Nash Equilibrium Applied to Autonomous Traffic Intersections

Matthew Paul Spencer
Department of Electrical and Computer Engineering
Michigan Technological University
Houghton, MI 49931-1295, United States of America
mpspence@mtu.edu

***Abstract*—This paper explores the implementation of utilizing Game Theory's Nash Equilibrium concept within a road-side unit (RSU) traffic light controller for intersections designed strictly for autonomous vehicles (AVs). A virtual simulation is modeled and constructed using the established SUMO traffic simulation software, wherein each vehicle represented is assumed to be fully autonomous and operating at a SAE Automation Level of 5. The simulated environment imitates a single 100x100m city block with 4-way intersections placed at each corner. Each intersection is outfitted with a controller that collects vehicle routing data from each incoming lane of approaching vehicles in groups of up to 4, and then assigns priorities based on a weighted Nash Equilibrium matrix of potential routing directions by changing the individual light phases for each lane. This prioritization is done to let vehicles through the intersection in an optimized manner. A specific set of simulated vehicles is then observed along a shared route by recording the average time the vehicles spent stopped waiting at intersections, along with the total time it took the simulation to complete at a set traffic flow rate. The data is then compared to a standard 45-second traffic light phase timer replacing the Nash Equilibrium controller to detect if overall microscopic traffic flow is improved.**

***Index Terms*—Nash, equilibrium, autonomous, vehicle, traffic, intersection, SUMO, simulation**

I. Introduction

The age of autonomous vehicles may still seem like a distant reality, but the demand for quality solutions regarding problems and questions surrounding a fully autonomous future keep growing with each passing year. Whether these problems relate to autonomous off-road localization or building stable vehicular ad-hoc network (VANET) wireless propagation systems [1], the current field is ripe for innovation. Being able to not just communicate quickly and reliably between different mobile on-board units (OBUs), but being able to do so within a challenging fluid network topology is an important problem that many are working hard at solving today.

An interesting question though that arises when looking at these technologies as a whole is the following, "How can these AV technological improvements be used to make locomotion not just easier with autonomy, but also be used to optimize and improve overall micro and macroscopic traffic flow?" By setting out to try to answer this question in regards to living in tomorrow's fully autonomous world, it may give a better vision of applications yet to be developed to those working on building and funding autonomous vehicle technologies today.

A. Overview

Improved traffic flow is explored by outfitting the well established Game Theory concept of Nash Equilibrium towards a proprietary traffic light RSU controller. This Nash Equilibrium controller is meant to prioritize certain microscopic directional routes of select vehicles over others, and is focused on deciding in what order the vehicles within each group should move through its assigned intersection.

This prioritization is built off of the assumption that by letting a more desired "winning" combination of vehicles through first, such as two parallel approaching vehicles over vehicles that wish to make a left-turn, the overall traffic flow will be much better than using the standard fixed phase timers currently found operating traffic lights today. The Nash Equilibrium controller is allowed to make these prioritizations within the following SUMO simulation by being able to set the current light state either red or green for any direction. Even though some work has been made recently by state governments to implement autonomous traffic light infrastructure for future connectivity with AVs [2], these additions are focused mainly on reducing intersection accidents with drivers still having manual control over the operation of their vehicle.

As this discussion will point out, the Nash Equilibrium Controller solution could in theory be implemented without the physical need for a traffic light at all, and operate completely wireless using a dedicated RSU located at each intersection.

B. Outline

The next section will briefly cover the background and history of Game Theory, along with explaining the concept of a Nash Equilibrium to give the reader a better understanding of the Nash Controller RSU algorithm that is to follow. The algorithm description is then proceeded by an overview of the setup and implementation of the simulated intersections and 100x100m city block using the popular software simulation tool SUMO (Simulation of Urban MObility).

Resulting data gathered from a fixed set of vehicles with a shared route using the Nash controller and a standard 3 phase traffic light timer are then analyzed, and a determination is made regarding which implementation performed best under different traffic densities in an optimal and perfectly autonomous scenario. The results are then commented on and a conclusion is made alongside future work that can be done.

II. Game Theory

This section deals with explaining the underlying concepts of Game Theory, what a Nash Equilibrium is, and explains how it can be applied to autonomous vehicles intersections.

A. Background

Game Theory states that any interaction between two agents (players) that affect a desired outcome (payoff) can be labeled as a game [3]. Pioneered by John von Neuman and Oskar Morgenstern in the mid 1940s, Game Theory devised a way for people to logically and mathematically work out solutions when decision making by assigning weights and probabilities to projected outcomes. At first, the formal practice was mainly applicable towards economics and monetary problem solving, but it quickly spread into other scientific and political applications as its concepts grew in popularity. Elements of formalized Game Theory are actually believed to originate with the Greek philosophers Plato and Socrates as they tackled political and social human reasoning thousands of years ago.

Today, there exist different classifications for common types of recurring games that happen throughout everyday social interactions, and we will explain a few here briefly for brevity. There are closed-games like chess wherein a player will always gain what the other player lost, infinitely-long games are games that by definition have no end (this is a common type of game found within politics for example), and then finally there are cooperative and non-cooperative games. A cooperative game is one where the players in the game make decisions together to acquire a shared desired outcome, whereas a non-cooperative game is one played between players who make individual, rational fixed decisions without interaction or influence from any outside player or source [4]. Nash Equilibrium deals with the latter, as it also assumes that there always exists a set of decisions that benefit the maximum payoffs for each player.

B. Nash Equilibrium

Building off of the foundation of non-cooperative games, John Nash stated that, "... an equilibrium point is an n-tuple, such that each player's mixed strategy maximizes his payoff if the strategies of others are held fixed." [4]. In lay-mans terms, assigning weights to decisions and their outcomes for each player in a game will always result in a set of decisions that benefit everybody. This of course is due to the fact that the decisions being made by each player do not deviate from their best option for a given scenario. The n-tuple, or optimal set of decisions, is referred to today as the Nash Equilibrium of a game.

This non-cooperative Nash Equilibrium game is what emerges from a perfectly fully autonomous vehicle scenario, where each vehicle approaching a given intersection has a predestined route that does not change. This unchanging route being decided upon does not take into account the unpredictable variability of an actual traffic environment, such as vehicle break downs or traffic accidents, but rather is assumed not to change under perfect conditions. By applying a weighted decision matrix to find the Nash Equilibrium regarding the best decisions possible for each approaching group of vehicles, a "smart traffic light" RSU equipped intersection can make educated decisions and deal out priorities for vehicles traversing through the intersection based on the weighted calculated outcomes. Parallel traffic will be given the highest weighted priority, while potential crashes caused by left-hand turns or perpendicular routes will be given the lowest weighted priority. More on this is explained in detail throughout the later sections.

C. Previous Work

Applying Game Theory to traffic intersections and traffic flow is not a new concept, as researchers for the past few decades have been coming up with unique different methods and intersection control mechanisms to improve both the micro and macroscopic flow of traffic. The use of finding a Nash Equilibrium with respect to intersection traffic was proven to be feasible in the early 2000s by Champion and his colleagues, where they had proved a simple non-cooperative game example applied towards vehicles approaching a traffic light could be done [5]. Only as of recently within the past decade has it become a growing area of interest towards AVs, where the emergence of more capable and powerful hardware created the necessary types of vehicle-to-infrastructure (V2I) communication systems between RSU and OBUs to be developed.

By using Nash Equilibrium methods within traffic light controllers installed at intersections, it has been shown that when applied to a non-autonomous scenario the Nash equipped controller had improved microscopic traffic flow in lower density traffic areas with a higher degree of predictability [6] to make traffic light decisions based on the Nash Equilibrium in relation to vehicle wait times at intersections. It was found though *not* to preform optimally in an anarchistic environment, which isn't too surprising as Nash stated himself that his theory relies on a fixed decision game whose decisions do not change.

The Nash controller algorithm in this paper and its resulting simulation results regarding AV intersections is expected to resemble the results of a previous experiment, wherein the Nash Equilibrium was found between groups of approaching vehicles based on their vehicle speed and current traffic lane flow instead of using vehicle routing information as explained in later sections. Results of the previous experiment using speed and traffic flow had resulted in an average decrease of traffic light wait times by 56 to 58 percent when compared to using a more traditional timed traffic light at intersections [7]. Their results show a promising improvement.

III. Nash Equilibrium Traffic Controller

This section goes over the different parts and algorithmic implementation of the Nash Equilibrium Traffic Controller (NETC), which is responsible for finding the Nash Equilibrium of an approaching group of vehicles at a given intersection and changing the light state for each incoming lane accordingly.

The decision matrix that assigns weights to certain routes and how it is used will be explained, along with how the NETC applies these weights to priorities that are utilized to successfully control the flow of traffic safely through the intersection.

A. Weighted Decision Matrix

A weighted decision matrix is created and used to apply the concept of arbitrarily applying greater weights or values to more desirable microscopic vehicle routing decisions, creating a Nash Equilibrium point that the NETC can use to calculate and find which vehicle(s) have priority over others in a given group. The assigning of weights as mentioned prior is a common practice within Game Theory, and is therefore used here for the non-cooperative game played out within an AV intersection. Take chess again for example: in order for players to make better decisions throughout the game, weights are placed on more valuable pieces. A queen has the high value of 9 as a pawn merely has the value of 1, which makes sense since the queen is more valuable. Therefore, we can apply greater weights to some vehicles over others according to a preset hierarchy of desired routes:

1) Parallel routes: Vehicles that are facing each other get the highest weighted priority of 9, ensuring vehicles which are driving head-on through on intersections are allowed to keep moving. This situation can be seen by a vehicle wanting to move south-to-north (SN) through an intersection parallel to a second vehicle that wishes to cross through the intersection going north to south (NS).
2) Straight routes: The next priority is given to vehicles moving straight through an intersection, which also gets a value of 9. A tie could happen at this weighted priority by two parallel or perpendicular traveling groups of vehicles, therefore the NETC is programmed to handle this edge case internally NETC by using a tie-breaker method which is explained in further detail later.
3) Right-hand turns: The second to last priority goes to vehicles wishing to make right hand turns, and is given a priority of 2. This allows the NETC to handle multiple right-hand turns to happen simultaneously if needed, followed by letting right-hand turns go before left-hand turns.
4) Left-hand turns: Left hand turns are the most dangerous, as they can cause accidents if paired with vehicles wishing to do any of the routes listed above. For this reason all interactions with left-hand turns are given a negative priority, ensuring that they go last.

The NETC calculates the Nash Equilibrium point by finding the summed weight for each vehicle $\boldsymbol{v}$ assigned by a weighted matrix $\boldsymbol{M}$ in relation to every other vehicle in the total set $\boldsymbol{S}$ of approaching vehicles for a given intersection. Every set $\boldsymbol{S}$ comprises of $\boldsymbol{N}$ number of vehicles in maximum groups of 4, one for each direction; North, South, East and West. [Eq. 1]

$$v_k = \sum_{i=1}^{N} M(S_k, X_i) \quad \{X \in S | S_k \neq X\} \tag{1}$$

The higher resulting weighted value from Eq. 1 within the total set $\boldsymbol{v}$ are prioritized by the NETC over the lower ones, setting the traffic light phases accordingly for each lane. The completed decision matrix $\boldsymbol{M}$ can be seen in Tab. 1, where the directions describe the route of the vehicle. For example, (WE, EW) represents the weight of the vehicle whose route is moving west-to-east in relation to another vehicle whose route is moving east-to-west. The resulting weight for this interaction would be 9 using Eq. 1.

TABLE I
NETC WEIGHT MATRIX

	NE	NS	NW	ES	EW	EN	SW	SN	SE	WN	WE	WS
NE	0	0	0	-3	-3	-3	-3	-3	-3	-3	-3	-3
NS	0	0	0	-1	-1	-1	-1	9	-1	-1	-1	-1
NW	0	0	0	-2	-2	2	-2	2	2	-2	2	2
ES	-3	-3	-3	0	0	0	-3	-3	-3	-3	-3	-3
EW	-1	-1	-1	0	0	0	-1	-1	-1	-1	9	-1
EN	-2	2	2	0	0	0	-2	-2	2	-2	2	-2
SW	-3	-3	-3	-3	-3	-3	0	0	0	-3	-3	-3
SN	-1	9	-1	-1	-1	-1	0	0	0	-1	-1	-1
SE	-2	2	-2	-2	2	2	0	0	0	-2	-2	2
WN	-3	-3	-3	-3	-3	-3	-3	-3	-3	0	0	0
WE	-1	-1	-1	-1	9	-1	-1	-1	-1	0	0	0
WS	-2	-2	2	-2	2	2	-2	-2	-2	0	0	0

B. Algorithm Pseudo-Code

The NETC algorithm explained in Alg. 1 deals with the general steps and procedure taken within the NETC to deal out priorities. The algorithm sets a priority 1D array structured as [N,E,S,W] from 1 to 4 to signal in which order the incoming lane's traffic light state should be toggled from either green (go) to red (stop), or vice-versa. Once a group of approaching vehicles are detected and a priority array is generated, it is then immediately executed on the simulated traffic light. Only tie-breakers are applied towards negative values, as positive ties indicate either group right-hand turns or parallel traffic. Not shown is the edge case for perpendicular traffic mentioned earlier, were a separate function takes care of the tie-breaker based on the duration time of incoming vehicles, in-which the pair with the longer average duration time goes through first. The algorithm was implemented using the latest version of Python 3 at the time of writing, along with the TraCI library to interface with the SUMO traffic network simulation.

Instead of implementing a VANET propagation model using an external network simulator such as NS-3, detectors within the simulation were placed along each lane of approaching vehicles to signal to the intersection NETC that a vehicle is approaching and to extract its current routing data for computation. These detectors are placed in close proximity to the traffic light, negating the need for a queue to keep track of future groups. The NETC only handles the current group of vehicles at the intersection and then resets once each group is handled accordingly. Each vehicle's route is programmed beforehand within the simulation for easy extraction. This

attempts to simulate a very basic wireless line-of-sight propagation implementation that could be used within future RSU devices communicating with each approaching vehicle's OBU for the requested routing information.

IV. Simulation

The traffic simulation software SUMO was modified to implement the NETC on a modern Windows 10 computer outfitted with an Intel i5-10400 processor running 16GB of DDR4 2133 MT/s RAM memory. This section goes into detail about the creation and parameters set to achieve a successful simulation of a 100x100m city block containing 4 intersections that can have both a standard traffic controller implementation alongside the proprietary NETC for comparison benchmarking purposes.

A. Implementation

First, straight 100 meter 2-lane roads are developed along with each intersection by specifying their locations within the edge and node XML configuration files. The XML configuration files are loaded by SUMO during the initialization step of the simulation. Intersections are then properly formed within the connection configuration file by specifying which lanes are going to and from each labeled intersection junction ID number, along with the connecting entry and exit lanes for the simulated vehicles to spawn and leave from. Fig. 1 displays the overall setup of the custom mapped city-block with its junction and lane labels.

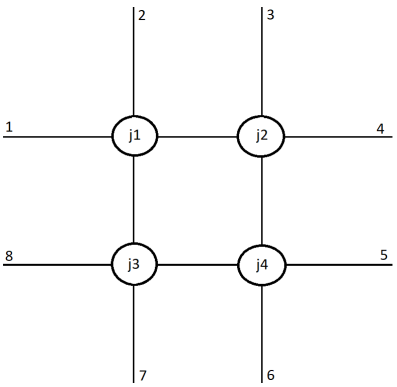

Fig. 1. City-Block SUMO Map

A detector XML configuration file is then created to install simulated vehicle detectors along each approaching lane at every intersection to then be used by the NETC simulation. Junction "j1" is shown in Fig. 2, where the yellow squares represent the simulated detectors. These detectors signal the route and duration data for the most recent vehicle that crosses over it back to the NETC to calculate the equilibrium point prioritization of the current group. The detectors are placed close towards the intersection, guaranteeing that no vehicle from a future or past group is accidentally calculated alongside the current priority phase. Each detector along with which lane they occupy is thus saved and stored beforehand to relay the correct information to the intersection's NETC.

Algorithm 1 NETC

while 1 **do** // Always check for approaching vehicles
  $M \leftarrow$ Initialize 2D weight matrix (Table 1)
  $N \leftarrow 4$ // Init number of lanes
  $b \leftarrow$ false // Init found vehicle boolean
  $n \leftarrow 0$ // Init vehicle counter
  $P[N] \leftarrow 0$ // Init priority array [N,E,S,W]
  $v[N] \leftarrow 0$ // Init vehicle array that holds summed weights [N,E,S,W]
  $D[N] \leftarrow$ null // Init 4 detectors for each inlet lane direction to sense approaching vehicles [N,E,S,W]
  $S[N] \leftarrow 0$ // Init array to hold detected vehicle data
  TrafficLightState() $\leftarrow$ [0,0,0,0] // Set all traffic light directions to red
  for $i = 1, ..., N$ **do**
    if $D[i]$!= null **then**
      $r \leftarrow D[i].route$ // Get vehicle route direction
      $t \leftarrow D[i].travel$ // Get travel time for tie breakers
      $S[i] \leftarrow [r, t]$ // Store vehicle data for weight matrix
      $b \leftarrow$ True // Find priority of approaching vehicles
      $n \leftarrow n$+1 // Increase vehicle counter
    end if
  end for// Loop through found vehicles
  if b == True **then**
    for $i = 1, ..., N$ **do**
      for $j = 1, ..., N$ **do**
        $v[i]$ = $[v[i]$ + $M[S[i][1], S[j][1]]$ // Sum weights
      end for
    end for// Deal with any potential ties
    for $i = 1, ..., N$ **do**
      for $j = 1, ..., N$ **do**
        if $v[i]$ == $v[j]$ and $v[i] < 0$ **then**
          if $S[i].t > S[j].y$ **then**
            $v[i]$ ++ // Tie-breaker
          end if
          if $S[i].t < S[j].y$ **then**
            $v[j]$ ++
          end if
        end if
      end for
    end for// Populate priority matrix
    for $k = 1, ..., n$ **do**
      for $i = 1, ..., N$ **do**
        if $v[i]$!= void and $v[i]$ == max(v) **then**
          $P[i]$ = n // Assign current priority
          $v[i] = null$ // Mark vehicle as counted
        end if
      end for
    end for// Change traffic light
    for $i = 1, ..., n$ **do**
      $T[N] \leftarrow 0$ // Init traffic light state array [0,0,0,0]
      for $k = 1, ..., N$ **do**
        if $P[k]$ == i **then**
          $T[k]$ = 1
        end if
      end for
      TrafficLightState() $\leftarrow T$ // Set traffic light state
    end for
  end if

A second simulation is generated alongside the NETC simulation using standard traffic light controllers at each intersection, where the NS and WE facing directions are opposite of each other, alternating between green and red phases every 45 seconds. A yellow phase is also included which signals to the approaching vehicles to slow down, and happens for 7 seconds during the transition from green to red.

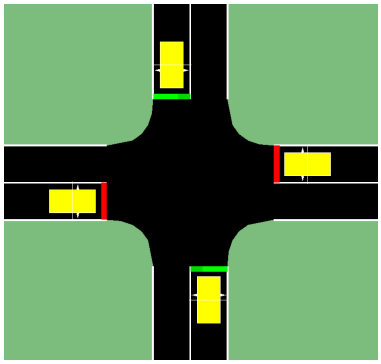

Fig. 2. SUMO Intersection "j1"

Vehicles are randomly generated to traverse throughout the map shown in Fig. 1, entering and leaving at specified lane entry and exit points located on the ends of each lane towards the outskirts of the map. The simulation is randomly seeded to ensure that the simulation with the same vehicles and their paths are reproducible for testing both the NETC and standard traffic controllers. Total simulation time and average vehicle stop time is gathered at the end of simulation from vehicles traveling along the following route; $lane1 \rightarrow j1 \rightarrow j2 \rightarrow j4 \rightarrow lane5$

B. Parameters

Parameters for the simulation were chosen based on the overall goal of making a perfect scenario for AVs to operate in. This includes such things as giving each vehicle node the ability to have fast and controlled braking and acceleration, with a fixed distance gap of 5m between each vehicle node's front end and next node's back end. Every road is set to a length of 100m and given speed limit of 11.176 meters/s (which converts to 25mph), which are common urban city-block distance measurements and speed limitations found within the USA today.

Intersection crossing functionality is also modified to instantly move vehicles through the intersection on their desired route instead of having SUMO auto-generate intersection lanes to visualize the movement. Each group of vehicles belonging to a current priority phase of the NETC is given 1 simulation second to traverse through the intersection. No left-hand turn lanes are implemented, as this experiment is only considering 2-lane roads maximum for AV operation. Traffic collisions are allowed to ensure that the NETC is safely and properly moving traffic through each intersection without accidents, also ensuring that the NETC routing functionality is safe and works as intended. No potential VANET signal propagation noise or loss is accounted for, along with the expectation that none of the vehicles during the simulation will break down or cause unwanted behavior to jam up the traffic network.

$$q_{lane} = \frac{n_{vehicles}}{t_{seconds}} * \frac{3600s}{1hour} \tag{2}$$

Traffic density is manipulated during the simulation using Eq. 2 by setting the vehicle spawn rate per hour $\boldsymbol{q}$ (v/l/h) for each 100m lane edge approaching an intersection (lanes 1 - 8 as seen in Fig. 1). The variable $\boldsymbol{t}$ is the average time in seconds that elapses between $\boldsymbol{n}$ number of vehicles being spawned into the simulation. The simulation sets $\boldsymbol{q}$ to be the same for each incoming lane 1 - 8. Data is gathered by setting each subsequent simulation to run each lane at an increasing vehicle spawn rate of $\boldsymbol{q}$ v/l/h by continually decreasing $\boldsymbol{t}$ by 1 second and fixing $\boldsymbol{n}$ = 1 vehicle. The simulation's resulting vehicle data is then observed at different spawn rates between using the NETC and standard (STD) traffic light controller's traffic light timer in the next section.

V. Results and Discussion

The following data tables Tab. 2 and Tab. 3 with their respective plots shown in Fig. 3 through 6 are gathered by setting $\boldsymbol{t}$ = [40, 39, 38 ..., 7], which sets each incoming lane 1 through 8 [Fig. 1] to have a fixed vehicle spawn rate (traffic flow) per hour by applying Eq. 2 to each incoming lane that is set to spawn vehicle nodes at rate $\boldsymbol{q}$. The data gathered relates to the vehicle testing group following a specified route as explained earlier. The waiting time each vehicle in the group spends stopped at an intersection is added up and averaged over the total number of vehicles in the specified group. Overall simulation time for all of the simulated vehicles to reach their destinations is recorded for each time step $\boldsymbol{t}$ as well. Line plots in Fig. 3 through Fig. 6 were generated by importing the data into MATLAB.

TABLE II
Total Simulation Time

$\boldsymbol{t}$	$\boldsymbol{q}$	NETC Sim Time (s)	STD Sim Time (s)
40	90	3903	4019
35	102.857	3906	4017
30	120	3892	4018
25	144	3899	3976
20	180	3905	3976
15	240	3900	4069
14	257.1429	3901	11236
13	276.9230	3902	18195
12	300	3900	25103
11	327.2727	3898	30170
10	360	3902	33313
9	400	3902	37375
8	450	22110	43003
7	514.286	64562	53189

Average vehicle wait time was drastically decreased using the NETC compared to the STD controller consistently throughout the majority of the experiment. Average wait times are shown to be mere fractions of a second when compared

TABLE III
AVERAGE VEHICLE WAIT TIME

t	q	NETC Wait Time (s)	STD Wait Time (s)
40	90	0.7262	37.6788
35	102.8571	0.0498	38.7811
30	120	0.1190	41.3580
25	144	0.1334	43.9742
20	180	0.2327	50.5799
15	240	0.4253	108.8877
14	257.1429	0.4739	1622.5587
13	276.9230	0.6324	3255.4120
12	300	0.8242	4693.2091
11	327.2727	1.0578	5284.2030
10	360	1.5442	5514.9951
9	400	2.1928	5834.3645
8	450	2333.2807	6069.7030
7	514.2857	7945.0755	6469.4684

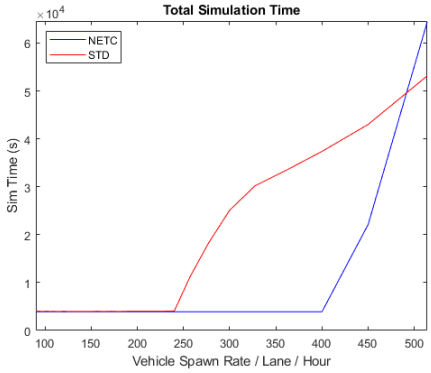

Fig. 3. Total Simulation Time

to the STD wait times shown in Fig. 6. A drastic difference at the microscopic level when looking at the macroscopic overall simulation time is observed in Tab.2 and Fig.4, as they show that both controllers are able to handle the desired traffic flow similarly well despite the gap in difference regarding average wait times at intersections displayed in Tab. 3 and Fig. 5. The spawn rate of the STD controller is observed to grow worse and fail to keep up with the demand when $t < 15$. Average wait times for the STD controller increases steeply when q

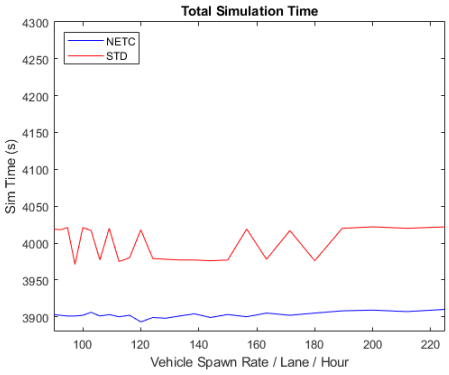

Fig. 4. Total Simulation Time (Zoomed)

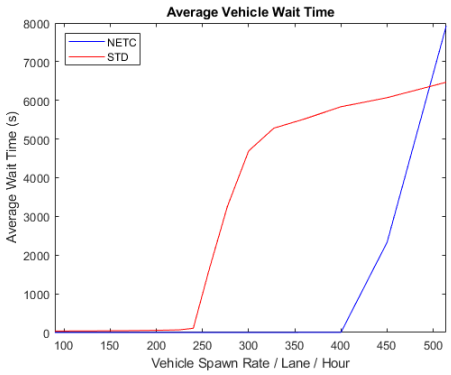

Fig. 5. Average Vehicle Wait Time

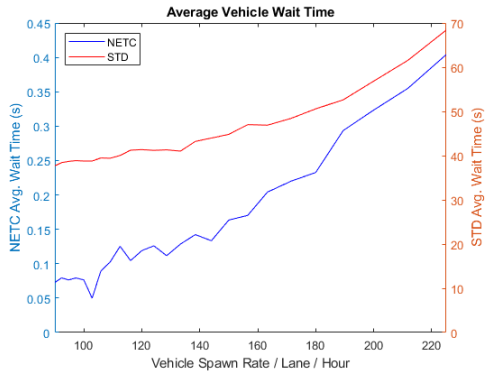

Fig. 6. Average Vehicle Wait Time (Zoomed)

equals between 230 and 300v/l/h, while the NETC controller holds steady until it starts exponentially increasing sharply after q = 400v/l/h. This result shows that the NETC not only out preforms the STD at lower densities, but also can also preform optimally where the STD starts to fail. The sudden increase in overall simulation time and average wait times between the NETC and STD simulations cause the lowered resolutions seen within the data recorded in Tab. 2 and Tab. 3 and displayed in Fig. 3 and Fig. 5.

The similar plots make sense between Fig. 3 and Fig. 5 considering the average wait time is correlated with the total overall simulation time. This means that the longer vehicles spend waiting at each intersection, the longer the overall simulation time. The average vehicle wait time of the STD traffic controller is only shown to more efficient after $q > 450$v/l/h, where the NETC is observed to be heavily exponentially increasing in wait and overall simulation time compared to the STD implementation. The STD controller total simulation time is shown be on a slightly more aggressive upward trajectory in Fig. 3 compared to its average wait time in Fig. 5 after q is increased to 350v/l/h, but the STD controller is quickly surpassed by the NETC at this point which is exponentially rising. These sudden increases between the STD and NETC have more to do with SUMO's preset underlying built-in parameters handling traffic jams than the traffic light controller implementations themselves.

It is observed between Fig. 3 and Fig. 5 that there exists independent points where the NETC and STD stop being efficient. The point where the STD controller stops being efficient is $t < 15$, and the point for the NETC is $t < 9$. The unusually high jumps where the relationship between the overall simulation time and wait time break down observed between Tab. 2 and Tab. 3 mentioned above, and seen in Fig. 3 and Fig. 5, is caused by the SUMO simulation. SUMO removes vehicles from the simulation that spend over 300 seconds being idly stopped at an intersection, and then proceeds to re-spawn the vehicle back into the simulation along its generated path once enough space in the starting approaching lane becomes available. It can be observed that the STD controller stats to recover once enough vehicles are removed, but the NETC keeps exponentially increasing due to its implementation.

The current NETC algorithm is to be blamed for the rapid exponential increase due to there being no queuing system or other safeguard mechanisms put into place to check if vehicles have successfully made it through the intersection during their allotted priority during a traffic jam, and thus the NETC is not able to successfully deal out further priorities until the current vehicle priority group is handled appropriately. This oscillation during a traffic jam causes vehicles to start being idle for too long, causing SUMO to start removing them from the simulation and waiting to re-enter them.

Looking at the intersection in Fig. 2 during a simulation where $q >= 450$v/l/h, the green and red light phases dealt out by the NETC start to oscillate quickly as the NETC rapidly calculates the priorities of the current group stopped at the intersection over and over again while they remain stuck due to their outgoing lane being jammed. All vehicles remain in a jammed state at a fixed distance of 5m away from one another. The simulation is programmed to give each vehicle 1 simulation second allocated for each priority change, which explains why the sudden exponential increase in average wait and simulation times for the NETC happen once the average wait time reaches above 1 second as shown in Tab. 3. Each NETC continues to loop through current jammed priorities until one is able to move a vehicle into an opening caused by SUMO removing a vehicle from the simulation.

A solution to this oscillation matter would be to implement a second set of detectors onto each outgoing lane for each intersection, sending an ACK signal back to the NETC when the current set of prioritized vehicles is detected to have successfully crossed the intersection. When the detectors signal a successful crossing, the NETC would then deal out the next priority, ensuring that the unwanted rapid oscillation problem does not happen. The only concerns with this solution would be a "deadlock" scenario where all 4 intersections are waiting to deal out a priority that cannot happen due to outgoing traffic being jammed, which is currently being prevented by SUMO removing vehicles that reach 300 seconds of idle time. Due to this concern, the NETC controller could then be programmed to switch between the NETC algorithm and the STD controller when a certain traffic flow threshold is reached as a fail-safe mechanism. In this simulation where vehicles are given 1 second to cross with near instantaneous acceleration and deceleration, the threshold would be programmed to exist between $300 < q < 327$ v/l/h based on the information given in Tab. 2 and Tab. 3.

Therefore, before the threshold of q being 450v/l/h is reached (where the NETC controller can no longer handle priorities in a timely manner due to the rapid oscillation problem explained previously), the NETC was shown to drastically improve microscopic traffic flow at intersections by implementing the NETC algorithm and weighted decision matrix when compared to using a standard 45 second traffic lane phase timer during a perfect AV scenario. The NETC was also shown to be able to handle greater densities at optimally low waiting times where the STD couldn't. Further experimentation and optimizations could be explored to enable the NETC to handle even higher traffic densities.

ACKNOWLEDGMENT

I would like to thank my family and friends for their continued love and support which makes work like this possible. I would also like to thank my professors and advisors within the Electrical and Computer Engineering department at Michigan Technological University who never cease to inspire and push me forward throughout my academic career.

REFERENCES

[1] A. Silva, N. Reza, A. Oliveira, "Improvement and Performance Evaluation of GPSR-Based Routing Techniques for Vehilcular Ad Hoc Networks", IEEE Access, Department of Electrical and Computer Engineering, Michigan Technological University, Feb 27, 2019.

[2] D. Weingarten, "From new traffic singals to mumble strips, safety is a focus for MDOT", MDOT Bulletin, Nov. 9, 2018, [Online] Available: https://content.govdelivery.com/accounts/MIDOT/bulletins/219625f, [Accessed: Apr. 18, 2021].

[3] D. Ross, "Game Theory", The Stanford Encyclopedia of Philosophy 2019 Winder Ed., 2019, [Online] Available: https://plato.stanford.edu/entries/game-theory/, [Accessed: Apr. 18, 2021]

[4] J. Nash, "Non-Cooperative Games", The Annals of Mathematics, vol. 54, no. 2, pp. 286-295, Sep. 1951.

[5] A. Champion, R. Mandiau, S. Espie, C. Kolski, "Multi-Agent Road Traffic Simulation: Towards Coordination by Game Theory Based Mechanism", Conf. ITS World Congress Australia, Oct. 2001.

[6] P, Tersmette, A. Czechowski, F. Oliehoek, "Using Game Theory to Analyse Local and Global Performance of Traffic Signal Control Strategies", IFAC Proceedings Volumes, vol. 39 iss. 12, Jan. 2016, pp. 319-324.

[7] H. Abdelghaffar, H. Yang, H. Rakha, "Isolated Traffic Signal Control using a Game Theoretic Framework", 2016 IEEE 19th ITSC Brazil, Nov. 1-4, 2016.