Algorithm Comparison: Finding Optimum Path in Two-way Traffic Network Simulation

Meiqi Liu

Final Due Date: 20th Dec

1. Introduction

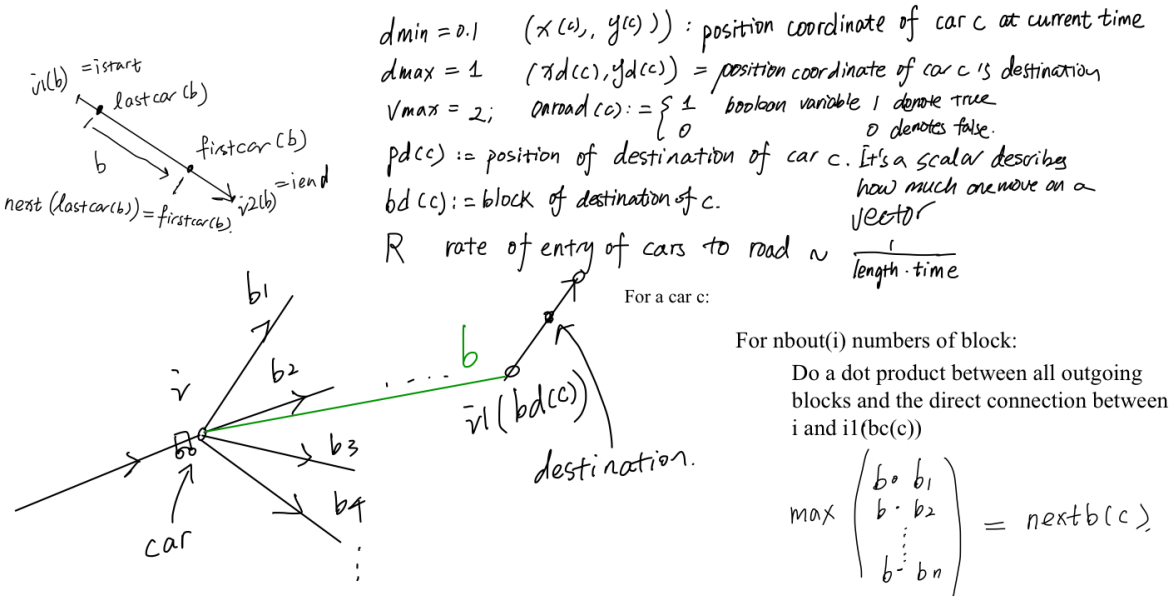
Living in an Urban area can be somehow troublesome to commute if the traffic occurs. To find the optimum next block at each intersection and to avoid traffic has become an important issue to solve for urban citizens to save time on transportation in order to have an easier life. This triggers my interest in using Matlab to simulate the traffic flow of a city. For convenience, the effect of traffic light will be ignored. Just with little variation to the single-way traffic system in Prof.Peskin's lecture notes I implement a two-way traffic network alike to my hometown Beijing. This project will compare the dot product and Dijkstra algorithm as two ways to solve the optimization problem and I will compute the average time taken for a certain number of cars, which are randomly generated, to arrive at their destinations in certain time interval and I will repeatedly simulate this to get different average values in multiple trials.

2.Methodology and Equations

The basic structure of simulation is the same as lecture note online. To run this program properly we need to set up variables:

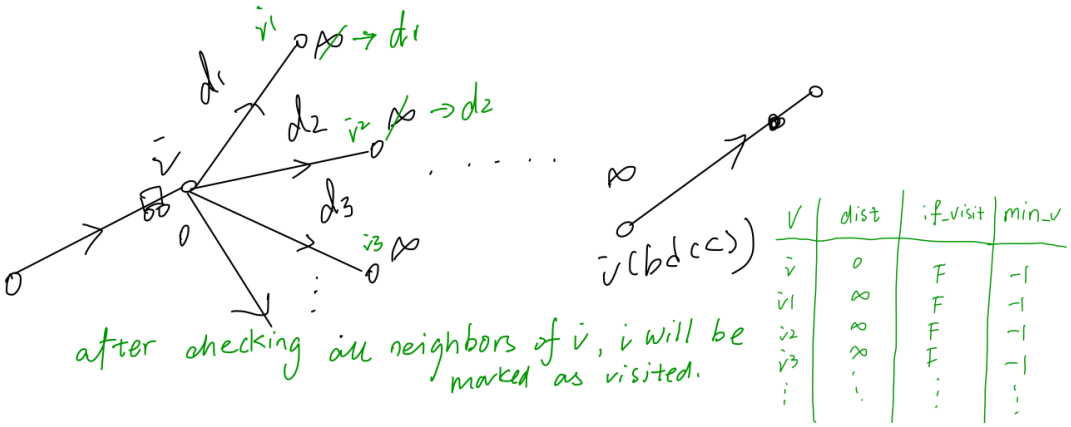
```
nC = 0;  % number of cars
ni = 5;  % number of intersections
nb = 10; % number of blocks.
i1 = [ 1; 2; 2; 4; 2; 3; 3; 4; 4; 5 ];   % index of i.
i2 = [ 2; 1; 4; 2; 3; 2; 4; 3; 5; 4 ];   % i1 & i2 are 2 ends of block; i1: start; i2: end;
nbin = zeros(ni,1); % nbin(i) := # of blocks entering intersection i.
nbout = zeros(ni,1); % nbout(i) := # of block leaving intersection i.
bin(i,j) = index of jth block entering intersection i, j = 1 .... nbin(i)
bout(i,j) = index of jth block leaving intersection i, j = 1 .... nbout(i)
L(b): length of block(b).
Ux(b) / Uy(b): unit vector on x/y direction of block b.
firstcar(b): car index of first car on block b.
lastcar(b): car index of last car on block b.
```

```
nextcar(c): index of a car in front of car c
tenter(c): the time of entering system of car c.
texit(c): the time of leaving of car c.
nextb(c): the next block of car c at some intersection.
Takem_taken(c): The time car c take to travel to its destination
xi(i): the x coordination of intersection i
yi(i)
xi = [0; 1.25; 5.5; 1; 4.7];
yi = [0; 2; 3; 5; 8.3];
```



Algorithm 1: The dot product method

The idea of the dot product method is simple. One can select the block with the least angle between the chosen block and a line that connects the current intersection and the starting point of the destination block. This way is closer to the strategy of choosing block with shortest distance. It has drawbacks when there is a single end path that doesn't connect to other blocks the algorithm would return impossible suggestion next block.



Algorithm 2: the Dijkstra method

The Dijkstra algorithm is used to find the distance of the shortest path between any two vertices a and b in the map. The basic idea is to pick the unvisited vertex with the low distance, calculate the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. It marks (set to true) visited vertex when done checking all neighbors. The function terminates when all vertices are visited.

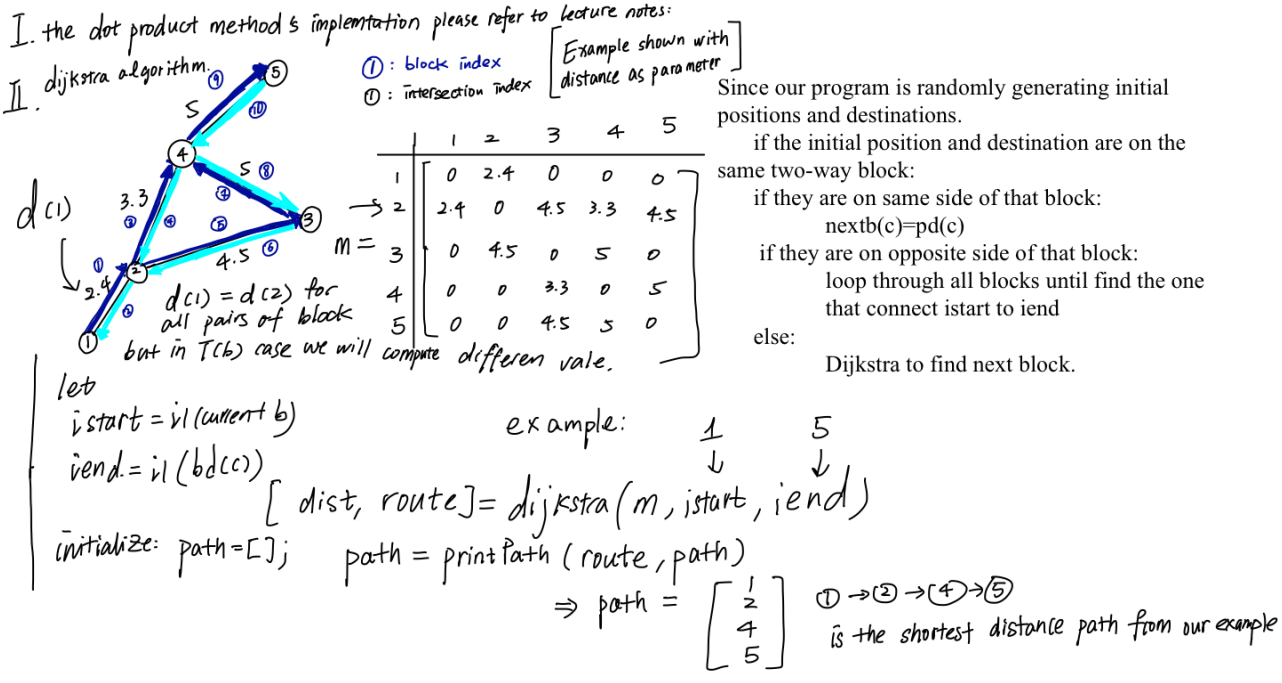
In our simulation, the distance should be replaced by a new parameter, called T(b) which is the time for a car to pass through this block and this can be calculated by

$$T(b) = \frac{L(b)}{V(\frac{L(b)}{n(b)+1})}$$

n(b):=use a function n to get the number of cars on a particular street b at a particular time
L(b):=L is an array of size nb; L(b) store the length of block b at index b
$d = \frac{L(n)}{n(b)+1}$:= d is the distance between car and car ahead
V(d):= the velocity of cars on block b according to its current level of congestion.
dmin, dmax , and vmax are global ;

$$V o\,(d) = 0 \qquad \text{if } d \le dmin$$
$$= V max \ \frac{log(d/dmin)}{log(dmax/dmin)} \qquad \text{if } dmin \le d \le dmax;$$
$$= \text{Vmax} \qquad \text{if } d \ge dmax$$

3.Implementation of Two Algorithm



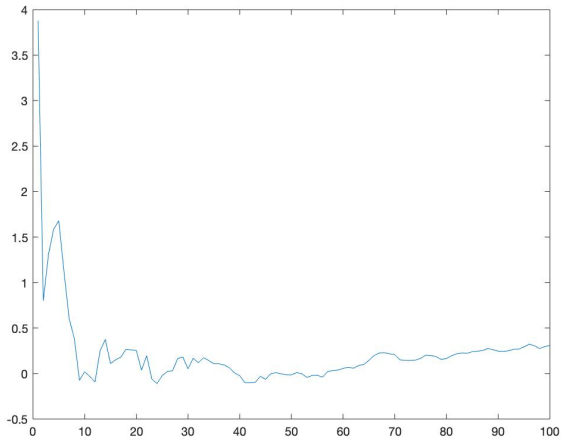
4.Numerical method

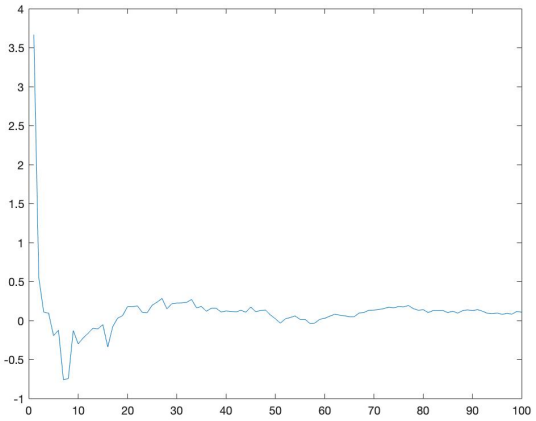
```
trial_res=zeros(n_trial,1);
for trial=1:n_trial
                                        for clock = 1: clockmax
                t = clock · dt;
                create_car;
                move_car;
                plotcar;
        end
 s=sum(time_taken);
 index=find(time_taken);
 avg=s/size(index,1);
disp(avg);
 trial_res(trial)=avg;

end
```

5.Result and Discussion

	Clock max	Dot_Product Method	Dijkstra Method
Trial 1 (dt=0.02)	400	15.6	4.08
Trial 2	800	6.43	3.93
Trial 3	1200	3.7267	2.9943
Trial 5	1600	4.378	4.235
Trial 6	2000	5.0373	3.77

For any random trial, for the best quality of animation, I decided to set clock max = 2000, dt=0.01. (except the trial 1 has dt =0.02 because in a time period of 400 there could be no car arrive at its destination). So from multiple trials, we can observe in each time interval to run this program, there will be around 15 cars that have arrived at their destinations. And by randomly picking 6 trials we list the data of average time taken for the 15 cars to arrive at their destination. As a result, the dot product algorithm shows more time taken required in all different time intervals that we record the data. The difference between the two times taken varies from trial to trial due to the randomness caused by the randomly creating car destination and initial positions, which might uneven distribution of short-distance trips and long-distance trips.





If we run 100 trials for each algorithm and plot the average time taken versus the trial index from 0 to 100. There will be around 1500 cars arrived at their destinations in order to collect enough data to estimate the randomness in our result due to the fact that our program is generating initial position and destination randomly and just knowing one trial of comparison does not guarantee that the optimum algorithm will give less time value. From the graph below, we can observe that both graphs have a drastic decreasing trend at the beginning between the trial index from 0 to 10 means the more car in systems the algorithm will perform better. Both graphs stabilize around t=0 which is the lowest time value in logic. However, the Dijkstra algorithm has an obvious close fluctuation to line t=0 (being slightly above since all negative time value could be a result

of error), while the dot product result is stabilized around somewhere close but above the line t=0. At the 100th trial, the time taken for the dot product method is in an increasing trend and stop at around t=0.25. Therefore, for all of the trials, the Dijkstra has less time taken if you have a trial to trial comparison and quantitively we can observe clearly that Dijkstra is not only time-efficient but also attains very stabilized performance.

6.Conclusion

In this matlab project, I try to solve an optimization problem of finding the optimum path that minimizes travel time with two algorithms: the dot product method from class and the dijkstra algorithm. The essence of the dot product method is to navigate by strategy of choosing the shortest distance path while the Dijkstra algorithm navigates base on the strategy of choosing the path with minimum travel time(the value is estimated by formulas). Overall results show that dot product method is less efficient compare to Dijkstra. With consideration of randomness brought in by generating the initial positions and destination randomly, I tested the average time taken for a car to arrive at its destination in 100 trials, each trial in a fixed time interval, with Dijkstra and dot product algorithm. The Dijkstra algorithm appears to take less time and attains a more stable performance of navigation.

Reference

Charles S. Peskin. (2019). Notes on traffic flow.