Communications in Computer and Information Science 514

Commenced Publication in 2007
Founding and Former Series Editors:
Alfredo Cuzzocrea, Dominik Ślęzak, and Xiaokang Yang

Reversible Express Bus Lanes Simulation Software

Timur Bakibayev[1](✉), Gulnara Bekmagambetova[2], and Asem Turarbek[3]

[1] ADA University, Baku, Azerbaijan
timurbakibayev@gmail.com
[2] Research Institute of Transport and Communication, Almaty, Kazakhstan
gulnarabek@yahoo.com
[3] Al-Farabi Kazakh National University, Almaty, Kazakhstan
turarbek_asem@mail.ru

Abstract. Exempting buses from other traffic seems to be an obviously good idea. On the other hand, it is not always possible to dedicate necessary space in the city for an extra lane. Therefore, our goal was to optimize the existing roads for giving priority to buses. Again, it would be nice to dedicate the rightmost lane to buses, but the other drivers would feel strong discomfort when turning right or parking. Moreover, this approach requires two lanes, one lane in each direction. So we tried to examine whether it makes sense to take the lane in the median area of the road, make it reversible and give it to buses.

In our work we simulated such a reversible bus lane. We simulate buses that have to change to the leftmost lane and then change back to the rightmost lane for a bus stop. This way we have found out the minimum distance between bus stops such that buses would be able to use it.

Keywords: Traffic simulation · Bus lanes · Reversible lanes

1 Introduction

Our aim was to simulate the traffic with a reversible bus lane in the median area of the road between the two opposite directions on a straight line of several lanes apart from the other streets.

In [1,2] A. Valencia and R. Fernandez's presented a model for public transport corridors (COItBUS, COIridor for BUSes), which represents the progression of vehicles in consideration of all sources of potential delays at links, intersections, and stops. And while others (e.g. see [3,4]) study the bus lanes with the bus stops next to the lanes, we tried to reserve a lane for buses without reconstructing the roads (e.g., building bus stops in the center of the roads). We decided to check whether it makes sense to dedicate a corridor for buses in the middle of the road keeping bus stops at the sidewalks.

R. Rome (Ed.): EGC 2015, CCIS 511, pp. 54–63, 2015.
DOI: 10.1007/978-3-319-25043-4_6

A microsimulation of the bus stops has been done by R. Fernandez in [5], where he showed the optimal distances to stop lines, distances between two buses, the optimal width of the bus stop, so that the bus stop could serve buses in parallel.

In this paper we show the minimum distances between bus stops in case of dedicating a reversible bus corridor in the median area of the road. For this reason we have implemented a software for a microsimulation of buses on such model taking into account the traffic density and the number of lanes. The idea is that if the bus stops are at the sidewalks, the bus has to go through the traffic to the leftmost lane (in the case of the right-hand traffic) and then return to the rightmost lane in order to stop. And this is the key point of our simulation - throughout the simulation the buses try to change to the leftmost lane, then return to the rightmost lane, and then stop. Meanwhile, other cars behave normally, drive a little faster, and switch lanes only in order to overtake other vehicles. These cars represent an obstacle for the buses to switch lanes.

Another advantage of dedicating the median area of the road to buses is having less emissions on sidewalks. The situation in Almaty is such that although the ecological problem is considered to be most important, there are a lot of buses that emit much more unburned gasoline than allowed. And even such a small distance would play a big role for pedestrians or cyclists.

The source codes can be found at: http://www.kuanysh.kz/revbus/source.zip
The binary can be found at: http://www.kuanysh.kz/revbus/release.zip
Moreover, Sect. 5.3 gives detailed information for the programmers.

2 Reversible Bus Lanes

A reversible lane is a lane in which traffic may flow in both directions, depending on time of the day or other conditions. Reversible lanes are used for optimizing traffic, mostly during rush hours. A good example is a connection of sleeping quarters and city centers. The flow to the city center in the morning is much stronger than in the opposite direction and is comparable to the flow to the sleeping quarters in the evening.

Reversible lane is a good solution when there is no space to widen the road any more because of buildings or in tunnels. Usually these lanes have a specially designed traffic lights to show the current direction. Otherwise some movable physical objects can be used for separating.

In our case we propose a reversible lane that is dedicated to only public transportation. It should forbid both directions twice a day for at least one hour for switching direction.

The question is why should this lane be forbidden to other vehicles? It could speed up the whole road, hence the buses would move faster even on right lanes. But the reality is that if this reversible lane is allowed to other traffic, it would not help in traffic jams. Frequent traffic jams in Kazakhstan are the main reason for introducing a separate lane for buses. On the other hand, this could stimulate some drivers use public transportation.

3 Simulation Conditions

Since the simulation depends on many circumstances, we assume the following facts:

1. The modeled system is a straight road that has no crossroads.
2. The speed of the cars is, on average, higher than the speed of the buses. Hence we decided that the maximum speed of the bus is 10 km/h less than the maximum speed of the cars. On the other hand, we only restrict the maximum speed, and the initial speed of both cars and buses is random, and it is changing throughout the simulation.
3. The acceleration (both positive and negative) is fixed to 7 km/h/sec for both cars and buses.
4. Buses do not slow down in order to switch the lane. We need the average distances that buses have to go in order to change to the reversible lane and back without forcing them to slow down.

4 Simulation Process

Obviously, we need to implement different algorithms for buses and cars. First of all, the main difference between them is their tactics for changing lanes. In case of cars, they may change lanes in order to outrun other cars. In case of buses they need to get to the leftmost lane as soon as possible and then return back to the rightmost lane for the bus stop. Theoretically, buses should drive on the reversible lane for longer, otherwise it is useless. But our goal is to show the very minimum distance between the bus stops so that introducing such lanes would make sense.

4.1 Changing Lanes

In order to fix the rules of the game, we need to introduce some definitions.

Definition 1. Minimal safe distance *of a car is a distance that the car travels in 0.8 s. This time is fixed intuitively by observing real cars in the city and can be changed by the "safeDistanceInSeconds" variable. So, the higher speed of the car is, the bigger minimal safe distance is.*

For example, for a car with the speed of 60 km/h, the minimal safe distance is 5 m. Note that if one needs to find the minimal safe distance between two cars then these 5 m include the length of the following car.

Definition 2. Emergency situation *is a condition when a car has to change its speed for avoiding collision with another car.*

An algorithm for changing of a car A to a lane L is as follows:

1. L is next to the c
2. The car is in an
   speed to its maxi
3. There is no car or
4. There is no car or
   or that would cau
   its lane to L.

Note that if a car
speed less than its ma
it accelerates. See Sec
An algorithm for l

1. The principles of s
   (an emergency situ
2. In case if the bus
   and changes lane t
3. In case if the bus h
   and changes lane to
4. In case if the bus
   rightmost lane, it sl

4.2 Structures

We have implemented

Lane structure repre

1. *isReverse*: Boolean
2. *top*: Y-coordinate of
3. *middle*: Y-coordinate
4. *width*: The width of
5. *lastCarDistance*: Di
   it to check whether
6. *lastCarSpeed*: The s

Auto structure repre

1. *isABus*: Boolean var
2. *lane*: Returns the nu
   (zero corresponds to t
3. *passedDistance*: Dist
   movement; increasing
   that the X-coordinate
   scale.
4. *length*: The length of
   buses are longer than

1. L is next to the current lane (left or right) and is not reversible.
2. The car is in an emergency situation, or there is no possibility to increase speed to its maximum.
3. There is no car or bus on lane L that is next to car A.
4. There is no car or bus on lane L that would become in an emergency situation or that would cause an emergency situation for car A, in case if A changes its lane to L.

Note that if a car is not in an emergency situation and is driving with the speed less than its maximum speed (each car has its own maximum speed) then it accelerates. See Sect. 4.4 for details.

An algorithm for buses movement is as follows:

1. The principles of safety while changing lanes are exactly the same as for cars (an emergency situation must not occur).
2. In case if the bus has never been to the reversible lane, it keeps movement and changes lane to the left one once it is safe.
3. In case if the bus has already been to the reversible lane, it keeps movement and changes lane to the right one once it is safe.
4. In case if the bus has been to the reversible lane and currently is on the rightmost lane, it slows down until full stop and then disappears.

4.2 Structures

We have implemented two structures in our project - Lane and Auto.

Lane structure represents one lane and has the following properties:

1. *isReverse*: Boolean variable, *True* if the lane is reversible.
2. *top*: Y-coordinate of the top of the lane.
3. *middle*: Y-coordinate of the middle of the lane.
4. *width*: The width of the lane.
5. *lastCarDistance*: Distance traveled so far by the last car on the lane. We use it to check whether there is enough space for a new car to appear.
6. *lastCarSpeed*: The speed of the last car on the lane.

Auto structure represents the class of vehicles (both cars and buses):

1. *isABus*: Boolean variable. *True* if the vehicle is a bus.
2. *lane*: Returns the number of the lane in order, starting from the right one (zero corresponds to the rightmost lane), on which the car currently is.
3. *passedDistance*: Distance traveled so far. This variable is responsible for the movement: increasing this value by 1 causes the vehicle move by 1 m. Note that the X-coordinate of the vehicle depends on *passedDistance* and the scale.
4. *length*: The length of the vehicle. This is random for each vehicle, but the buses are longer than cars.

5. *pictureId*: The number of the picture on screen that corresponds to the vehicle (rectangle index).
6. *hasCarInFront*: Boolean variable, *True* if there is a vehicle in front that causes slowing down.
7. *killed*: Boolean, *True* for all vehicles that are no longer in use.
8. *busReturning*: Boolean, *True* for all buses that have already been to the reversible lane, shows that the bus is returning to the rightmost lane. This variable is responsible for changing lanes.
9. *busTimer*: How long the bus is moving on the current lane. We use this variable to prevent the bus from changing two or more lanes at once. Moreover, it is natural for the drivers to have some interval between changing lanes.

4.3 Appearance of Cars and Buses

Vehicles should appear depending on the given traffic density. The user may fix the densities for the cars and buses separately before simulation starts. We have implemented so called caches for cars and buses that are responsible for the appearance of new vehicles. W.l.o.g., we will only show the algorithm for the appearance of cars. The principle of bus appearance is the same except of the initial speed: the cars start with their maximum speed, but the buses start with 0 km/h, as if they would start from the bus stop.

The buffer for the cars is called *carCache*. For simplicity reasons, we assume that the function (and other functions responsible for movement) is called once a second. In fact, it is called more frequently, depending on *animationSmoothness* variable that can be found in the program.

The appearance of a new car works as follows. If a new car (or several cars) should appear at some point in order to keep the traffic density, it does not appear immediately, but it goes to some buffer (*carCache* is incremented). Once any lane is free for a new vehicle, and the buffer is not empty, the vehicle is built, and *carCache* is decremented.

So, the *carCache* is updated as follows:

$$carCache = carCache + carsAnHour/60/60; \tag{1}$$

I.e., if there should appear *carsAnHour* number of cars per hour, there should appear *carsAnHour*/60 cars per minute, and *carsAnHour*/60/60 cars per second.

Later, if $carsAnHour \geq 1$, we call a function responsible for the creation of a new object. But before creating an object, we should find out which lane is the "most free" (if there is at least one free lane), and we should check whether we can reuse an old object, if it has $killed = True$.

In order to find the best lane for a new vehicle, we *refresh* the distances and speeds of the last cars on each lane. Once refreshed the distances, we search for the "best" lane for a new vehicle to appear by finding a lane with the maximum *lastCarDistance* value. And in case if such a lane exists (it may happen that all the lanes are busy), we create a new car with the following properties:

1. Maximum speed: 50 to 70 (random)
2. Car length: 4 ± 1 m (random, up to the centimeter)
3. Lane number: the above found lane
4. Speed: if the last car on the lane is moving with the speed less than the maximum speed of the new car, we take that last car's speed. Otherwise we take the maximum speed of the new car.

Note that if there is an old object that is not used any more, then the object is reused rather than creating a new one. This is done for performance reasons. Our first version of the program without reusing objects was very slow on some laptops after about 10 s of simulation.

4.4 Vehicle Movement Principles

The following algorithm updates logic for vehicles (again, w.l.o.g., once a second):

1. If the speed of the vehicle is less than the maximum, add 7 km./h.
2. Check if there is another vehicle in front with lower speed.
3. Check the safety of the distance to the vehicle in front (see Definition 1).
4. If the vehicle in front creates an emergency situation (see Definition 2), low down the speed and change to the neighbor lane when possible. Note that the lane change decision for buses is different (see Sect. 4.1).
5. If the bus has already been to the reversible lane, slow down. Once the speed of the bus is equal to zero, disappear.

5 Software Overview and Experiment Results

5.1 Software Overview

The software for our needs was written on C # and it represents several lanes for cars and a reversible lane for buses. W.l.o.g., we have simulated the traffic in only one direction as the opposite movement has no affect on the results.

After launching the program, the user has a chance to change some properties, like density of cars, density of buses, number of lanes (other than the reversible one). Note that the car density is set for all lanes, not for each lane. Hence, if one increases the number of lanes, the density on each lane will decrease.

On the picture below (Fig. 1), we can see yellow rectangles for buses and red rectangles for cars. In order to start simulation, the user has to fill in the fields "Number of cars/hour", "Number of buses/hour", and "Number of lanes". In the bottom of the window, we can see the statistics for 30 buses (513 m in average) .

5.2 Experiment Results

Our experiment was to try the simulation with different traffic density and different number of lanes. As mentioned above, the cars are simulated until they go too far (in this case 3 000 m), and the buses are simulated until they return to

Fig. 1. Screenshot of the program

the rightmost lane and stop. There is one problem left unsolved when two buses are running with the same speed on neighbor lanes, so they can never change to each other's lane. This happens because in our model the buses never slow down to give a way to other vehicles.

On the picture below (Fig. 2) we can see three lines - the above line is the simulation with 4 lanes, the mid line is with 3 lanes, and the bottom line is with 2 lanes. The horizontal axis shows the number of vehicles per hour and the vertical axis shows the minimum bus stop distance.

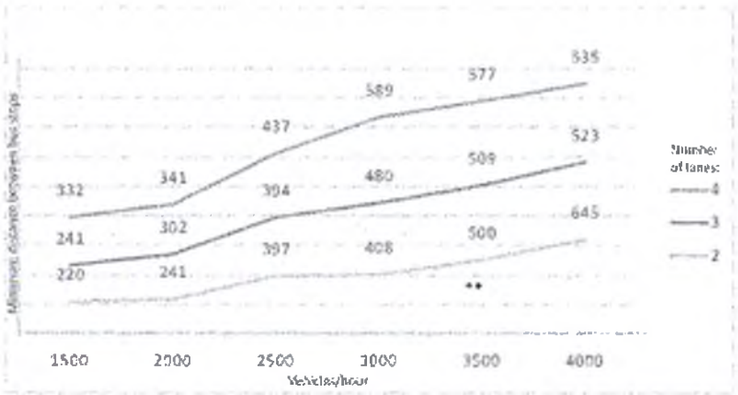

Fig. 2. Experiment results

We have simulated 30 buses for each experiment and calculated the average of all 30 buses. This way we have found the minimum distance between bus stops so that it would make sense to introduce such reversible lanes for buses.

5.3 Programming Details

This sections provides technical details on how this software functions. It could help understanding the source code and modify the software for ones needs.

The program is written in C# using WPF graphical subsystem. WPF uses the advantages of Direct X graphics device interface and has a very comfortable XAML language for designing user interfaces. The solution can be opened in Visual Studio 2012 or later.

Once Main Window is initialized, we initialize the timer that will be enabled once the user sets up all the properties and presses "Start simulation" button. Timer interval depends on the animationSmoothness and the speedFactor variables. Changing speedFactor variable will change the speed of the animation. But changing animationSmoothness will change only frames per second (the speed of the cars will be the same).

When the timer elapses, the following three functions are called: autoLogic, buildBuses, buildCars.

autoLogic function is responsible for all the logic of the program. This function works with each vehicle (current vehicle, midAuto) and does the following.

If the speed of the current vehicle is less than vehicle's maximum speed, we increase the speed by 7 km/h/animationSmoothness. Later we will check the speed and will decrease it if necessary. By now we force each vehicle to drive as fast as possible.

On the next step we check if there is any other vehicle in front of the current vehicle as follows. We take every vehicle ($auto[j]$) that is on the same lane

$$auto[j].lane == midAuto.lane,$$

in front of the current vehicle

$$auto[j].passedDistance > midAuto.passedDistance - midAuto.length,$$

too close to the current vehicle (depending on speed)

$$auto[j].passedDistance - midAuto.passedDistance - midAuto.length < toMetersPerSecond(midAuto.speed) * safeDistanceInSeconds$$

and slow it down to the safe speed:

$$midAuto.speed = toKmPerHour((auto[j].passedDistance - midAuto.passedDistance - midAuto.length)/safeDistanceInSeconds);$$

Here if there is such a car, we set the flag *hasCarInFront* to *True* in order to switch the lane when possible. This is the algorithm of changing lanes for cars.

For buses it is different. If a bus is on the same lane for more than 3 s, it should look for a chance to switch the lane. These three seconds guarantee that the bus will not switch more than one lane at a time. There is a variable *wishedLane* that defines the lane to which the bus should change next. It depends on whether

the bus has already been to the reversible lane or not. Then we check if it is safe to change the lane by calling *laneIsFreeToSwitch* function. On the other hand, if *wishedLane* = −1, it means that the bus has been to the reversible lane and is already on the rightmost lane. Hence, this is time to stop now.

The function *laneIsFreeToSwitch* is also used by other cars in order to check whether it is safe to switch the lane, and if it returns *True*, the vehicle's *lane* variable is changed.

After all these manipulations with lane switching, it is time to change the positions of the vehicles. The position of each vehicle is determined by its lane number and the distance that was passed by the vehicle. The lane number was already changed before, so we now only change the *passedDistance* variable:

$$midAuto.passedDistance = midAuto.passedDistance - toMetersPerSecond(midAuto.speed)/animationSmoothness;$$

In case if the passed distance is too big, we destroy the vehicle. Namely, we set the flag *killed* to *True* and increase the number of vehicles to be recycled. The first version of our program was deleting the object of the vehicle and then another object was created for the new vehicle. That version was very slow even on a modern desktop computer. The new version with reusing objects is fast even on most of the laptops.

Having updated the positions of each vehicle, we call *updateCarPicture* method for redrawing the rectangles.

buildBuses and *buildCars* functions are very similar, so we will concentrate on the *buildCars*. And since Sect. 4.3 describes the function detailed enough, we will only mention some deeper details that are not covered.

As we have mentioned before, this function is called every time timer elapses. And the interval of the timer depends on the number of frames we would like to display per second. Hence the function *buildCars* is called about, say, 50 times per second. Since we do not want to add vehicles so frequently, we count the number of calls (see Eq. 1 in Sect. 4.3):

$$carCache = carCache + carsAnHour/60/60/animationSmoothness;$$

This variable represents a buffer for the cars. If the cache becomes greater than one, this means that it is time to build one car. On the other hand, it may happen that the cache is even greater, this may happen if all the lanes are busy and there is no space for a new car. So the cache remembers how many cars are queued by now.

The animation part is pretty trivial. AnimationGrid represents a Grid control that holds all rectangles in Children collection. Each rectangle represents a vehicle and its width depends on the size of the car and the scale of the program. The scale can be chosen by changing *zoom* variable. AnimationGrid also holds several Line controls that represent the lane dividing lines.

6 Other Work

This experiment is a part of a bigger project named AutoSim for computer simulation of urban traffic. In this project we have developed a software for three-dimensional modeling of urban traffic using openstreetmap.org resources. The aim is to build a professional software with correspondence matrix, complicated calculations of emissions, etc. But as the first stage we aim to make a simple to use software for beginners in transportation, who could download any part of the world from openstreetmap.org and simulate it immediately without any fine-tuning. The problem of openstreetmap.org is that its map represents streets, not lanes. So we had to generate a lane-based map out of a streets map (see fig.). The hard part here is to come up with several lanes from each part of the street and connect them properly on crossroads. Some types of crossroads still presents a challenge for us.

Although AutoSim has quiet sophisticated algorithms for cars movement (changing lanes, accelerating, turning, finding shorter paths, etc.), nice looking cars, streets, houses, bridges, and trees, it still does not have any public transportation system like buses or trams. This has to be done in a later stage, and the results described in this paper will be used for lane changing decisions. We have faced some problems in this smaller project, that have to be solved before implementing buses in AutoSim.

Acknowledgments. We thank S. Badaev and K. Abeshev for the help, and the anonymous reviewers for useful suggestions. This work has been supported by the Ministry of Education of the Republic of Kazakhstan.

References

1. Valencia, A.: Modelo para planificación, operación y diseño físico de corredores de transporte público de superficie. M.Sc. thesis in Transport, Universidad de Chile, Santiago (2008)
2. Valencia, A., Fernandez, R.: Macroscopic simulation approach of public transport on exclusive lanes. In: European Transport Conference (2011)
3. Chen, X., Cai, P., Zhu, L., Yu, L.: Micro-simulation study of the effect of median bus lanes with midblock stop on capacity of urban signalized intersection. In: 2010 13th International IEEE Conference on Intelligent Transportation Systems (ITSC), 19–22 September 2010, pp. 1033–1038 (2010)
4. Ilahi, A., Irawan, M.Z.: A microsimulation model of median busway and ATCS (case study: Transjogja Bus, Yogyakarta, Indonesia). In: Proceedings of the Eastern Asia Society for Transportation Studies, vol. 9 (2013)
5. Fernandez, R.: A new approach to bus stop modelling. Traffic Eng. Control 42(7), 240–246 (2001)
6. Bakibayev, T., Abeshyev, K.: Adaptation of GIS Open Source Maps to Lanes Graph. Karagandy State University, No. 04 (72) (2013). http://www.ksu.kz/files/Vestnik_KSU/Mathematics_4_72_2013.pdf