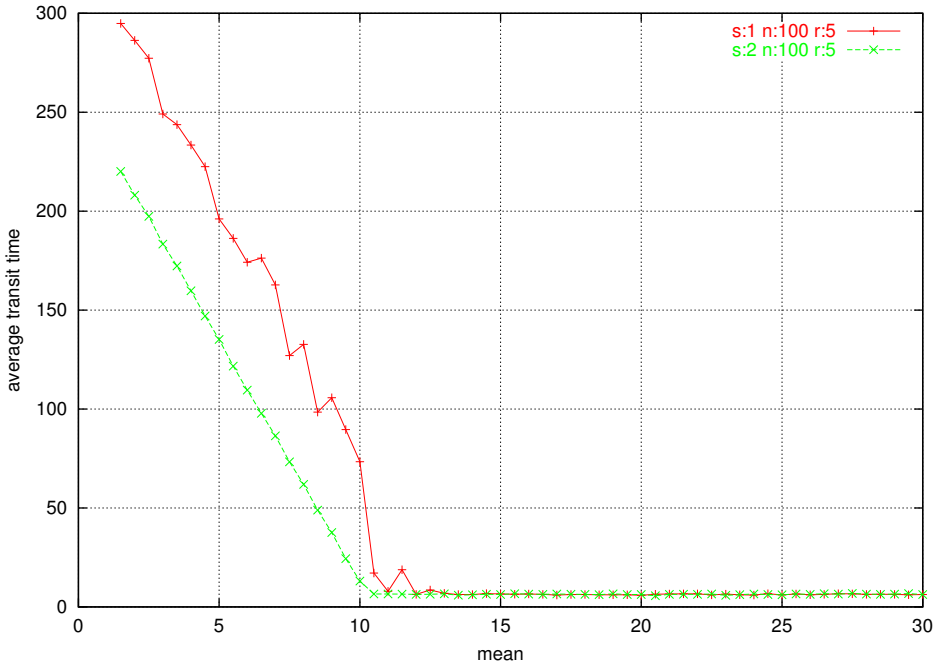

Figure 3: 100 cars, 5 experiments

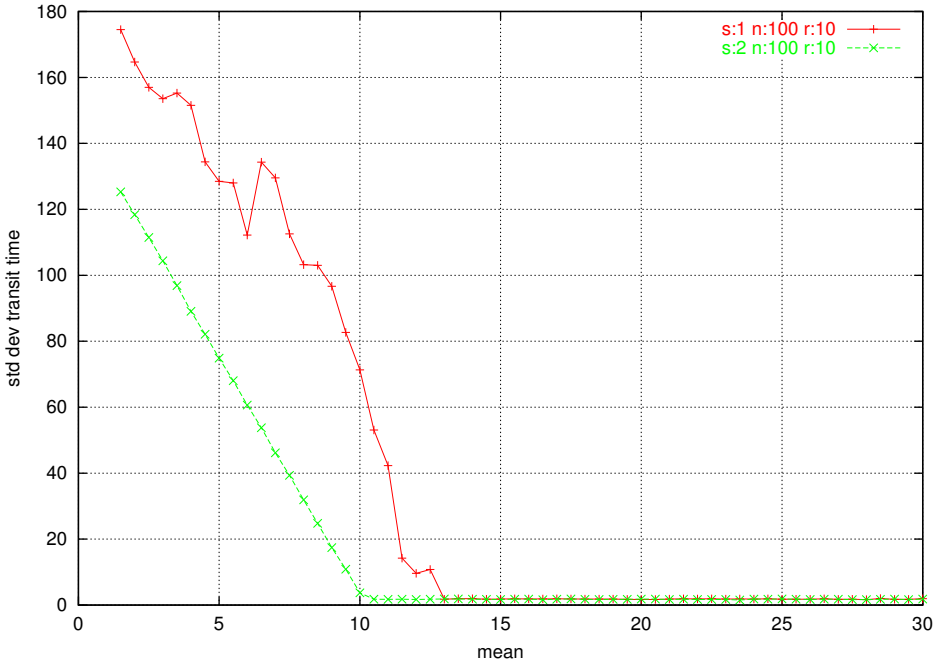

Figure 4: Standard Deviation