# EE126 Project Proposal: Traffic Modelling (TASEP)

Weijie Lyu (3035418676), Xinyi Liu (3035418507) Ganlin Zhang (3035418130) Teng Xu (3035418221)

## 1 Objective

1. Show two feasible methods that could be used for modelling.
2. Compare the two models.
3. Show some simulations of the elementary model. Come up with a set of interesting questions that the model might be able to solve.
4. Based on the questions, draft out the experiments that will be implemented next.

## 2 Traffic Setting & Assumptions

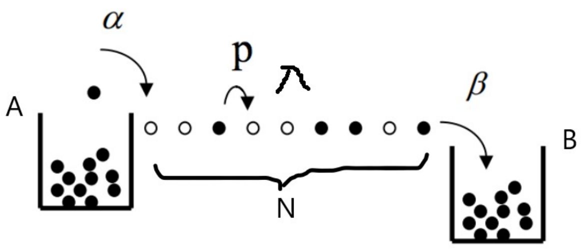


Fig. 2.0.1: Taken from `https://arxiv.org/pdf/0803.2625.pdf`

1. As shown in Fig.2.0.1, let's suppose there is one N-cell road, and cars are trying to get from point A to point B.
2. Each car has an **independent** exponential clock with rate $\lambda$ and every time their clock goes off, they are trying to move forward with probability p. However, if there is a car in front of them, they will not move. *Note: In the basic model we do not set different moving rate of the cars so all cars are identical.*
3. At the first and last cells, there are exponential clocks with rates $\alpha$ and $\beta$ that introduce and remove cars from the system, respectively.

## 3 Method 1: Universal Exponential Clock

### 3.1 State Space

As shown in Fig.3.1.2, consider a N-cell road with cells $i = 1, 2, \dots, N$. If a car occupies cell i at time n, it can be represented by $\tau_{i,n}$

$$\tau_{i,n} = \mathbb{1}\{A\, car\, occupies\, cell\, i\, at\, time\, n\}$$

where i is the cell concerned and n is the discrete time node of the current state. Therefore, A state from state space could be symbolically represented as $\{\tau_{i,n}\} = \{\tau_{1,n}, \tau_{2,n}, \tau_{3,n}, \dots \tau_{N,n}\}$.

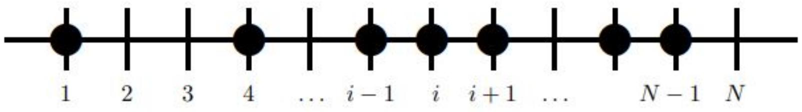


Fig. 3.1.2: One state from state space of the concerned discrete time markov chain discussed below, given time t = n, which can be represented as {1001...111...110}. Taken from `https://www.bristol.ac.uk/physics/media/theory-theses/dwandaru-brams-thesis.pdf`.

## 3.2 Categories of Events

We define that at each discrete time node, an event must happen. We define that there are 3 categories of events.

1. (Denote as $C_i$:) One of the car on the road (suppose in cell i) are attempting to move forward one cell. It succeeds if no cars in cell i+1, it fails if there is a car in cell i+1 and nothing happens.
2. (Denote as $A$:) The road is attempting to introduce a new car at the left side of cell 1. It succeed if the cell 1 has no car and it fails if the cell 1 has a car, in which case, nothing happens.
3. (Denote as $B$:) The road is attempting to remove a car at the right side of cell N. It succeed if the cell N has a car and it fails if the cell N has no car, in which case, nothing happens.

## 3.3 State Updating (Sequential Updating)

The model of ours follows the sequential updating [1]. Given the current state $\{\tau_{i,n}\}$, the next state $\{\tau_{i,n+1}\}$ can be decided by the following procedure:

1. Count # of 1s in $\{\tau_{i,n}\}$ as $k_n$ , then decide what event is happening:

$$\mathbb{P}(C_i) = \frac{\lambda}{k_n\lambda p + \alpha + \beta}, \forall\, i$$

$$\mathbb{P}(A) = \frac{\alpha}{k_n\lambda p + \alpha + \beta}$$

$$\mathbb{P}(B) = \frac{\beta}{k_n\lambda p + \alpha + \beta}$$

   The reasons will be explained in the CTMC scenario, now we can decide how to change the current state into the next state.

2. Change the state according to the events in 2.3 and get $\{\tau_{i,n+1}\}$.

## 3.4 Setting a Universal Exponential Clock

In order to convert the jumping DTMC model to the continuous time markov chain model, the right thing for us to do is to set up a universal exponential clock that runs off each time that certain event should be happened. Since we know that the events are all independent with each other, so there is a competition between $k_n$ cars' exponential clocks, A's and B's exponential clocks. Let $T_i$ be the ith interarrival time, $S_i$ be the ith arrival time, we have

$$T_i \sim Expo(k_n\lambda p + \alpha + \beta), \forall\, i \in \{1, 2, \ldots\}$$

$$S_i = \sum_{j=1}^{i} T_j \sim Poisson(k_n\lambda p + \alpha + \beta), \forall\, i \in \{1, 2, \ldots\}$$

According to the property of the splitting of Poisson process, at each $S_i$, the current state jump to different events according to the probability stated in 2.4.

# 4 Method 2: Bernoulli Approximation

## 4.1 State Space

We can use the same form with the state space using Method of exponential clock, however, the meaning of n has changed from "the nth event" to "the nth time inteval".

### 4.2 Parallel Updating

First we use small time interval $\delta t$ that small enough for the Bernoulli Approximation. In each time interval, there are three kinds of independent and parallel coin tosses are ongoing.

1. For each car in the first (N-1) cells: each car will attempt to move forward a cell *w.p.* $\lambda \delta t p$. If the next cell has no car, then the the car move forward; else, nothing happens.
2. Introducing new cars: the road will attempt to introduce a new car at cell l *w.p.* $\alpha \delta t$. If the cell 1 has no car, then the road introduce it; else, nothing happens.
3. Removing cars at cell N: the road will attempt to remove car at cell N *w.p.* $\beta \delta t$. If the cell N has a car, then the road remove it; else, nothing happens.

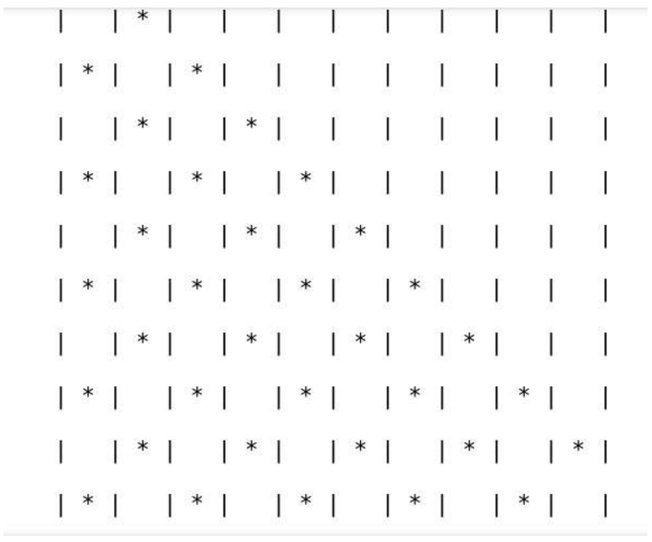
## 5 Comparing of the Two Methods

After analysing the pros and cons of the two methods, we choose the second method, since it has the following advantages:

1. It is time-saving. We do not need another function to simulate a Poisson process, which is painfully time-consuming as number of cars increases. Also, if each time we choose a random number and compare it with (n+2) numbers within [0,1], the accuracy will fall and the time complexity will rise.
2. The parallel form of updating is much more easier to code and trouble shooting in general.

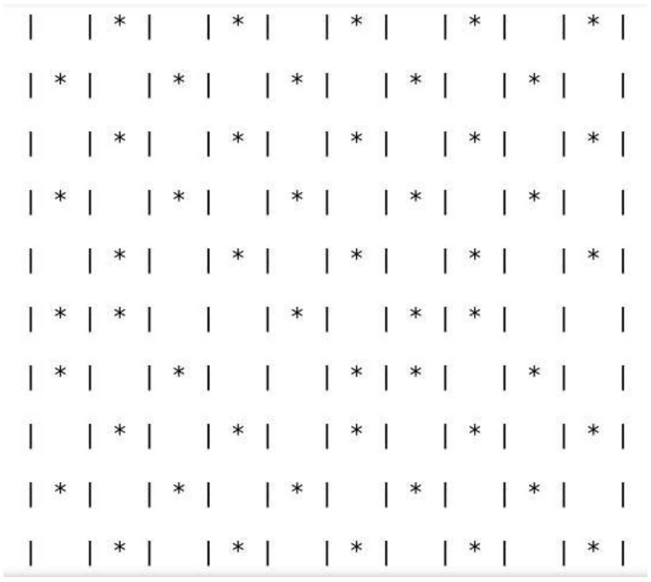
## 6 Simulation Result

We use "|" and "·" to visualize the states of one road at different time by printing out at each loop of the program *(See python code at Appendix A Listing 1)*. Here we present 3 snapshots of states at different time periods: the beginning of the simulation (no cars on the road), the transient states in the middle, and steady states as shown in Fig. 6.0.3, 6.0.4 and 6.0.5.



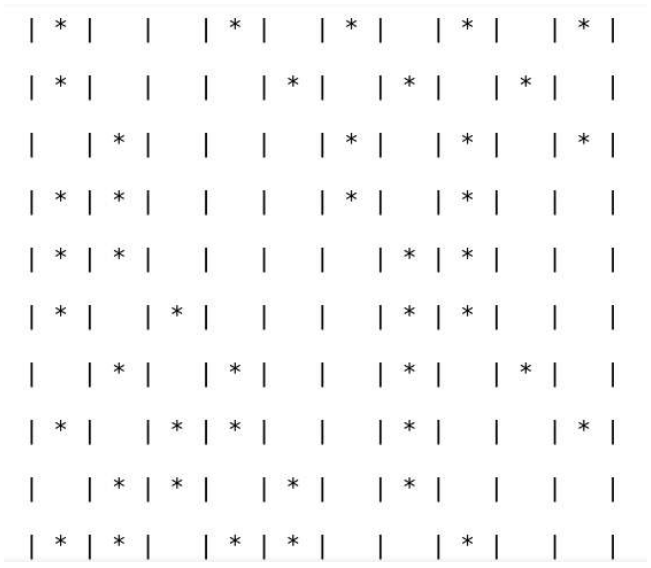
```
|   | * |   |   |   |   |   |   |   |   |
| * |   | * |   |   |   |   |   |   |   |
|   | * |   | * |   |   |   |   |   |   |
| * |   | * |   | * |   |   |   |   |   |
|   | * |   | * |   | * |   |   |   |   |
| * |   | * |   | * |   | * |   |   |   |
|   | * |   | * |   | * |   | * |   |   |
| * |   | * |   | * |   | * |   | * |   |
|   | * |   | * |   | * |   | * |   | * |
| * |   | * |   | * |   | * |   | * |   |
```

Fig. 6.0.3: Beginning of the simulation



```
|   | * |   | * |   | * |   | * |   | * |
| * |   | * |   | * |   | * |   | * |   |
|   | * |   | * |   | * |   | * |   | * |
| * |   | * |   | * |   | * |   | * |   |
|   | * |   | * |   | * |   | * |   | * |
| * | * |   |   | * |   | * | * |   |   |
| * |   | * |   |   | * | * |   | * |   |
|   | * |   | * |   | * |   | * |   | * |
| * |   | * |   | * |   | * |   | * |   |
|   | * |   | * |   | * |   | * |   | * |
```

Fig. 6.0.4: In the middle of the simulation



```
| * |   |   | * |   | * |   | * |   | * |
| * |   |   |   | * |   | * |   | * |   |
|   | * |   |   |   | * |   | * |   | * |
| * | * |   |   |   | * |   | * |   |   |
| * | * |   |   |   |   | * | * |   |   |
| * |   | * |   |   |   | * | * |   |   |
|   | * |   | * |   |   | * |   | * |   |
| * |   | * | * |   |   | * |   |   | * |
|   | * | * |   | * |   | * |   |   |   |
| * | * |   | * | * |   |   | * |   |   |
```

Fig. 6.0.5: Approaching an steady state

## 7 Questions & Further Steps

1. If there are different phases in this particular model? If so, what cause them and what insights will they bring to the real environment?
2. we are interested that what parameters will affect the following values in this model:
   (a) The number of contiguous cars (which indicate the blockages)
   (b) The number of cars in the system at a given time.
   (c) The stationary distributions of each cell (occupation density of a cell).
3. Increase the dimensions by adding cross roads with the following configuration:
   (a) Red lights that periodically switch to another road to prevent one road of cars from crossing.
   (b) No lights, two roads with same/different speed.
4. Try to analysis what effect it brings if we take in different speeds for different cars.
5. Try to simulate more real situations on roads, i.e. accidents, etc.

## A Associated codes

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import time
import random
import numpy as np
CAR_RATE = 3 # velocity of the car
TIME_CONST = 10000 #zoom in the time


# In[2]:


class road:
    def __init__(self, numOfCell, introRate, remvRate, index):
        #int numOfCell, int introRate, int remvRate, int index
        self.__introRate = introRate
        self.__remvRate = remvRate
        self.__index = index
        self.allCars = [] #cars in this road
        self.numOfCell = numOfCell
        self.cell = [0]*numOfCell #cells in this road
        self.passCar = 0 # number of the passing cars
    def introCar(self,interval):
        # introduce a new car to this road
        if ((random.random() < self.__introRate*interval)&(self.cell[0] == 0)):
            #print("new car!!!")
            self.cell[0] = 1
            newCar = car(0, CAR_RATE, 1, self.__index)
            self.allCars.insert(0,newCar)

    def remvCar(self, interval):
        # remove the car in the last cell of this road
        if ((random.random() < self.__remvRate*interval)&(self.cell[-1] == 1)):
            #print("remove car!!!")
            self.cell[-1] = 0
            self.allCars.pop()
            self.passCar += 1
    def showCar(self):
        for i in self.cell:
            if i == 0:
                print(" | "+" ",end="")
            else:
                print(" | "+"*",end="")
        print(" |\n")



# In[3]:


class car:
    def __init__(self, position, expClockRate, prob, index):
        #int position, int expClockRate, float prob, int index (which road)
        self.__position = position
        self.__expClockRate = expClockRate
        self.__prob = prob
        self.__road = allRoads[index]
    def moveForward(self, interval):
        if (random.random() < self.__expClockRate*interval)         &(random.random() < self.__prob)        &(self.__position < self.__road.numOfCell-1):
            # expo & not broken & in the road
            if (self.__road.cell[self.__position+1] == 0): # next cell is empty
                self.__road.cell[self.__position] = 0
                self.__position += 1
                self.__road.cell[self.__position] = 1 # move forward


# In[16]:
```

```python


allRoads = []
intervals = []
ourFirstRoad = road(10, 2, 2,len(allRoads))
allRoads.append(ourFirstRoad)
startTime = time.time()
lastTime = startTime
Stop = False
while not Stop:
    #print(ourFirstRoad.cell)
    currentTime = time.time()
    if currentTime -  startTime > 1:
        Stop = True
    interval = (currentTime - lastTime)*TIME_CONST # zoom in the time to simulate the actual ...
        case.
    intervals.append(interval)
    ourFirstRoad.introCar(interval)
    ourFirstRoad.remvCar(interval)
    for i in ourFirstRoad.allCars:
        i.moveForward(interval)
    lastTime = currentTime


# In[12]:


print(len(ourFirstRoad.allCars))


# In[13]:


print(ourFirstRoad.passCar)


# In[14]:


print(np.average(intervals))


# In[15]:


ourFirstRoad.showCar()


# In[ ]:





# In[ ]:
```

## B Bibliography

### References

[1] *https://www.bristol.ac.uk/physics/media/theory-theses/dwandaru-brams-thesis.pdf.*