IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 20, 2020, accepted May 4, 2020, date of publication May 7, 2020, date of current version May 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993200

A Cooperative Algorithm for Lane Sorting of Autonomous Vehicles

AADITYA PRAKASH CHOUHAN, GOURINATH BANDA, AND KANISHKAR JOTHIBASU
Discipline of Computer Science and Engineering, IIT Indore, Indore 453552, India
Corresponding author: Aaditya Prakash Chouhan (phd1501201006@iiti.ac.in)

ABSTRACT In this paper, we propose a generalized algorithm that converts traffic composed of vehicles located randomly in a set of lanes into sorted traffic in which vehicles are moved into the lane corresponding to their destination group. Focus is placed on the cooperative behavior of vehicles. The proposed algorithm architecture divides the entire scenario into various independent sections (called frames) that can be processed in parallel at the same time. Processing each frame involves solving an optimization procedure of a nonlinear programming problem reducible to a linear programming problem. The performance of the proposed algorithm is tested using the Simulation of Urban MObility (SUMO) simulator. Results are obtained and presented for average sorting distance required for sorting all vehicles in the scenario for different traffic settings.

INDEX TERMS Intelligent transportation system, lane sorting, linear programming, lane assignment, vehicle platooning.

I. INTRODUCTION

The autonomous vehicle has become a hot topic among researchers across the globe. There are various benefits of autonomous vehicles over the manually driven vehicles such as they enable children, elderly, disabled people to go from one place to another independently, they don't get tired so there is no problem of drowsiness or lack of attention, they can save the time of the driver which can be used in other tasks than driving, etc. One very significant advantage is that autonomous vehicles have the capability of communicating among themselves and/or with the infrastructure which enables them to behave in a manner that is globally best. This ability to communicate makes any autonomous traffic scenario a task in which the fleet of vehicles acts as a group. In this way, the entire fleet of vehicles works in unison to achieve the objective most efficiently. This level of coordination is not practically achievable by manually driven vehicles. This is the reason that the introduction of autonomous vehicles opens up a huge number of possibilities in devising better algorithms to manage autonomous vehicular traffic in different scenarios.

Despite the popularity of autonomous vehicles and vehicle to vehicle communication, works related to cooperative lane changing are rather scarce in the literature owing to the complex maneuvers involved [1]. So is the case with strategies related to getting all the vehicles into their destination lanes physically with a cooperative approach. Assigning destination lane to vehicles in the scenario is a research problem that goes by the name Lane Assignment [2]–[4]. Lane assignment techniques aim towards maximizing the traffic efficiency on a highway by grouping vehicles into groups that have common interests. This way there will be minimal traffic disruption events such as platoon splits, lane change, etc. Lane assignment can be done for individual vehicles as well as for platoons. The presented work finds application at a stage that comes after lane assignment i.e. all the vehicles in the scenario are aware of their destination lane.

This sounds similar to the platooning of vehicles. This indeed is, except that the inter-spacing of vehicles in a platoon is kept as small as possible. Whereas, in the presented work, the objective is to just bring every vehicle into their destination lane and not to form platoons. Platoons can later be formed by adjusting the inter-spacing of vehicles. This defines where the presented work may find application i.e. after the lane assignment and before the platoon formation. Many platooning related projects have been implemented in the past to establish the benefits of platooning. Some of them are PATH project [5], GCDC project [6] and a more recent one is SARTRE [7]. Readers can refer to the survey given in [8] for various works related to platoon based vehicular systems.

The motivation for the presented work comes from the fact that though these works talk about platoon assignment and platoon formation, they don't specifically talk about how vehicles will coordinate among themselves and physically get into the desired lane. For this reason, the presented work

The associate editor coordinating the review of this manuscript and approving it for publication was Bohui Wang.

is an attempt to fill a void that is present in the Intelligent Transportation Systems (ITS) strategies.

Sorting vehicles in lanes is a multi-vehicle lane change cooperation task. This will include motion planning of multiple vehicles which is inherently a difficult task due to the exponential scaling of computational complexity with the number of vehicles [9]. Authors in [10] analyze cooperative lane change maneuver of vehicles on a three-lane scenario. They use a group of eight vehicles because eight vehicles can cover all possibilities of a three-lane scenario. They proposed a multi-vehicle Minimum Safety Spacing (MSS) model between any two vehicles during the lane change. In [11], the authors formulate the multi-vehicle lane change motion planning task as a centralized optimal control problem. They proposed a Progressively Constrained Dynamic Optimization (PCDO) method for solving the optimal control problem. In [12], authors have presented an algorithm to minimize the disruption of traffic flow and thus maximizing the number of lane changes. They used time slack calculation and the concept of vehicle grouping and used a distributed algorithm to solve the problem. In [13], authors have considered a two-lane scenario with a critical point and presented an algorithm to perform lane change maneuver before the critical point.

Most of these works are close to the presented work but do not share the same objective of sorting vehicles into lanes in minimum time and space. Reference [13] comes close; however, the algorithm presented in this paper is a generalized algorithm in terms of number, width, and availability of lanes. Availability here means that it is not required to have as many number of lanes as there are destination groups. Also, the presented algorithm uses a batch-based approach that enables us to process each vehicle group in parallel. This directly results in the efficiency of computation which is directly reflected in the efficiency of the fleet. Taking inspiration from [14] and [15] in which authors use multi-objective optimization for controlling autonomous vehicles in two ITS scenarios, we formulate the cooperative lane change task as a non-linear optimization problem reducible to a linear-optimization problem.

In this paper, we propose an algorithm to sort vehicles in the lane corresponding to their route or destination. The destination lane of the vehicle can come from a traffic administrator whose job is to control vehicle travel to increase the efficiency of the overall traffic system. We assume in this work that the road considered ends on a four-way intersection and it has dedicated lanes for all three possibilities at the intersections which are turning left, turning right and going straight. To sort vehicles, we first divide the road into smaller sections called frames. Along with dividing the road length, vehicles present in the scenario are also divided into frames. Then for each of these frames, an optimization algorithm is used to get the best possible arrangement of vehicles inside the frame. Every frame is kept independent from other frames which means we can parallelize the processing of each of these frames. We have assumed the road to be straight. The rest of the paper is organized as follows: In section II, we define the scenario under consideration and the underlying architecture. Section III presents the lane sorting algorithm. In this section, we discuss associated sub-routines such as frame creation, Linear Programming (LP) formulation, frame merge, etc. Later in this section, we discuss a possible corner case and how it's been dealt with in sub-section III-F. In section IV, we present details of simulations performed and results obtained from it. Later in section V we conclude the paper.

II. SCENARIO

Consider a long stretch of straight, multi-lane road. Each lane in this road is drivable and is of the same width (not necessary). Long here means that we are not considering distance deadline for our task rather, we are considering that we have a road with some fixed number of lanes for infinite (sufficiently long) distance. The length of road required for any particular traffic setting can then be determined from the results obtained. The road does have a start line though. Incoming vehicles enter this road from the start line with some random velocity bounded by a speed limit of V_{max}. All vehicles are required to have SAE (Society of Automotive Engineers) level-2 autonomy [16] or more (as the vehicle will need to perform autonomous longitudinal and latitudinal movements to change lanes) and are capable of communicating with the Scenario Controller (SC) using wireless communication either directly or via Road Side Units (RSU). Scenario Controller is the central controller which contains the presented algorithm and is responsible for performing all associated tasks such as communication and computation.

As shown in Figure 1, the start line is followed by a road section called Velocity Transition Section (VTS). Incoming vehicles adjust their velocity in this section to achieve the common velocity (V_{common}). The length of VTS can be determined using the Newton's equations of motion and the limits over velocity and acceleration of vehicles in the scenario. For example, for a maximum velocity of 60$kmph$ and minimum acceleration (or deceleration) of $3m/s^2$, the required length of VTS will be 46.3 metres.

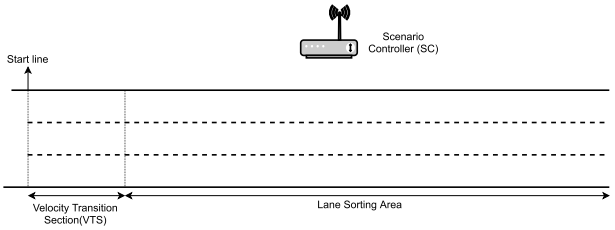


FIGURE 1. The figure shows the Velocity Transition Section (VTS) which is followed by the Lane Sorting Area in the considered scenario.

V_{common} is one of the two parameters that can be varied to suit the incoming traffic density and available road length. The other one is frame length (discussed later). Their values are taken from a lookup table which has previously been generated using the experiments shown in this paper. In case

when sufficiently long stretch of road is available, V_{common} can be set to the V_{max}. Otherwise, V_{common} shall be decreased with a decrease in the length of the available road because reduced velocity will give more time to vehicles for adjusting their position. In some traffic scenarios, different lanes can have different speed limits for example at highways and to satisfy the speed limits of all these lanes, the V_{common} can be chosen from the intersection of the speed limits of all the lanes. For instance, if all the lanes have different upper limits on vehicle velocities, the V_{common} will be bounded by the minimum of these upper limits. In this manuscript we have considered a scenario that has a lane sorting area prior to an intersection and this lane sorting area has no speed restrictions on individual lanes. By considering such a scenario, we demonstrate a possible architecture that can be realized prior to any scenario that is benefited by sorted traffic, for example, a Tollgate or an intersection, etc. In the experimentation, for demonstration purpose, we have performed simulations for V_{common} equals to 5m/s, 10m/s and 15m/s. These velocities in kilometers per hour correspond to 18, 36 and 54 kmph, which are very common for any urban traffic scenario. We also perform the same set of experiments for velocities 20m/s, 25m/s, and 30 m/s. These velocities in kilometers per hour correspond to 72, 90, and 108 kmph. These velocities cover the range of velocities that is common for vehicles on a highway.

All incoming vehicles perform wireless communication with SC and pass their information such as their length, width, velocity, route, source (incoming) lane (l_s), etc. For simplicity, we have assumed the length of every vehicle to be the same. SC in return passes V_{common} and the lane in which vehicle has to shift (destination lane, l_d). The destination lane of a vehicle may or may not be the same as its initial or incoming lane. Deciding the destination lane of incoming vehicles is a research topic in itself known in the research community by the name Lane Assignment. Lane assignment can have different approaches such as grouping by destination, dynamic grouping, grouping by size [17] etc. Since in this work, the grouping of vehicles is done based on the destination direction of each vehicle at the intersection, we can say that SC is using the grouping by destination strategy of lane assignment in the background. Although the considered scenario is an example of grouping by destination, the presented algorithm may well be used along with any other lane assignment technique. Vehicles start transiting to V_{common} as soon as they enter VTS. Lane sorting starts after vehicles have left VTS and are now in the sorting area.

Although the presented work is generalized in sense of the number of lanes i.e. there can be any number of lanes on the road, we have used for explanation purposes, a scenario with a road that has three lanes wherever required. We have abstracted away from the imperfections in communication to keep the objective of this manuscript clear and that is to propose a lane sorting algorithm hence the communication is assumed to be flawless.

III. ALGORITHM

Sorting area and vehicles inside it are divided into various sections known as Frames. All frames are of the same size when created and move with a velocity same as the vehicles contained in it i.e. V_{common}. As a result, the frame-vehicle association is always preserved unless the frame is to be merged with some other frame. Frames are created at the start of the sorting area. A frame after being created can merge with another frame if required and frames are destroyed when all their contained vehicles get past the lane sort area. In between their creation and destruction, they move with V_{common} velocity. All these frames are nonoverlapping and every vehicle in the scenario should be associated with a frame. Frames can have a gap in between them i.e. they are not necessarily required to be back-to-back as shown in Figure 2.

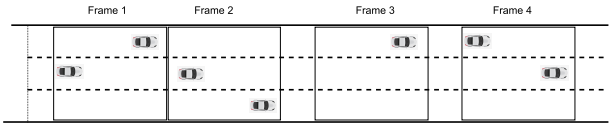


FIGURE 2. The figure shows a possible distribution of frames on a stretch of road.

The geometrical center of the vehicle is taken as the position of a vehicle. As stated earlier, all vehicles and frames move with the same longitudinal velocity, when seen from a perspective of the moving frame i.e. when we see a frame from a reference moving along with the frame, then it will look stationary with fixed relative positioning of vehicles inside it. From this perspective, a lane change maneuver will look like the vehicle is moving horizontally in between the lanes given that the vehicle maintains this longitudinal velocity while changing the lane as well.

In continuation of the above observation, we can say that a vehicle can perform a safe lane change maneuver if it can obtain a horizontal space that is nonoverlapping with other vehicles. We call this horizontal space as *Channel*.

This reduces our task at hand at this stage to get independent channels for vehicles wanting to change lanes to their destination lanes. We make use of Linear Programming over the positioning of vehicles in the frame to achieve this task. But first, we discuss how frames are created in the scenario.

A. FRAME CREATION

To explain how frames are created, let us consider that initially there are no vehicles in the scenario. As vehicles start coming in the scenario, they are first added to a temporary list. A check is made at every step to get the position of the first vehicle in the scenario. As long as the position of the first vehicle is less than the frame length, all incoming vehicles are added to the temporary list. As soon as the first vehicle is past the frame length, a frame is created and all the vehicles in the temporary list are assigned to the frame just created and the temporary list is cleared. Once a frame is created, it starts moving with the V_{common} velocity. As a result, the relative positioning of vehicles inside the frame shall be fixed unless vehicles are in a maneuver to change lane or getting into the

assigned position in the frame. After vehicles complete the maneuver, they will again attain the V_{common} velocity.

As vehicles don't cross their frame and frames don't overlap, we can say that through frames, the complete scenario has been divided into various independent sections that can be processed in parallel. Processing each frame involves formulating a Linear Programming (LP) problem that will return vehicle positions as a result. In case the solution of the linear programming formulation is not feasible, the frame is merged with another frame. We will next discuss the LP formulation.

B. LINEAR PROGRAMMING FORMULATION

As discussed earlier, we breakdown the task of sorting vehicles on the entire road into sorting vehicles in smaller road sections called frames. We will now discuss the formulation of the linear programming problem which gives channel positions as its solution. We first define some sets of vehicles present in the frame under consideration. The first set is named *Vehicles*, it is the set of all the vehicles inside the frame. There are three subsets of the *Vehicles* set. These subsets are *SortedVehicles* set, *UnsortedVehicles* set and *SupportingVehicles* set. As their names suggest, *SortedVehicles* set contains those vehicles that are already in their destination lane, *UnsortedVehicles* set contains vehicles that are not in their destination lane and will demand a channel for lane-change maneuver, and *SupportingVehicle* set contains those vehicles which are not yet in their destination lane but are behaving as sorted vehicles temporarily. Unsorted vehicles always demand a channel whereas sorted and supporting vehicles always clear the space for creating channels. We will discuss later in this section how and when an unsorted vehicle is made supportive.

Let us consider the frame shown in Figure 3. The frame starts at f_{start} and end at f_{end}. Let the vehicle i with initial position (x_i, y_i) is being assigned a channel centered at y_{Ci}. Vehicle i will need to shift by a distance of $\Delta y_i = (y_i - y_{Ci})$ to get into its corresponding channel. To keep this shift minimum, we would want that the channel should be as close

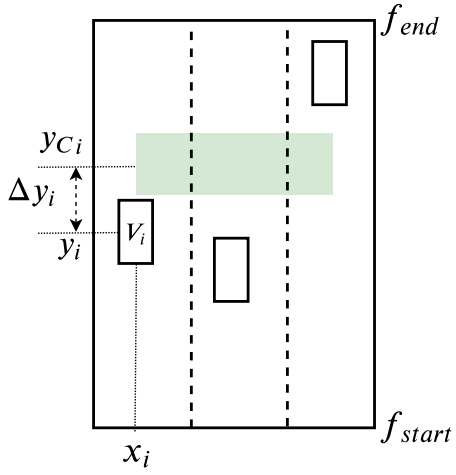


FIGURE 3. This figure shows a frame along with notations used in equations.

as possible to the vehicles' initial position. Along with this, we would also like to minimize the shift required in the position of the sorted and unsorted vehicles as well. This requirement gives us the following objective to our linear programming problem.

$$Objective : Min.\ \ \Sigma|\Delta y_i|\ \ \forall i \in Vehicles \tag{1}$$

We will now discuss associated constraints:

1) **Vehicles are always separated longitudinally by at least a safe distance.** We call this safe distance as Safety Gap (SG). This means channels, sorted vehicles and supporting vehicles should always be positioned such that vehicles will always have a longitudinal physical separation of at least SG in between them. The associated constraint will look like follows.

$$\begin{aligned} &|(y_i + \Delta y_i) - (y_j + \Delta y_j)| \geq v\_len + SG \\ &\quad \forall i, j \in Vehicles,\ i \neq j \\ &|l_{sj} \in (l_{si}, \ldots l_{di})\ or\ l_{dj} \in (l_{si}, \ldots l_{di}) \end{aligned} \tag{2}$$

Here l_s and l_d represent the source lane and the destination lane respectively. The condition $l_{sj} \in (l_{si}, \ldots l_{di})\ or\ l_{dj} \in (l_{si}, \ldots l_{di})$ select vehicles that can have conflict while changing lanes. For vehicles that can not possibly conflict with each other while changing lanes, this constraint is not applicable. v_len represents length of the vehicle.

2) **Vehicles associated with a frame should lie completely inside the frame.** This constraint makes all the frames independent because no vehicle is allowed to cross the frame boundary. The associated constraints will look like.

$$y_i + \Delta y_i \geq f_{start} + \frac{v\_len}{2} + \frac{SG}{2} \tag{3}$$

$$y_i + \Delta y_i \leq f_{end} - \frac{v\_len}{2} - \frac{SG}{2} \tag{4}$$

$\frac{SG}{2}$ is added in both constraints to prevent double safety spacing between vehicles near the boundary of two frames.

3) **Actual order of vehicles in lanes should be preserved.** This constraint keeps vehicles in the same lane in their actual ordering. Since the linear programming solution will contain a change in position in the geometrical center of the vehicle, this constraint will put limits on the values of Δy such that actual order of vehicles is maintained.

$$\begin{aligned} &(y_i + \Delta y_i) - (y_j + \Delta y_j) >= v\_len + SG \\ &\quad \forall i, j \in Vehicles \mid i \neq j, x_i = x_j, y_i > y_j \end{aligned} \tag{5}$$

$$\begin{aligned} &(y_j + \Delta y_j) - (y_i + \Delta y_i) >= v\_len + SG \\ &\quad \forall i, j \in Vehicles \mid i \neq j, x_i = x_j, y_j > y_i \end{aligned} \tag{6}$$

As we can see that objective and constraint given in equations 1 and 2 respectively are not in a form appropriate for linear programming formulation because they contain

absolute value functions. To resolve this we can either use a quadratic programming solver or we could transform these equations into a form acceptable by linear programming solvers. We choose the second option as using a quadratic programming solver is more resource-intensive than a linear one.

C. TRANSFORMING TO LINEAR PROGRAMMING

To transform these expressions into linear programming solver acceptable form, we need to add additional variables and expressions. We do this one by one starting with the objective itself. To transform the objective, we add an extra variable named $\overline{Obj}$ and a variable array $\overline{\Delta y}$ in which all the elements have a lower bound of zero, along with following additional constraints.

$$\overline{\Delta y_i} >= \Delta y_i \quad \forall i \in Vehicles \tag{7}$$

$$\overline{\Delta y_i} >= -\Delta y_i \quad \forall i \in Vehicles \tag{8}$$

$$\Sigma\overline{\Delta y_i} <= \overline{Obj} \quad \forall i \in UnsortedVehicles \tag{9}$$

$$\Sigma\overline{-\Delta y_i} <= \overline{Obj} \quad \forall i \in UnsortedVehicles \tag{10}$$

And our new objective will become

$$Min.\ \overline{Obj} \tag{11}$$

To transform the constraint given in equation 2 into an appropriate form, we need to add one binary variable (b) for each combination of i and j and a variable with a constant value (M) large enough to satisfy constraints. Every instance of equation 2 will be replaced by following two constraints.

$$(y_i+\Delta y_i)-(y_j+\Delta y_j)+M*b_{ij} \geq v\_len+SG \tag{12}$$

$$(y_j+\Delta y_j)-(y_i+\Delta y_i)+M*(1-b_{ij}) \geq v\_len+SG \tag{13}$$

The value of M should be large enough to satisfy the above equations. We take it to be double the frame length, any larger value will also work fine. Now, this linear programming is solved for a solution. In the first iteration, all the unsorted vehicles inside the frame demand for a channel. If it is feasible for every unsorted vehicle to get a channel in the first iteration itself, the solution is obtained. However, if it is not possible for every vehicle to get a channel, the LP solver will report no feasible solution. This is where we are required to shift vehicles from the *UnsortedVehicles* list to the *SupportingVehicles* list. This could be done iteratively by shifting one vehicle to the *SupportingVehicles* list at each iteration until the solution is obtained.

After obtaining a solution, vehicles are required to move into their assigned positions. Unsorted vehicles have to move into their assigned channels; whereas, sorted and supporting vehicles move to the positions obtained by solving LP. Every LP solution is followed by a movement step in which all vehicles shift to assigned positions and then lane change maneuvers are performed. After the movement step is over, the next sorting step is started. This sequence of sorting and moving is repeated until all vehicles in the frame are sorted. When traffic is high or in the case when the number of vehicles destined to one lane is more than the lane capacity, it is not possible to get all vehicles in the frame in their respective lanes. In such a case, the frame is merged with the frame upstream. We can see an example in Figure 4. In this figure, the vehicles are shown as the smallest rectangle that will fit the vehicle when seen from above. Color of the rectangle mean following:

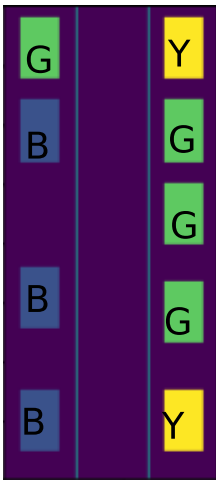


FIGURE 4. This figure shows a case in which frame merge will be required as number of vehicles destined to travel in the right most lane is more than the lane capacity.

- **Blue**: Vehicles destined to go to the left most lane
- **Cyan**: Vehicles destined to go to the middle lane
- **Green**: Vehicles destined to go to the right most lane
- **Yellow**: Vehicles already in their destined lane.

When such a condition occurs, all vehicles in the *UnsortedVehicles* set are shifted to *SupportingVehicles* set. Thus to detect this case, we check for the number of elements in *UnsortedVehicles* set and *SupportingVehicles* set. If the number of elements in *UnsortedVehicles* set is zero and the number of elements in *SupportingVehicles* set is greater than zero, the frame has to be merged with the frame upstream.

D. CHOOSING VEHICLES FOR SUPPORTING VEHICLES LIST

As mentioned earlier, when there is no feasible solution to the formulated LP problem, vehicles are shifted from *UnsortedVehicles* set to *SupportingVehicles* set. This shifting, however, is not done arbitrarily rather, we need to decide first what number of vehicles are to be shifted and in what order. If the number of vehicles to be shifted is not known beforehand, one vehicle would be shifted in every run followed by formulation and solving of the entire LP problem. And this will be repeated multiple times until required number of vehicles are not shifted. Thus not knowing the required number of vehicles to be shifted would result in an unnecessary delay in computation. To prevent this, we first find out the required number of vehicles to be shifted into the *SupportingVehicles* set.

We now explain analytically the process of finding the required number of vehicles to be shifted into the *SupportingVehicles* set. Let the maximum number of vehicles that can be accommodated in one lane is *lane_max* and there are

n number of lanes in the scenario. Now for each lane, we need to find the number of vehicles each lane has to provide occupancy for in order to perform lane change of vehicles. Each lane will have to provide occupancy for vehicles of three classes which are:

1) Vehicles already present in that lane,
2) Vehicles in other lane and destined to that lane and
3) Vehicles in other lane that has to cross that lane in order to reach its' destination lane.

For each lane in the scenario, we then add the number of vehicle occupancies that have to be provided corresponding to these three classes; let this sum be called *num_occupancy*. Number of vehicles that have to be shifted to the *SupportingVehicles* list (N) is then obtained by adding the number of vehicle occupancy each lane has to provide in excess to *lane_max* i.e.

$$N = \Sigma(num\_occupancy_i - lane\_max)$$
$$for\ i = 1 \ldots n\ ;\quad num\_occupancy_i > lane\_max \quad (14)$$

N is the number of vehicles that have to be made supportive to obtain a solution. In case this number is equal to the number of unsorted vehicles, at any stage, then there will be no further feasible solution and the frame will have to be merged to proceed with lane sorting.

Table 1 gives the occupancy requirements from lanes in the example scenario shown in Figure 5. The number of vehicles to be shifted is the addition of difference with the maximum occupancy (which is 5 for the example case; with the length of each vehicle = 3 meters, SG = 2 meters and frame length of 25 meters) for lanes having occupancy greater than maximum occupancy. Hence, we need to shift 1 vehicle to the *SupportingVehicles* set.

TABLE 1. The table gives the occupancy requirements from lanes in the example scenario shown in Figure 5.

	Class 1 (Already present)	Class 2 (Destined)	Class 3 (Passing)	Total
Left lane	3	0	0	3
Middle lane	4	2	0	6
Right lane	1	1	0	2

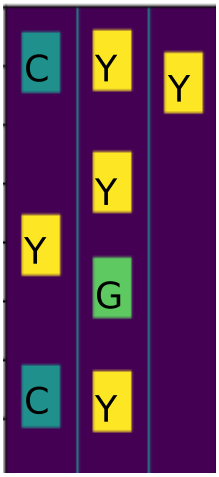


FIGURE 5. Example scenario to find number and order of vehicles shifted into *SupportingVehicles* set.

To get the number of vehicles to be shifted into the *SupportingVehicles* set, an efficient method will be to iterate through all the vehicles and increment the lane counter for each lane in between the current lane and the destination lane of the vehicle. Where lane counter is an integer array with an equal number of elements as there are the number of lanes and initialized with zero in all the elements. Python pseudo-code is given below.

```
N = 0 # Number of vehicles to be shifted
lane_counter = [0 for i in range(
    num_lanes)]
for vehicle in vehicle_list:
  l_s = vehicle.source_lane
  l_d = vehicle.dest_lane
  if l_s < l_d:
    for i in range(l_s, l_d):
      lane_counter[i] += 1
    lane_counter[l_d] += 1

  elif: l_s > l_d:
    for i in range(l_d, l_s):
      lane_counter[i] += 1
    lane_counter[l_s] += 1
  else:
    lane_counter[l_s] += 1


for i in range(num_lanes):
  if lane_counter[i] > max_occupancy:
    N += (lane_counter[i] -
      max_occupancy)
```

Even after getting the number of vehicles that have to be made supportive, the decision of choosing which vehicle to be shifted to the supporting vehicles list is also important. Choosing wrong vehicles can result in a greater number of runs for sorting vehicles or it can also result in no solution at all. For instance, consider the frame shown in Figure 5. As we can see in this figure, all three unsorted vehicles (one green and two cyan) can not move into their respective lanes simultaneously because the middle lane has space to accommodate only one more vehicle. Hence one vehicle has to be made supportive. Suppose that green vehicle is made supportive. Now, since the green vehicle is required to behave as a sorted vehicle temporarily, it will not shift lane but will make space for one of the cyan vehicles and that cyan vehicle will then move into the middle lane. Now, in the next sorting step, the green vehicle will again be eligible for attempting to change the lane along with the leftover cyan vehicle. Now suppose the green vehicle is again made supportive, then in that case, there will be no space left in the center lane to accommodate the leftover cyan vehicle. This will end up to be a no solution case. On the other hand, if the leftover cyan vehicle is chosen as a supportive vehicle instead of green,

the green vehicle would have got a channel to shift to the right lane and then finally in the next sort step, the remaining cyan vehicle will get chance to change lane. This example shows that the selection of vehicles to be made supportive is also crucial.

To find which vehicles are appropriate to be made supportive we extend the method used to calculate the number of vehicles to be made supportive. We discussed that there are three classes of occupancy that a lane has to provide to allow lane changes to take place. Vehicles of the first class are already present in the lane and if any of those vehicles are made supportive then it will not reduce space requirements from the lane. On the other hand, if any of the vehicles from the second or third class are made supportive then that vehicle will not demand channel and hence will reduce space requirements from that lane. This is the principle behind selecting supporting vehicles.

For each lane, a list of candidate vehicles is generated by combining vehicles of second and third class as discussed above. Then from this list of candidate vehicles, as many numbers of vehicles are made supportive as there is the requirement of space in excess to the maximum occupancy limit from that lane. Let us take the example given in Figure 5. Since only the middle lane has occupancy requirement in excess of maximum lane occupancy of 5, we create a candidate vehicle list for only that lane. This candidate list will contain the two cyan vehicles out of which any one can be chosen as the supportive vehicle.

Although the choice of supportive vehicles from the candidate vehicles can be arbitrary, an attempt should be made to select that vehicle first which is present in multiple candidate lists. Such vehicles will reduce the number of vehicles to be made supportive as they reduce the occupancy requirement of multiple lanes.

When all the discussed steps are followed, we will know beforehand whether the solution of the LP formulation will be feasible or not. This will prevent unnecessary runs of the solver and thus will save time. When a solution does exist and is returned by the solver, then vehicles are required to move to the assigned positions. Supporting and Sorted vehicles will move to clear channels and Unsorted vehicles will move into their respective channel and later perform the lane change maneuver.

These steps are repeated iteratively until all the vehicles are sorted in the frame. At the start of each iteration, *Vehicles* set is divided into the three sub-sets, this means, supporting vehicles in one iteration will take part as unsorted vehicles in the next iteration initially and later the division will be done based on the requirements. Algorithm 1 depicts the complete work flow that goes in sorting one frame.

Algorithm 1 Sorting Vehicles in a Frame

```
Input: Frame object
Output: Sorted Frame
all_vehicles_sorted = False;
while !all_vehicles_sorted do
    Get SortedVehicles set;
    Get UnsortedVehicles set;
    Find N;
    if N < len(UnsortedVehicles) then
        Shift N vehicles to SupportingVehicles set;
        Solve optimization (LP) problem;
        if N == 0 then
            if new_frame == True then
                Rearrange vehicles in frame;
            end
            all_vehicles_sorted = True
        end
        Move vehicles;
    end
    else
        if new_frame == True then
            Rearrange vehicles in frame;
        end
        Merge frames;
    end
end
```

E. FRAME MERGE

As we have discussed above, when the distribution of vehicles in the frame is such that either there are a larger number of vehicles to be shifted into a lane than it's occupancy limit or all the unsorted vehicles are required to be made supportive, then the solution of the LP problem is not feasible. In such a case, the frame is to be merged with an another frame. In our implementation, we have merged such frames with the frame upstream to that frame. This is because, the frame upstream have spent greater time in the scenario and is more likely to be sorted. We have also put the restriction that the frame upstream should be sorted before it merges. The current frame will wait for the upstream frame to be sorted before it can merge and in the mean-time will just continue traveling with V_{common} velocity. When two frames merge, the f_{start} of the following frame is the start position of the new frame and the f_{end} of the leading frame is the end position of the new frame. Since a sorted frame might help an unsorted frame in getting sorted, frames are never destroyed while they are inside the lane sort area.

In case the first frame in the scenario can't find a solution, then it will just increase its frame length to 1.5 times its current length.

F. REARRANGE VEHICLES IN FRAME

As can be seen in the Algorithm 1, when N is greater than or equal to the number of unsorted vehicles in the frame, we will be doing a frame merge operation. This is because the LP solver will not be able to give any solution for the positions of the vehicles in the frame and also until the frame ahead is not sorted, merge operation will be in a pending state. Now suppose, the frame that needs to merge has just

been created and the last vehicle's front center is just inside the frame and the rest of the body of the vehicle is outside the frame as shown in the Figure 6. The last vehicle is in a dangerous position and will remain in that position because it's repositioning is only governed by the solution of the LP solver. Now since the solution of the LP solver is not feasible, all the vehicles in this frame will be traveling as they are in the frame. This creates a vulnerable scenario in which if a new frame is created just after this frame then the last vehicle will be inside the following frame's boundary and can cause a collision.

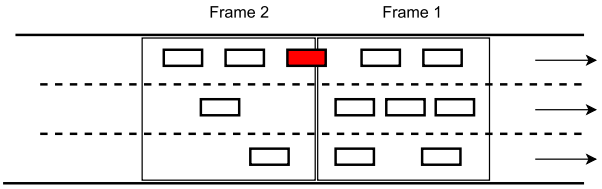


FIGURE 6. The figure shows a case where rearrangement of vehicles will be required to prevent collision. The red vehicle is associated with Frame 1 but protrudes into Frame 2 as it is not moved by the lane sorting LP formulation.

To prevent such a scenario, we have a routine that rearranges vehicles in a frame when the frame is created and it's N is greater than or equal to the number of unsorted vehicles in it or when all the vehicles in the frame are already sorted. The rearrange routine is placed as shown in Algorithm 1 (lines 11 and 19). It contains another LP formulation that only forces vehicles to be inside the frame. The constraints are same as constraints given in equations 3 and 4. And the objective is to minimize the overall vehicle movement. The variable *new_frame* is initialized with True when a frame is created and is set to False after it is processed by either the sort routine or the rearrange routine for the first time. Now a question might arise that what if even rearranging of vehicles inside a frame is not possible? This will only happen when the incoming vehicles are very tightly packed and the separation between vehicles is less than the safety gap (SG) that have been considered in the LP formulation. This puts a limit on the value of the safety gap we have used in our work. We can thus make the following statement:

The value of the safety gap (SG) that can be used in the implementation is limited by the spacing of vehicles in the incoming traffic. Or we can also say that the minimum value of the SG that we can use during lane sorting is the average spacing of vehicles in the incoming traffic.

Please note that the average spacing of vehicles in every frame in the incoming traffic should also be greater than or equal to the SG considered.

G. ADJUSTING V_{common} AND FRAME LENGTH

V_{common} and frame length are considered to be constant in the discussion so far. However, these parameters can be changed to accommodate variation in the traffic. Decreasing V_{common} will give more time to the algorithm to sort vehicles in the scenario thus enabling the algorithm to handle heavier traffic. On the other hand, by varying the frame length, we can adjust the batch size of vehicles that are processed at a time. The experimentation shown in the next section can be repeated for the given traffic density to decide the values of V_{common} and frame length that will result in a sorting distance less than the available road length.

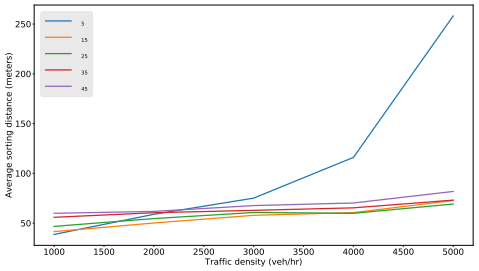
IV. SIMULATION

The algorithm presented considers a general road with n number of lanes. Every vehicle has a source lane and a destination lane which may or may not be the same. Vehicles are spawned into the system on a random lane. However, in the simulation we consider a scenario consisting of three lanes. The road is assumed to be destined towards an intersection such that the vehicles in the scenario are destined to go either right, left or straight. Thus the objective of the algorithm is to sort vehicles into the lane corresponding to the destination direction of vehicles i.e. vehicles destined to go right should be brought to the right lane, vehicles destined to go left should be brought to the left lane and the vehicles destined to go straight should be brought on the middle lane. Vehicles may arrive on any lane which may or may not be the same as their destination lane.

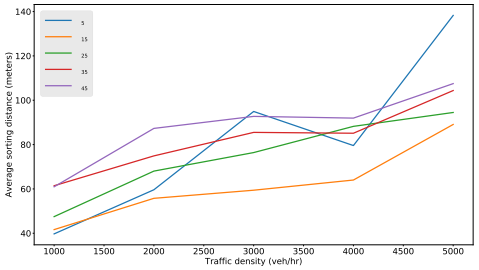
The algorithm may well be applied to a scenario in which the number of destination direction is not equal to the number of lanes on the road. In such a case, one or more lanes will carry the traffic of vehicles corresponding to multiple destinations. For instance, a road with two lanes, destined towards an intersection, can have one exclusive lane for the left turn and a shared lane for right turning and straight going vehicles.

For simulation, we have used the Simulation for Urban MObility (SUMO) simulator which is an open-source, microscopic road traffic simulator. SUMO accepts the scenario, network, and route information in.xml format. The control logic is implemented inside a Python script which is interfaced with the SUMO simulator using the value retrieval and value setting functions given in the Traci library provided by SUMO. We have used the Mixed Integer Linear Programming (MIP) solver for formulating and solving linear programming problems. It is included in the Python package index and requires Python 3.5 or newer [18]. In the simulation, we have kept the length of the road to be sufficiently large and then recorded the average distance required by vehicles to get sorted. The experimentation related source codes and system setup details have been uploaded on to GitHub and accessible via the link: https://github.com/aadiprakash163/lane_sorting For simulations, we have generated random route files with 50 vehicles corresponding to traffic densities of 1000, 2000, 3000, 4000 and 5000 vehicles per hour. The frame length is varied in the range of 5 to 45 meters in steps of 10. Variation of frame length corresponds to varying occupancy limit from 1 vehicle to 9 vehicles in the frame as with the values of vehicle length (3 m) and safety gap (2 m) considered, one vehicle will need a space of 5 meters.

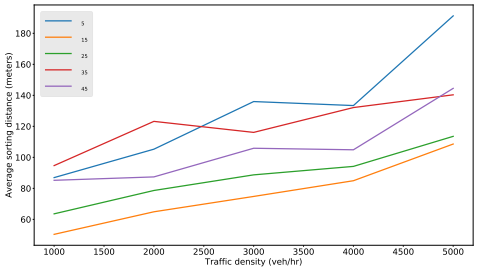
Variation of the average distance to sort with respect to traffic density and frame length can be seen in Figure 7. In these figures, on the abscissa, we have varying traffic density and on the ordinate, we have average sorting distance. Each line in plots corresponds to different values of frame length.



(a) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 5m/s$



(b) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 10m/s$



(c) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 15m/s$

FIGURE 7. Average sort time vs. traffic density for different values of V_{common}.

All these graphs show an increasing trend of average sorting distance with respect to traffic density for each frame length. However, the frame length of 5 meters has a profound behavior and this is because this length corresponds to the occupancy of 1 vehicle in each lane. For very low traffic in which vehicles are very sparse, this frame length is most suitable because vehicles will rarely have conflicts and also will be able to perform lane change as soon as they enter the scenario. Thus this frame length will result in minimum sorting distance. On the other hand, at very high traffic, frame length of one will result in an excessive number of frames which in turn will result in an increased number of frame merge requests. Since frames can't be merged unless the frame upstream is not completely sorted, this will result in additional distance traveled while waiting for upstream frame

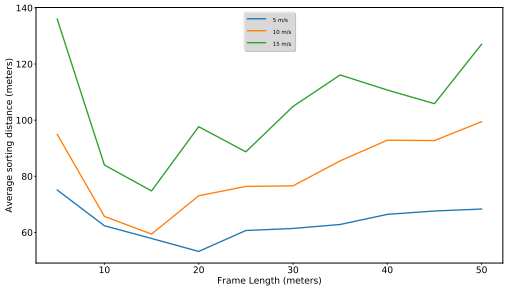
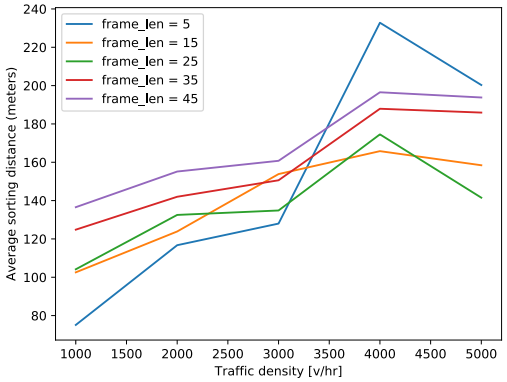
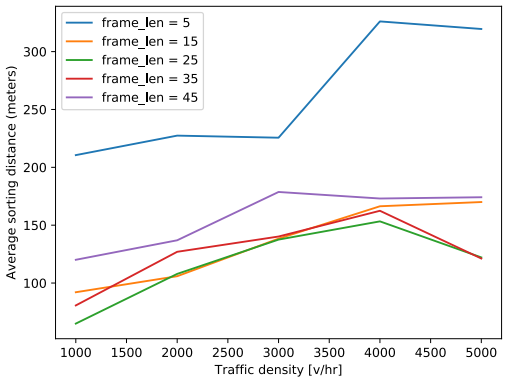


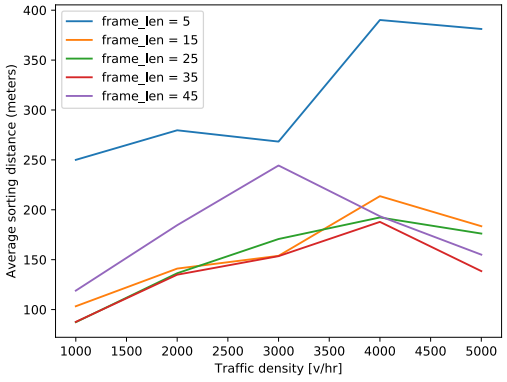
FIGURE 8. Variation of average sorting distance with respect to varying frame length for traffic density = 3000 veh/hr.



(a) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 20m/s$



(b) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 25m/s$



(c) Variation of average sorting distance with respect to varying traffic density for $V_{common} = 30m/s$

FIGURE 9. Average sort time vs. traffic density for different values of V_{common} in highway scenario.

to get sorted. Thus frame length of 5 meters will result in maximum sorting distance at very high traffic. Following thumb-rule can be given by observing:

Appropriate frame length for any traffic density will be the one that will result in a good balance between the number of vehicles in the frame and free space present inside the frame which vehicles can use to get sorted.

For traffic density other than 1000 vehicles per hour, this balance is found at a frame length greater than 5 meters.

Figure 8 shows a different representation of results for a traffic density of 3000 vehicles per hour. As we can see all three lines corresponding to the three velocities, have minimum sort distance at an intermediate value of frame length and any other frame length whether greater or smaller results in higher sort distance.

As the presented algorithm have potential applications in a highway scenario as well, we extend the above simulations to velocities common in such a scenario. We perform the same set of experiments on velocities 20m/s, 25m/s, and 30m/s. The maximum velocity corresponds to 108 kilometers per hour, which is towards the higher end of the speed limits as regulated by authorities in most of the countries [19]–[23]. The performance of the lane sorting algorithm for the set of higher velocities is given in Figure 9. The most appropriate choice of V_{common} for any given traffic density and length of road available can be determined using the discussed experimentation. Using these experiments, on the range of the values of V_{common}, frame length, and traffic density and the average length required by vehicles to sort themselves is recorded in a look up table. This look up table is then referred in the future to decide the frame length and V_{common} that has to be kept so that vehicles incoming with the given traffic density sort themselves within the available road length. This look up table based approach is very efficient as there is no need of additional processing in addition to traffic monitoring, hence less computation and infrastructure is required.

V. CONCLUSION

In this paper, we approached the unexplored problem of cooperative lane sorting among autonomous vehicles. The lane sorting problem is presented as a group task that all the vehicles in the scenario perform cooperatively. The presented algorithm first simplifies the problem by restricting each vehicle to maintain a fixed velocity that is common for all the vehicles in the scenario. This allows us to formulate a non-linear programming problem reducible to linear programming. Application of this linear programming formulation along with a well-structured skeleton of the algorithm that uses a frame based approach on straight road results in a collision-free lane sorting of vehicles. The algorithm presented except assuming a straight road does not assume any particular scenario rather is generalized in terms of the number of lanes, incoming traffic density, length of vehicles and width of lanes. We do specify the safety-gap used in this paper and discuss how it depends on the inter-spacing of vehicles in the incoming traffic. The implementation of the proposed algorithm is performed in the SUMO simulator and results are presented for different values of traffic density, velocity, and frame length. The experimentation presented can be used to decide the frame length and V_{common} for a given traffic density and available road length.

REFERENCES

[1] Y. Luo, Y. Xiang, K. Cao, and K. Li, "A dynamic automated lane change maneuver based on vehicle-to-vehicle communication," *Transp. Res. C, Emerg. Technol.*, vol. 62, pp. 87–102, Jan. 2016.
[2] R. W. Hall, "Longitudinal and lateral throughput on an idealized highway," *Transp. Sci.*, vol. 29, no. 2, pp. 118–127, May 1995.
[3] T.-S. Dao, C. M. Clark, and J. P. Huissoon, "Distributed platoon assignment and lane selection for traffic flow optimization," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2008, pp. 739–744.
[4] T.-S. Dao, C. M. Clark, and J. P. Huissoon, "Optimized lane assignment using inter-vehicle communication," in *Proc. IEEE Intell. Vehicles Symp.*, Jun. 2007, pp. 1217–1222.
[5] *PATH*. Accessed: Apr. 19, 2020. [Online]. Available: http://www.path.berkeley.edu/
[6] M. Lauer, "Grand cooperative driving challenge 2011 [ITS Events]," *IEEE Intell. Transp. Syst. Mag.*, vol. 3, no. 3, pp. 38–40, Oct. 2011.
[7] T. Robinson, E. Chan, and E. Coelingh, "Operating platoons on public motorways: An introduction to the sartre platooning programme," in *Proc. 17th World Congr. Intell. Transp. Syst.*, vol. 1. 2010, p. 12.
[8] D. Jia, K. Lu, J. Wang, X. Zhang, and X. Shen, "A survey on platoon-based vehicular cyber-physical systems," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 263–284, 1st Quart., 2016.
[9] M. Chen, J. F. Fisac, S. Sastry, and C. J. Tomlin, "Safe sequential path planning of multi-vehicle systems via double-obstacle Hamilton-Jacobi-Isaacs variational inequality," in *Proc. Eur. Control Conf. (ECC)*, Jul. 2015, pp. 3304–3309.
[10] Y. Luo, G. Yang, M. Xu, Z. Qin, and K. Li, "Cooperative lane-change maneuver for multiple automated vehicles on a highway," *Automot. Innov.*, vol. 2, no. 3, pp. 157–168, Sep. 2019.
[11] B. Li, Y. Zhang, Y. Ge, Z. Shao, and P. Li, "Optimal control-based online motion planning for cooperative lane changes of connected and automated vehicles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 3689–3694.
[12] D. Desiraju, T. Chantem, and K. Heaslip, "Minimizing the disruption of traffic flow of automated vehicles during lane changes," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 3, pp. 1249–1258, Jun. 2015.
[13] M. Atagoziyev and W. Klaus, "Lane change scheduling for autonomous vehicles," *IFAC-PapersOnLine*, vol. 49, no. 3, pp. 61–66, 2016.
[14] X. Li and J.-Q. Sun, "Signal multiobjective optimization for urban traffic network," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 11, pp. 3529–3537, Nov. 2018.
[15] X. Li and J.-Q. Sun, "Defensive driving strategy and control for autonomous ground vehicle in mixed traffic," in *Proc. Numer. Evol. Optim. Workshop (NEO)*. Tlalnepantla de Baz, Mexico: Springer, Sep. 2016, pp. 3–44.
[16] *SAE Autonomy Levels for Self-Driving Cars*. Accessed: Apr. 19, 2020. [Online]. Available: https://cdn.oemoffhighway.com/files/base/acbm/ooh/document/2016/03/automated_driving.pdf
[17] R. Hall and C. Chin, "Vehicle sorting for platoon formation: Impacts on highway entry and throughput," *Transp. Res. C, Emerg. Technol.*, vol. 13, nos. 5–6, pp. 405–420, Oct. 2005.
[18] H. G. Santos and T. A. Toffolo. *Mixed Integer Linear Programming With Python*. Accessed: Apr. 19, 2020. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/python-mip/latest/python-mip.pdf
[19] *United States Speed Limits*. Accessed: Apr. 19, 2020. [Online]. Available: https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/summary_state_speed_laws_12th_edition_811769.pdf
[20] *United Kingdoms Speed Limits*. Accessed: Apr. 19, 2020. [Online]. Available: https://www.gov.uk/speed-limits
[21] *India Speed Limits*. Accessed: Apr. 19, 2020. [Online]. Available: https://pib.gov.in/Pressreleaseshare.aspx?PRID=1539335
[22] *Europe Speed Limits*. Accessed: Apr. 19, 2020. [Online]. Available: https://ec.europa.eu/transport/road_safety/specialist/knowledge/speed/speed_limits/
[23] *NSW (Australia) Speed Limits*. Accessed: Apr. 19, 2020. [Online]. Available: https://www.rms.nsw.gov.au/roads/safety-rules/road-rules/speed.html