# AVIAN Documentation

***Release 1.0.0***

**Mahmoud Pourmehrab**

**May 16, 2018**

# CONTENTS

CHAPTER

ONE

# OVERVIEW

## 1.1 Introduction

The program is to simulate the performance of an isolated intersection under traffic of automated vehicles and conventional vehicles with variety of signal control methods. It is developed as part of the *AVIAN* project supported by National Science Foundation under grant award 1446813. The base implementation was done in MATLAB programming language in 2015 - 2017 (see[1]). For comments and questions please contact email me at mpourmehrab@ufl.edu. For details on the AVIAN project visit AVIAN.ESSIE.UFL.EDU.

**Note:**

- The interpreter version requirement is set to `3.6.4`. If using conda, do:

```
$ conda update conda
$ conda install python=3.6.4
```

- Update pip and install packages using:

```
$ python -m pip install --upgrade pip
$ pip3 install -r requirements.txt
```

- To execute the code:

```
$ python main.py <intersection_name> <optimization_algo> <run mode>
```

- **The printed information in the command line may have the following prefixes:**
  - `>>>` phase addition to the end of SPaT
  - `<<<` phase removal to the beginning of SPaT
  - `>->` phase extension (only can happen to the last phase)
  - `\\\` vehicle addition
  - `///` vehicle removal
  - `>@>` vehicle departure scheduled
  - `>*>` vehicle trajectory planned through base SPaT
  - `>#>` vehicle trajectory planned through unserved module (temporary trajectory)
- **Outputs are stored under `/log/`:**

[1] Pourmehrab, M., Elefteriadou, L., Ranka, S., & Martin-Gasulla, M. (2017). *Optimizing Signalized Intersections Performance under Conventional and Automated Vehicles Traffic*. arXiv preprint arXiv:1707.01748 (link).

  - The outputs are named by the format of `/log/<intersection name>/<UTC timestamp>_<sc>_*.csv`
  - The UTC time stamp sample that prefixes the filename is `13May2018_14-15-20`
  - `<intersection name>_vehicle_level.csv` includes input csv plus the `departure time`, `vehicle ID` and `elapsed time` columns
  - `<intersection name>_trj_point_level.csv` includes the trajectory points
- SI units are used (speed in $m/s$, length in $m$, time in $s$, acceleration in $m/s^2$)

---

**Warning:**

- As of now, no traffic generator module is developed as part of the main workflow. The traffic is input in csv format under `/data/<intersection name>/` directory.
- For simulation, the directory `/data/<intersection name>/` shall include `<timestamp>_sc_sig_phase_level`, `_trj_vehicle_level`, `_trj_point_level.csv` which has the scenarios to be tested. Note the filename should match the intersection name.
- **The csv file must include columns with the following heading:**
  - *lane*: lane index (one-based)
  - *type*: vehicle type {0: *CNV*, 1: *CAV*}
  - *arrival time*: arrival time at the stop bar measured in second from a fix reference point
  - *curSpd*: detection speed
  - *dist*: detection distance
  - *desSpd*: desired speed
  - *dest*: destination {0: right turn, 1: through, 2: left}
  - *L*: length of vehicle
  - *maxAcc, maxDec*: maximum acceleration, deceleration rate vehicle can execute

| lane | type | arrival time | L | maxAcc | maxDec | dest | curSpd | desSpd | dist |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 1 | 33.2 | 4.6 | 3.0 | -4.6 | 0 | 5.7 | 16.3 | 500.0 |
| 2 | 0 | 27.6 | 4.8 | 3.0 | -4.6 | 1 | 5.5 | 18.0 | 500.0 |
| 6 | 1 | 35.2 | 4.8 | 3.0 | -4.6 | 1 | 8.1 | 17.6 | 500.0 |
| 8 | 1 | 41.2 | 4.6 | 3.0 | -4.6 | 2 | 9.7 | 17.5 | 500.0 |
| 3 | 0 | 49.0 | 3.7 | 3.0 | -4.6 | 0 | 10.8 | 18.5 | 500.0 |
| 1 | 1 | 28.6 | 4.0 | 3.0 | -4.6 | 1 | 5.4 | 18.5 | 500.0 |
| 4 | 0 | 40.0 | 4.2 | 3.0 | -4.6 | 0 | 7.8 | 16.7 | 500.0 |
| 6 | 0 | 50.8 | 4.2 | 3.0 | -4.6 | 2 | 9.4 | 16.9 | 500.0 |
| 7 | 1 | 64.9 | 4.6 | 3.0 | -4.6 | 2 | 11.3 | 17.2 | 500.0 |

**You can add any intersection in the `src/intersection/data.py`. The list of all available intersections is:**

- `13th16th`: A physical one, google map it in Gainesville for the image and lane assignment detail
- `TERL`: Located at 2612 Springhill Road, Tallahassee, FL 32305. Note the lane numbering in the code

is 1: Southbound (all movements), 2: Westbound (through and right turn), 3: Westbound (left turn), 4: Northbound (all movements), 5: Eastbound (through and right turn), 6: Eastbound (left turn).

- `reserv`: for the reservation based model intersection that has 12 incoming lanes: 3 per approach and all lanes are exclusive (for more detail check UT Texas AIM).
- Some possible intersections to add are `RTS, 42nd40th, SolarPark`

**You also can choose from the following signal control methods:**

- `GA`
- `pretimed`
- `MCF` (under development)
- `actuated` (under development)

**You can run in either of the following modes (*pay attention to the requirements of each run mode*):**

- `simulation`
- `realtime`

The **UML diagram** of the project is as the following (*you may want to zoom in*):

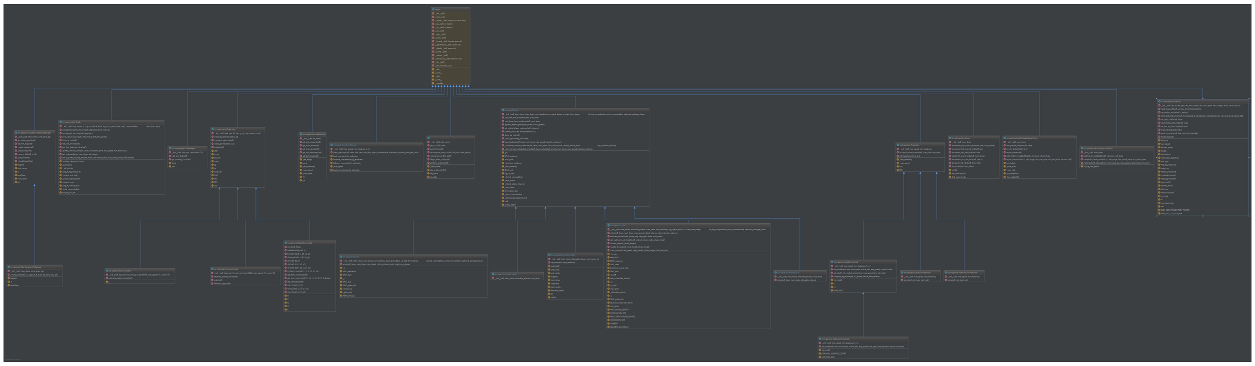

Fig. 1: The UML diagram of project.

## 1.2 Terms

**AVIAN** Autonomous Vehicles at Intelligent intersections and Advanced Networks. Read more here.

**SPaT** Signal Phase and Timing

**CAV** Connected and Automated Vehicle

**CNV** Conventional Vehicle

**Trajectory** A list of triples (time stamp, distance to stop bar, speed) to describe a vehicle's movement.

**LP** Linear Program used to optimize the AVs trajectories. Read more here.

**CPLEX** An optimization solver developed by IBM. Read more here.

**Gipps CF** A car following model developed by Gipps 1981 that models conventional vehicles movement. Read more here.

**pretimed** Refers to a type of signal control that the phase sequence and timing is predefined and get executed in a cyclic manner. Read section 5.2.1 of the Traffic Signal Timing Manual.

**GA** Genetic Algorithm used to optimize SPaTs decision. here.

**badness** In the context of GA, badness defines as the negative of fitness of an individual where less is preferred.

**MCF** Minimum Cost Flow model used to optimize SPaTs decision. Read more here.

## 1.3 Main Script

The `main.py` file implements the following work flow:

`main.check_py_ver()`
checks the python version to meet the requirement (`ver 3.6.4`)

`main.run_avian(`*inter_name, method, sc, start_time_stamp, do_traj_computation, log_at_vehicle_level, log_at_trj_point_level, log_signal_status, print_commandline, optional_packages_found*`)`

**For logging and printing of information set boolean variables:**

- `log_at_trj_point_level` saves a csv under `\log` directory that contains all trajectory points for all vehicles
- `log_at_vehicle_level` saves a csv file under `\log` directory that contains departure times and elapsed times and vehicle IDs

**The work flow is as the following:**

- Tests for python version
- Checks the input arguments to be valid
- **Instantiate:**
  - *`Intersection`*
  - *`Lanes`*
  - *`Traffic`*
  - **trajectory planners: all bellow**
    - *`LeadConventional`*
    - *`LeadConnected`*
    - *`FollowerConventional`*
    - *`FollowerConnected`*
  - **signal: one of followings**
    - *`GA_SPaT`*
    - *`Pretimed`*
- **set simulation start time to when first vehicle shows up**
  - *`TimeKeeper`*
- **main loop stops only when all vehicles in the provided input traffic csv file are assigned a departure time.**
  - remove vehicles that are served
  - update SPaT

- update vehicle information (includes addition too)
- do signal
- plan trajectories
- update time and check of termination

**Parameters**

- **inter_name** (*str*) – intersection name
- **method** (*str*) – pretimed, GA, . . .
- **sc** (*int*) – scenario number (*should match the appendix of the input csv filename*)
- **do_traj_computation** –
- **log_at_vehicle_level** –
- **log_at_trj_point_level** –
- **log_signal_status** –
- **print_commandline** –
- **optional_packages_found** – optional packages for testing

**Returns**

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**Organization** University of Florida

CHAPTER

TWO

# SIMULATOR/DATA

## 2.1 Simulation

**class** src.time_keeper.**TimeKeeper**(*sim_start*, *resolution=2.0*)

Bases: object

**Objectives:**

- Keeps the time
- Moves the simulation clock forward

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**__init__**(*sim_start*, *resolution=2.0*)

Clock keeps the simulation starting time in seconds.

**Parameters**

- **sim_start** – start time of simulation to be initialized
- **resolution** – Simulation resolution: the time steps to move the simulation forward in seconds

**next_sim_step**()

Move simulation clock forward

**get_running_clock**()

Get the current clock

## 2.2 Python Data

data.data.**get_general_params**(*inter_name*)

**Returns max speed ($m/s$), min_headway ($s$), detection range ($m$), $k, m$, number of lanes. Where:**

- $k$ = # $n$ will be in 0, ..., k-1 (odd degree of polynomial is preferred: k to be even)
- $m$ = # to discretize the time interval

**Note:**

- The distance to stop bar will be input from either csv file or fusion. However, the number provided here is used for generic computations.

---

**Warning:** Is required for trajectory optimization

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

`data.data.`**`get_pretimed_parameters`**`(`*inter_name*`)`

This returns the parameters needed for pre-timed control.

---

**Note:**

- The sequence field includes the phases and is zero-based.
- You need to compute green splits and yellows, all-reds based on traffic flow theory.

---

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

`data.data.`**`get_conflict_dict`**`(`*inter_name*`)`

Returns a **dictionary** of sets where the **keys** are lane numbers and must be coded in one-based and the **value** for each key is a set of lane numbers that are in conflict with the key lane (again must be one based).

An intersection configuration can be specified by its lanes and movements (left, through, right) that are allowed in each lane. The lane-lane incidence matrix of an intersection is a squared matrix that holds 1 (shown by solid circles in the figures), if two lanes are in conflict. The standard types of conflicts that may wanted to be avoided are cross, merge, and diverge conflicts. Depending on the design, the definition of conflicts points can be broader or more limited. For instance, if volume of a lane is too low and extensive gaps can be found, some of conflict points can be relaxed as non-conflicting points. In the following figures, only cross and merge conflict points are indicated.

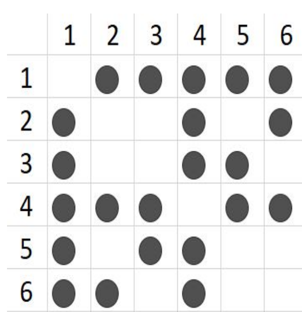


Fig. 1: The TERL facility.

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

`data.data.`**`get_phases`**`(`*inter_name*`)`

Returns a dictionary of sets The key is the phase number is one-based The value to a key is set of lanes included

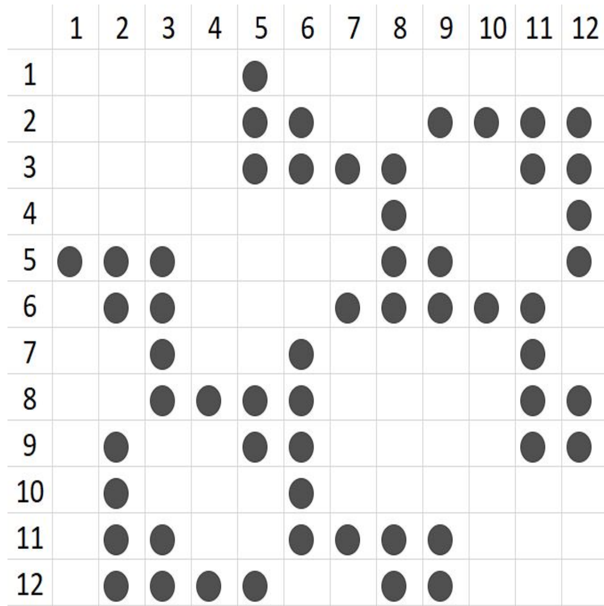

Fig. 2: The reservation-based intersection.

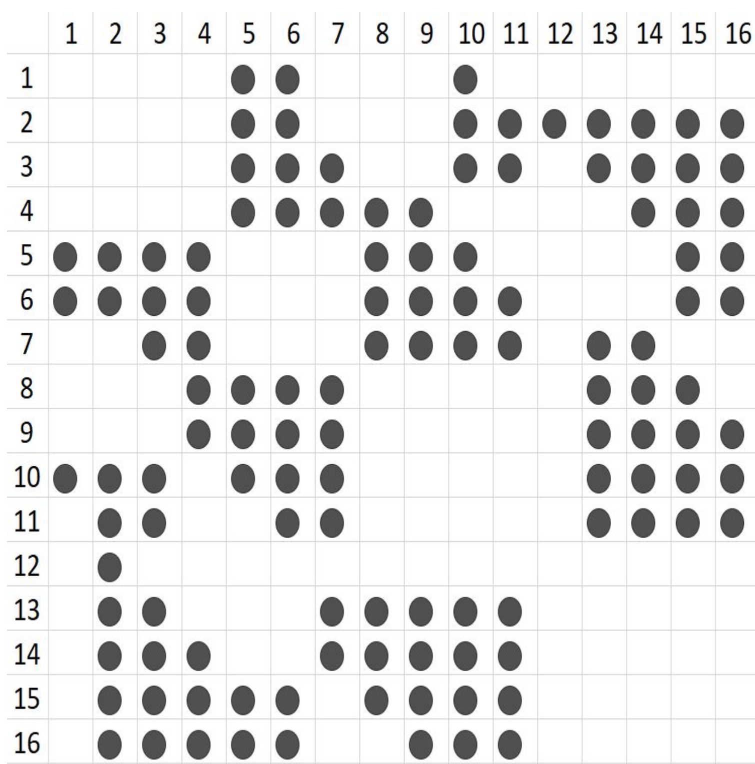

Fig. 3: The intersection of 13th and 16th, Gainesville, FL.

in that phase (lanes are one-based too) Use the phase enumerator for new intersections of refine manually The rule is each set must include non-conflicting lanes # todo add the phase enumarator to the project

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

`data.data.`**`get_signal_params`**(*inter_name*)

Required for GA signal control ALL yellow, all-red, min green, max green times are in seconds

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

# CHAPTER THREE

# INTERSECTION

Intersection: gets parameters that are needed to specify the configuration of problem

**class** src.intersection.**Intersection**(*int_name*)

Bases: object

**Objectives:**

- Keeps intersection parameters

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**__init__**(*int_name*)

**Parameters** **int_name** – comes from what user input in the command line as the intersection name

**get_poly_params**()

**Returns** K and M

**get_num_lanes**()

**get_max_speed**()

**get_min_headway**()

**get_det_range**()

**class** src.intersection.**Lanes**(*num_lanes*)

Bases: object

Dictionary that the key is lane index and value is an arrays that keeps queue of vehicle in that lane.

**Objectives:**

- Keeps vehicles in order
- Keep track of index of last vehicle in each lane (useful for applications in Signal())
- Remove served vehicles, and update first unserved and last vehicle's indices accordingly
- Check if all lanes are empty

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**__init__**(*num_lanes*)
Data Structure for keeping vehicles in order in the lanes in the form of a dictionary of arrays

**Parameters** **num_lanes** – number of lanes

**decrement_first_unsrvd_indx**(*lane*, *num_served*)
When vehicles get served, the first index to the unservd vehicle in a lane should change.

**Parameters**

- **n** – number of served vehicle
- **lane** – the lane at which the vehicles are served

**increment_first_unsrvd_indx**(*lane*)

**increment_last_veh_indx**(*lane*)

**reset_first_unsrvd_indx**(*num_lanes*)

**decrement_last_veh_indx**(*lane*, *n*)

**purge_served_vehs**(*lane*, *indx*)
Deletes vehicles from 0 to `indx` where `indx` is the pointer to the last served

---

**Note:** deletion also includes vehicle at `indx`

---

**Parameters**

- **lane** (*int*) – the lane number
- **indx** – from vehicle 0 to `indx` are intended to be removed by this method

**all_served**(*num_lanes*)

**Returns** True if all lanes are empty, False otherwise

**class** src.intersection.**Vehicle**(*det_id*, *det_type*, *det_time*, *speed*, *dist*, *des_speed*, *dest*, *length*, *amin*, *amax*, *indx*, *k*)
Bases: `object`

**Objectives:**

- Defines the vehicle object that keeps all necessary information
- Update/record the trajectory points once they are expired
- Keep trajectory indexes updated
- Print useful info once a plan is scheduled
- Decides if a trajectory re-computation is needed
- Quality controls the assigned trajectory

---

**Note:** Make sure the `MAX_NUM_TRAJECTORY_POINTS` to preallocate the trajectories is enough for a given problem

---

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**EPS = 0.01**

**MAX_NUM_TRAJECTORY_POINTS = 300**

**MIN_DIST_TO_STOP_BAR = 50**

**__init__**(*det_id*, *det_type*, *det_time*, *speed*, *dist*, *des_speed*, *dest*, *length*, *amin*, *amax*, *indx*, *k*)
Initializes the vehicle object.

> **Attention:**
>
> - The last trajectory point index less than the first means no trajectory has been computed yet
> - The last trajectory index is set to -1 and the first to 0 for initialization purpose
> - The shape of trajectory matrix is $3 \times n$ where $n$ is the maximum number of trajectory points to be held. The first, second, and third rows correspond to time, distance, and speed profile, respectively.
> - The vehicle detection time shall be recorded in `init_time`. GA depends on this field to compute travel time when computing *badness* if an individual.

**Parameters**

- **det_id** (*str*) – the *ID* assigned to this vehicle by radio or a generator
- **det_type** – 0: *CNV*, 1: *CAV*
- **det_time** – detection time in $s$ from reference time
- **speed** – detection speed in $m/s$
- **dist** – detection distance to stop bar in $m$
- **des_speed** – desired speed in $m/s$
- **dest** – destination 0: right turn, 1: through, 2: left
- **length** – length of vehicle in $m$
- **amin** – desirable deceleration rate in $m/s^2$
- **amax** – desired acceleration rate in $m/s^2$
- **indx** – the original row index in the input csv file
- **k** – number of coefficients to represent the trajectory if vehicle is connected
- **self.trajectory** – keeps the trajectory points as columns of a 3 by N array that N is `MAX_NUM_TRAJECTORY_POINTS`
- **self.first_trj_point_indx** – points to the column of the `trajectory` array where the current point is stored. This gets updated as the time goes by.
- **self.last_trj_point_indx** – similarly, points to the column of the `trajectory` where last trajectory point is stored.
- **self.poly** (*dictionary that keeps the reference time and the coefficients to reproduce trajectory of an AV*) – only CAVs trajectories are represented in the form of polynomials as well as the trajectory matrix
- **self.earliest_departure** – the earliest arrival time at the stop bar
- **self.scheduled_departure** – the scheduled arrival time at the stop bar

- **self.reschedule_departure** (*bool*) – True if a vehicle is open to receive a new departure time, False if want to keep previous trajectory
- **self.freshly_scheduled** (*bool*) – True if a vehicle is just scheduled a **different** departure and ready for being assigned a trajectory
- **self._times_sent_to_traj_planner** – number of times this vehicle is sent to trajectory planner

---

**Note:**

- By definition `scheduled_departure` is always greater than or equal to `earliest_arrival`.
- Prior to run, make sure teh specified size for trajectory array by `MAX_NUM_TRAJECTORY_POINTS` is enough to store all under the worst case.
- A vehicle may be open to be rescheduled but gets the same departure time and therefore `freshly_scheduled` should hold `False` under that case.
-

---

**reset_trj_points** (*sc*, *lane*, *time_threshold*, *file*)

Writes the trajectory points in the csv file if their time stamp is before the `time_threshold` and then removes them by updating the first trajectory point.

> **Warning:** Before calling this make sure at least the first trajectory point's time stamp is less than provided time threshold or such a call would be pointless.

**Parameters**

- **sc** – scenario number being simulated
- **lane** – lane number that is zero-based (it records it one-based)
- **time_threshold** – any trajectory point before this is considered expired (normally its simulation time)
- **file** – initialized in *Traffic.__init__()* method, if `None`, this does not record points in csv.

**set_earliest_departure** (*t_earliest*)

Sets the earliest arrival time at the stop bar. Called under *Traffic.update_vehicles_info()* method

**set_scheduled_departure** (*t_scheduled*, *d_scheduled*, *s_scheduled*, *lane*, *veh_indx*, *print_signal_detail*)

It only schedules if the new departure time is different and vehicle is far enough for trajectory assignment .. note:

```
- When a new vehicle is scheduled, it has two trajectory points: one for the␣
↪current state and the other for the final state.
- If the vehicle is closer than ``MIN_DIST_TO_STOP_BAR``, avoids appending␣
↪the schedule.
- Set the ``freshly_scheduled`` to True only if vehicle is getting a new␣
↪schedule and trajectory planning might become relevant.
```

**Parameters**

- **t_scheduled** – scheduled departure time ($s$)
- **d_scheduled** – scheduled departure distance ($m$)
- **s_scheduled** – scheduled departure speed ($m/s$)
- **lane** – the lane this vehicle is in (*for printing purpose only*)
- **veh_indx** – The index of this vehicle in ots lane (*for printing purpose only*)
- **print_signal_detail** – `True` if we want to print schedule

**set_poly**(*beta*, *t_ref*)
Sets the coefficients that define the polynomial that defines trajectory of a connected vehicle

**set_first_trj_point_indx**(*indx*)
Sets the fist column index that points to the trajectory start

**set_last_trj_point_indx**(*indx*)
Sets the last column index that points to the trajectory start

**static map_veh_type2str**(*code*)

**For the purpose of printing, this method translates the vehicle codes. Currently, it supports:**

- 0 : Conventional Vehicle (*CNV*)
- 1 : Connected and Automated Vehicle (*CAV*)

**Parameters code** (*int*) – numeric code for the vehicle type

**increment_times_sent_to_traj_planner**()
Increments the count on how many times sent to trajectory planner

**print_trj_points**(*lane*, *veh_indx*, *identifier*)
Print the first and last trajectory points information. This may be used either when a plan is scheduled or a trajectory is computed.

**Parameters**

- **lane** – zero-based lane number
- **veh_indx** – index to find the vehicle in its lane array
- **identifier** – use `*` for optimized trajectory, and `@` for scheduled departure

**class** src.intersection.**Traffic**(*inter_name*, *sc*, *log_at_vehicle_level*, *log_at_trj_point_level*, *print_commandline*, *start_time_stamp*)
Bases: `object`

**Objectives:**

- Adds new vehicles from the csv file to the `lanes.vehlist` structure
- Appends travel time, ID, and elapsed time columns and save csv
- Manages scenario indexing, resetting, and more
- Computes volumes in lanes
- removes/records served vehicles

**Note:**

- **The csv should be located under the `data/` directory with the valid name consistent to what inputted as an** argument and what exists in the data.py file.

- The scenario number should be appended to the name of intersection followed by an underscore.

---

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**__init__**(*inter_name*, *sc*, *log_at_vehicle_level*, *log_at_trj_point_level*, *print_commandline*, *start_time_stamp*)

**Objectives:**

- Sets the logging behaviour for outputting requested CSV files and auxiliary output vectors
- Imports the CSV file that includes the traffic and sorts it
- Initializes the first scenario number to run

**set_row_vehicle_level_csv**(*departure_time*, *veh*)
Sets the departure time of an individual vehicle that is just served.

**Parameters**

- **departure_time** – departure time in seconds
- **veh** (`Vehicle`) – vehicle to be recorder

**set_elapsed_sim_time**(*elapsed_t*)
Sets the elapsed time for one simulation of scenario.

**Parameters** **elapsed_t** – elapsed time in seconds

**save_veh_level_csv**(*inter_name*, *start_time_stamp*)
Set the recorded values and save the CSV at vehicle level.

**close_trj_csv**()
Closes trajectory CSV file.

**last_veh_arrived**()

**Returns** True if all vehicles from the input csv have been added at some point, False otherwise.

---

**Note:** The fact that all vehicles are *added* does not equal to all *served*. Thus, we check if any vehicle is in any of the incoming lanes before halting the program.

---

**get_first_detection_time**()

**Returns** The time when the first vehicle in current scenario shows up.

**update_vehicles_info**(*lanes*, *simulation_time*, *max_speed*, *min_headway*, *k*)

**Objectives**

- Appends arrived vehicles from the csv file to *Lanes*
- Assigns their earliest arrival time

**Parameters**

- **lanes** (`Lanes`) – vehicles are added to this data structure
- **simulation_time** – current simulation clock in seconds measured from zero
- **max_speed** – maximum allowable speed at the intersection in $m/s$

- **min_headway** – min headway in $sec/veh$
- **k** – one more than the degree of polynomial to compute trajectory of connected vehicles. We need it here to preallocate the vector that keeps the polynomial coefficients for connected vehicles.

**static get_volumes**(*lanes*, *num_lanes*, *det_range*)

Unit of volume in each lane is $veh/sec/lane$. Uses the fundamental traffic flow equation $F = D * S$.

**Parameters**

- **lanes** (Lanes) – includes all vehicles
- **num_lanes** – number of lanes
- **det_range** – detection range is needed to compute space-mean-speed

**Return volumes** array of volume level per lanes

**serve_update_at_stop_bar**(*lanes*, *simulation_time*, *num_lanes*, *print_commandline*)

This looks for/removes the served vehicles.

**Parameters**

- **lanes** (Lanes) – includes all vehicles
- **simulation_time** – current simulation clock
- **num_lanes** – number of lanes
- **print_commandline** –

src.intersection.**earliest_arrival_connected**(*det_time*, *speed*, *dist*, *amin*, *amax*, *max_speed*, *min_headway=0*, *t_earliest=0*)

**Uses the maximum of the followings to compute the earliest time vehicle can reach to the stop bar:**

- Accelerate/Decelerate to the maximum allowable speed and maintain the speed till departure
- Distance is short, it accelerates/decelerated to the best speed and departs
- Departs at the minimum headway with its lead vehicle (only for followers close enough to their lead)

**Parameters**

- **det_time** –
- **speed** –
- **dist** –
- **amin** –
- **amax** –
- **max_speed** –
- **min_headway** –
- **t_earliest** – earliest time of lead vehicle that is only needed if the vehicle is a follower vehicle

**Returns**

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

`src.intersection.`**`earliest_arrival_conventional`**(*det_time*, *speed*, *dist*, *min_headway=0*, *t_earliest=0*)

**Uses the maximum of the followings to compute the earliest time vehicle can reach to the stop bar:**

- Maintains the detected speed till departure
- Departs at the minimum headway with the vehicle in front

**Parameters**

- **`det_time`** –
- **`speed`** –
- **`dist`** –
- **`min_headway`** –
- **`t_earliest`** – earliest time of lead vehicle that is only needed if the vehicle is a follower vehicle

**Returns**

---

**Note:** Enter `min_headway` and `t_earliest` as zeros (default values), if a vehicle is the first in its lane.

---

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**class** `src.intersection.`**`TrajectoryPlanner`**(*max_speed*, *min_headway*, *k*, *m*)

Bases: `object`

**`__init__`**(*max_speed*, *min_headway*, *k*, *m*)

Instantiates the **trajectory** classes

**`plan_trajectory`**(*lanes*, *veh*, *lane*, *veh_indx*, *print_commandline*, *identifier*, *optional_packages_found*)

**Parameters**

- **`lanes`** (`Lanes`) –
- **`veh`** (`Vehicle`) –
- **`lane`** –
- **`veh_indx`** –
- **`print_commandline`** –
- **`identifier`** – Shows type of assigned trajectory
- **`optional_packages_found`** –

CHAPTER

# FOUR

# SIGNAL PHASE AND TIMING (SPAT)

**class** src.signal.**Signal**(*inter_name*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)

Bases: `object`

**The class serves the following goals:**

- Keeps the SPaT decision updated
- **Makes SPaT decisions through variety of control methods. For now it supports:**
  - Pre-timed control
  - Genetic Algorithm
  - Min Cost Flow model

Set the class variable `LAG` to the time (in seconds) that from start of green is not valid to schedule any departurs.

---

**Note:**

- `LAG` also is used in `Trajectory()` class. Set them consistent.
- `LARGE_NUM` is a large number to initialize badness of alternatives in GA. Make sure cannot be beaten by worst alternative.
- The signal status is saved under `\log\<intersection name>\` directory.

---

Use Case:

Instantiate like:

```
$ signal = GA_SPaT/Pretimed(.)
```

Perform SPaT computation by:

```
$ signal.solve(.)
```

**param LAG** the lag time from start of green when a vehicle can depart

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**LAG = 0.0**

**LARGE_NUM = 999999999**

**__init__**(*inter_name*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)

**Elements:**

- Sequence keeps the sequence of phases to be executed from 0
- `green_dur` keeps the amount of green allocated to each phase
- `yellow` and `all-red` is a fix amount at the end of all phases (look at class variables)
- start keeps the absolute time (in seconds) when each phase starts

**Note:** SPaT starts executing from index 0 to the end of each list.

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**_set_lane_lane_incidence**(*num_lanes*)

This converts a dictionary of the form: key is a lane and value is *set* of lanes that are in conflict with key (note numbering starts from 1 not 0) to `lane_lane_incidence` which includes the conflict matrix $|L| \times |L|$ where element $ij$ is 1 if $i$ and $j$ are conflicting movements

**Parameters num_lanes** –

**_set_phase_lane_incidence**(*num_lanes*)

Sets the phase-phase incidence matrix of the intersection

**Parameters num_lanes** –

**Returns**

**_append_extend_phase**(*phase*, *actual_green*)

Append a phase to the SPaT (append a phase and its green to the end of signal array) Note SPaT decision is the sequence and green duration of phases

**Parameters**

- **phase** – phase to be added
- **actual_green** – green duration of that phase

**set_critical_phase_volumes**(*volumes*)

Not used in GA since the phasing configuration is unknown prior to cycle length formula that is derived from time budget concept

**Warning:** Do not call this on a signal method that does not take `allowable_phases` as input

**Parameters volumes** –

**Returns**

**update_SPaT**(*time_threshold*, *sc*)

**Performs two tasks to update SPaT based on the given clock:**

- Removes terminated phase (happens when the all-red is passed)
- Checks for SPaT to not get empty after being updated

**Attention:**

- If all phases are getting purged, either make longer SPaT decisions or reduce the simulation steps.

**Parameters**

- **time_threshold** – Normally the current clock of simulation or real-time in $s$
- **sc** – scenario number to be recorded in CSV

**close_sig_csv**()

Closes the signal csv file

**_flush_upcoming_SPaTs**()

Just leaves the first SPaT and flushes the rest. One more severe variant to this is to even reduce the the green time of first phase.

**_do_base_SPaT**(*lanes*, *num_lanes*, *max_speed*, *trajectory_planner*)

This method aims to serve as many vehicles as possible given the available SPaT. Depending on the signal method, the set of current SPaT could be different. For example:

- If called by `Pretimed()` solver, the current SPaT may include multiple phases as a pretimed SPaT never gets flushed.
- If called by `GA_SPaT()` solver, since the SPaT gets flushed before calling. The goal is to serve as many vehicles with only the single current phase in SPaT.
- It plans trajectories if necessary.

**The condition to be served is to meet the following criteria:**

- Respect the minimum headway to the lead vehicle (if present)
- Respect the initiation of green plus a lag time specified by LAG as a class variable
- Respect the earliest available time at the stop bar controlled by the speed limit acc/dec rates
- Vehicle is allowed to acquire a new trajectory (`veh.reschedule_departure` holds True)

The method does not compute or return the badness metric since the it does not aim to change current phase and timing.

It may only gets called once per each Signal solve call prior to computation of the new SPaTs.

The schedule keeps the earliest departures at the stop bars of each lane and gets updated when a signal decision goes permanent. It is made by a dictionary of arrays (key is lane, value is sorted earliest departures).

`lanes.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `lanes.first_unsrvd_indx` will be used after this to avoid reserving and double-counting those already served with base SPaT. This also returns `any_unserved_vehicle` array that has True if any lane has vehicles that could not be unserved with base SPaT.

**Note:**

- Since base SPaT never gets changed (for safety and practical reasons), any vehicle served by it has to get `reschedule_departure` value set to `False`.

- It is feasible that if fusion algorithm updates the info on this vehicle and wants an update on trajectory, it rolls back the `reschedule_departure` to be `True`. However, this should be decided outside this method.
- The reason that this does not return schedule of departures is because they are already set inside this method. Late, the set method skips these.
- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.
- all-red from the end and `LAG` time from the beginning of a phase are note utilizes by any vehicle.
- The `veh.reschedule_departure` is set to False for vehicles that get schedules here, however if decided a vehcile needs to be rescheduled, make it True wherever that decision is being made.

---

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –
- **max_speed** –
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**Returns** The `lanes.first_unsrvd_indx` array that keeps index off the first unserved vehicle in each lane, is initialized to zero before calling this method and gets updated by the end of this call. It also returns `served_vehicle_time` that shows the schedule

**_do_non_base_SPaT**(*lanes*, *num_lanes*, *first_unsrvd_indx*, *served_vehicle_time*, *any_unserved_vehicle*)

Most of times the base SPaT prior to running a `solve()` method does not serve all vehicles. However, vehicles require trajectory to be provided. One way to address this is to assign them the best temporal trajectory which only has some of general qualities necessary for continuation of program. In this method we do the followings to compute the `departure times` of such trajectories:

- Without use of phases, schedule vehicles one after the other at minimum headway restricted by the saturation headway. This gives an overestimate of teh departure time since one vehicle gets served by intersection at a time, while having allowing to depart in phases let multiple simultaneous departures.
- This may be called after a signal `solve()` method decided to complete those that did not get served.
- Also this assumes min headway after green starts instead of `LAG` time which is a simplification.
- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.

**Warning:**

- Since the departure times are definitely temporal, DO NOT set `reschedule_departure` to `False`.
- The `lanes.first_unsrvd_indx` cannot be used since it does not keep GA newly served vehicles. However, it would work for pretimed since the method is static.

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –

- **first_unsrvd_indx** – keeps the index of first unserved vehicle in lanes.
- **served_vehicle_time** – includes schedule of departures for those served by base SPaT
- **any_unserved_vehicle** – *Has 'False'* for the lane that has all vehicles scheduled through base SPaT and the `solve()`, `True` otherwise.

**Returns** `served_vehicle_time` that now includes the schedules of all vehicle except those served through base SPaT

**_set_non_base_scheduled_departures** (*lanes*, *scheduled_departure*, *num_lanes*, *max_speed*, *trajectory_planner*)

Sets the scheduled departure in the trajectory of the vehicle and plans trajectory of vehicle

**Note:**

- Departure schedule of those which were served by base SPaT is set in `base_badness()` and not here.

**Parameters**

- **lanes** (`Lanes`) –
- **scheduled_departure** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in $m/s$
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**class** src.signal.**Pretimed**(*inter_name*, *first_detection_time*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)

Bases: *`src.signal.Signal`*

**Note:**

**Assumptions:**

- The sequence and duration are pre-determined
- **Cycle length is computed using the time budget concept in traffic flow theory**
  - min and max of 60 and 120 seconds bound the *cycle length*

**Warning:** Must choose `NUM_CYCLES` at least 2.

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**NUM_CYCLES = 2**

**`__init__`**(*inter_name*, *first_detection_time*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)
Initialize the pretimed SPaT

**`solve`**(*lanes*, *num_lanes*, *max_speed*, *critical_volume_ratio*, *trajectory_planner*)

**The phases sequence is exactly as the provided in `data.py`. The flow is:**

1. First serves using the available SPaT
2. This simply adds a cycle to SPaT if a cycle is terminated
3. Serves unserved vehicles, if any present
4. Next it provides the departure schedule

**Note:** The `scheduled_departures` is made only to call `complete_unserved_vehicles()`. It only stores departures for those vehicles nit served bt base SPaT.

**Parameters**

- **`lanes`** (`Lanes`) –
- **`num_lanes`** –
- **`max_speed`** –

**`LAG = 0.0`**

**`LARGE_NUM = 999999999`**

**`_append_extend_phase`**(*phase*, *actual_green*)
Append a phase to the SPaT (append a phase and its green to the end of signal array) Note SPaT decision is the sequence and green duration of phases

**Parameters**

- **`phase`** – phase to be added
- **`actual_green`** – green duration of that phase

**`_do_base_SPaT`**(*lanes*, *num_lanes*, *max_speed*, *trajectory_planner*)
This method aims to serve as many vehicles as possible given the available SPaT. Depending on the signal method, the set of current SPaT could be different. For example:

- If called by *`Pretimed()`* solver, the current SPaT may include multiple phases as a pretimed SPaT never gets flushed.
- If called by *`GA_SPaT()`* solver, since the SPaT gets flushed before calling. The goal is to serve as many vehicles with only the single current phase in SPaT.
- It plans trajectories if necessary.

**The condition to be served is to meet the following criteria:**

- Respect the minimum headway to the lead vehicle (if present)
- Respect the initiation of green plus a lag time specified by LAG as a class variable
- Respect the earliest available time at the stop bar controlled by the speed limit acc/dec rates
- Vehicle is allowed to acquire a new trajectory (`veh.reschedule_departure` holds True)

The method does not compute or return the badness metric since the it does not aim to change current phase and timing.

It may only gets called once per each Signal solve call prior to computation of the new SPaTs.

The schedule keeps the earliest departures at the stop bars of each lane and gets updated when a signal decision goes permanent. It is made by a dictionary of arrays (key is lane, value is sorted earliest departures).

`lanes.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `lanes.first_unsrvd_indx` will be used after this to avoid reserving and double-counting those already served with base SPaT. This also returns `any_unserved_vehicle` array that has True if any lane has vehicles that could not be unserved with base SPaT.

---

**Note:**

- Since base SPaT never gets changed (for safety and practical reasons), any vehicle served by it has to get `reschedule_departure` value set to `False`.
- It is feasible that if fusion algorithm updates the info on this vehicle and wants an update on trajectory, it rolls back the `reschedule_departure` to be `True`. However, this should be decided outside this method.
- The reason that this does not return schedule of departures is because they are already set inside this method. Late, the set method skips these.
- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.
- all-red from the end and `LAG` time from the beginning of a phase are note utilizes by any vehicle.
- The `veh.reschedule_departure` is set to False for vehicles that get schedules here, however if decided a vehcile needs to be rescheduled, make it True wherever that decision is being made.

---

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –
- **max_speed** –
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**Returns** The `lanes.first_unsrvd_indx` array that keeps index off the first unserved vehicle in each lane, is initialized to zero before calling this method and gets updated by the end of this call. It also returns `served_vehicle_time` that shows the schedule

**_do_non_base_SPaT**(*lanes*, *num_lanes*, *first_unsrvd_indx*, *served_vehicle_time*, *any_unserved_vehicle*)

Most of times the base SPaT prior to running a `solve()` method does not serve all vehicles. However, vehicles require trajectory to be provided. One way to address this is to assign them the best temporal trajectory which only has some of general qualities necessary for continuation of program. In this method we do the followings to compute the `departure times` of such trajectories:

- Without use of phases, schedule vehicles one after the other at minimum headway restricted by the saturation headway. This gives an overestimate of teh departure time since one vehicle gets served by intersection at a time, while having allowing to depart in phases let multiple simultaneous departures.
- This may be called after a signal `solve()` method decided to complete those that did not get served.
- Also this assumes min headway after green starts instead of `LAG` time which is a simplification.

- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.

> **Warning:**
>
> - Since the departure times are definitely temporal, DO NOT set `reschedule_departure` to `False`.
> - The `lanes.first_unsrvd_indx` cannot be used since it does not keep GA newly served vehicles. However, it would work for pretimed since the method is static.

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –
- **first_unsrvd_indx** – keeps the index of first unserved vehicle in lanes.
- **served_vehicle_time** – includes schedule of departures for those served by base SPaT
- **any_unserved_vehicle** – *Has 'False'* for the lane that has all vehicles scheduled through base SPaT and the `solve()`, `True` otherwise.

**Returns** `served_vehicle_time` that now includes the schedules of all vehicle except those served through base SPaT

**_flush_upcoming_SPaTs**()

Just leaves the first SPaT and flushes the rest. One more severe variant to this is to even reduce the the green time of first phase.

**_set_lane_lane_incidence**(*num_lanes*)

This converts a dictionary of the form: key is a lane and value is *set* of lanes that are in conflict with key (note numbering starts from 1 not 0) to `lane_lane_incidence` which includes the conflict matrix $|L| \times |L|$ where element $ij$ is 1 if $i$ and $j$ are conflicting movements

**Parameters** **num_lanes** –

**_set_non_base_scheduled_departures**(*lanes*, *scheduled_departure*, *num_lanes*, *max_speed*, *trajectory_planner*)

Sets the scheduled departure in the trajectory of the vehicle and plans trajectory of vehicle

> **Note:**
>
> - Departure schedule of those which were served by base SPaT is set in `base_badness()` and not here.

**Parameters**

- **lanes** (`Lanes`) –
- **scheduled_departure** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in $m/s$
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**_set_phase_lane_incidence**(*num_lanes*)
Sets the phase-phase incidence matrix of the intersection

**Parameters** **num_lanes** –

**Returns**

**close_sig_csv**()
Closes the signal csv file

**set_critical_phase_volumes**(*volumes*)
Not used in GA since the phasing configuration is unknown prior to cycle length formula that is derived from time budget concept

> **Warning:** Do not call this on a signal method that does not take `allowable_phases` as input

**Parameters** **volumes** –

**Returns**

**update_SPaT**(*time_threshold*, *sc*)

**Performs two tasks to update SPaT based on the given clock:**

- Removes terminated phase (happens when the all-red is passed)
- Checks for SPaT to not get empty after being updated

> **Attention:**
>
> - If all phases are getting purged, either make longer SPaT decisions or reduce the simulation steps.

**Parameters**

- **time_threshold** – Normally the current clock of simulation or real-time in *s*
- **sc** – scenario number to be recorded in CSV

**class** src.signal.**GA_SPaT**(*inter_name*, *allowable_phases*, *first_detection_time*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)
Bases: *src.signal.Signal*

Under this class, the *SPaT* is decided optimally by a *GA*.

**Parameters**

- **allowable_phases** – subset of all possible phases to be used.
- **MAX_PHASE_LENGTH** – do not include more than this in a phase sequence (is exclusive of the last: 1,2, . . . , `MAX_PHASE_LENGTH`-1)
- **POPULATION_SIZE** – this is the maximum size of individuals per iteration of *GA*
- **CROSSOVER_SIZE** – this specifies how many of the individuals from `POPULATION_SIZE` to be computed using crossover..
- **LAMBDA** – The weight factor to convert average travel time to throughput and give the *badness* of an individual.

- **BADNESS_ACCURACY** – 10 raised to the number of digits we want to keep when hashing the *badness* of an individual

**Warning:**

- `allowable_phases` **must** cover all lanes or some would not get green at all.
- `allowable_phases` **must** be zero-based unlike what is provided in `data.py`

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**MAX_PHASE_LENGTH = 4**

**POPULATION_SIZE = 20**

**MAX_ITERATION_PER_PHASE = 10**

**CROSSOVER_SIZE = 10**

**LAMBDA = 0.002**

**BADNESS_ACCURACY = 100**

**__init__**(*inter_name*, *allowable_phases*, *first_detection_time*, *num_lanes*, *min_headway*, *log_signal_status*, *sc*, *start_time_stamp*, *do_traj_computation*, *print_commandline*, *optional_packages_found*)

**Parameters**

- **inter_name** –
- **allowable_phases** (`tuple`) – zero-based subset of phases
- **first_detection_time** –
- **num_lanes** –
- **min_headway** –
- **log_signal_status** –
- **sc** –
- **start_time_stamp** –
- **do_traj_computation** –
- **print_commandline** –
- **optional_packages_found** –

**solve**(*lanes*, *num_lanes*, *max_speed*, *critical_volume_ratio*, *trajectory_planner*)

**This method implements Genetic Algorithm to determine *SPaT*. The high-level work flow is as the following:**

1. From the available *SPaT*, only keep the ongoing one due to safety and practical reasons (*Here we do not change the timing of the first phase, however a variant is to reduce the timing to the minimum green time*).
2. Serve as many as possible with the remaining phase.
3. If any unserved vehicle is present, do *GA*.

**Attention:**

- We define *badness* (the opposite of fitness) as the measure that less of it is preferred for choosing a SPaT.
- GA has access to only the given subset of phases provided by `allowable_phases` from the full set in `data.py` file.
- If an alternative beats the best known SPaT, it takes the `best_SPaT` spot inside the `evaluate_badness()` call.
- GA tries cycles with 1 up to the defined number of phases and for each it computes the cycle length using the time budget concept in traffic flow theory.
- GA keeps the alternative in a sorted dictionary that the key is `badness` and the value keeps the corresponding SPaT decision. This helps when we want to replace worse individuals with new ones from crossover.
- The phase sequence are randomly drawn from the set of phases **without** replacement.
- The timings are random but respects the minimum and maximum green. They also sum to the cycle length.
- Note since the dictionary hashes individuals based on their `badness`, it may overwrite one individual with anther. Hence the population may fall less than what defined initially.
- The crossover step is in-place, meaning it replaces the individuals with higher badness with crossovered ones. This way elite selection step is implemented at the same time crossover executes.
- Eventually, the best SPaT may not serve all vehicles. In that case, `_schedule_unserved_vehicles()` method gets called to provide temporary schedule for the unserved vehicles.

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –
- **max_speed** –
- **critical_volume_ratio** –
- **trajectory_planner** (`TrajectoryPlanner`) –

**_evaluate_badness** (*phase_seq*, *time_split*, *lanes*, *num_lanes*)

**This method computes the badness (opposite if fitness) of an alternative using the equation $\lambda \times t - c$, where:**

- $c$ is the count of served vehicles in $veh$
- $\lambda$ is weight factor in $veh/s$
- $t$ is the average travel time in $s$, under the given *SPaT*.

**Attention:**

- A rough approximate for $\lambda$ is the inverse of the detection range.

- Here we do not account for the vehicles served with base *SPaT* as they are already served.
- We create a copy of `first_unsrvd_indx` since there is no guarantee this *SPaT* is the best by the end of *GA*.
- The vehicle to be served by this method should have had `veh.reschedule_departure` set to `True`.
- An individual which has `throughput` of zero is not qualified for comparison to best known *SPaT*.
- Please note base on the provided definition *badness* can acquire negative values.

**Parameters**

- **`phase_seq`** –
- **`time_split`** –
- **`lanes`** (`Lanes`) – holds the traffic intended to be served
- **`num_lanes`** –

**Returns** The corresponding *badness* for given SPaT defined by `phase_seq` and `time_split` to be added to the population. It also sets, if qualified, this individual as the best known so far.

**`_get_optimal_cycle_length`**(*critical_volume_ratio*, *phase_length*)

Uses the time budget concept from traffic flow theory to compute the cycle length $C = \frac{n \times ar}{1 - V_{cr}}$.

**See also:**

Refer to HCM 2010 for more details.

**Parameters**

- **`critical_volume_ratio`** –
- **`phase_length`** –

**Returns**

**`_mutate_seq`**(*phase_length*)

Generates a randomized sequence from the provided subset of allowable phases.

**Parameters** **`phase_length`** –

**Returns** seq

**`_mutate_timing`**(*cycle_length*, *phase_length*)

Creates the random phase split. A valid timing should respect the min/max green requirement unless it conflicts with the cycle length requirement which in that case we should adjust the maximum green to avoid the slack in time.

**Note:** A phase timing should be between $g_{min} + y + ar$ and $g_{max} + y + ar$

**Parameters**

- **`cycle_length`** –

- **phase_length** –

**Returns** time_split

**_cross_over** (*left_parent*, *right_parent*, *phase_length*, *half_max_indx*)
Performs the crossover operation in GA.

**Parameters**

- **left_parent** –
- **right_parent** –
- **phase_length** –
- **half_max_indx** –

**Returns** child with valid SPaT inherited from provided parents.

**LAG = 0.0**

**LARGE_NUM = 999999999**

**_append_extend_phase** (*phase*, *actual_green*)
Append a phase to the SPaT (append a phase and its green to the end of signal array) Note SPaT decision is the sequence and green duration of phases

**Parameters**

- **phase** – phase to be added
- **actual_green** – green duration of that phase

**_do_base_SPaT** (*lanes*, *num_lanes*, *max_speed*, *trajectory_planner*)
This method aims to serve as many vehicles as possible given the available SPaT. Depending on the signal method, the set of current SPaT could be different. For example:

- If called by *Pretimed()* solver, the current SPaT may include multiple phases as a pretimed SPaT never gets flushed.
- If called by *GA_SPaT()* solver, since the SPaT gets flushed before calling. The goal is to serve as many vehicles with only the single current phase in SPaT.
- It plans trajectories if necessary.

**The condition to be served is to meet the following criteria:**

- Respect the minimum headway to the lead vehicle (if present)
- Respect the initiation of green plus a lag time specified by LAG as a class variable
- Respect the earliest available time at the stop bar controlled by the speed limit acc/dec rates
- Vehicle is allowed to acquire a new trajectory (`veh.reschedule_departure` holds True)

The method does not compute or return the badness metric since the it does not aim to change current phase and timing.

It may only gets called once per each Signal solve call prior to computation of the new SPaTs.

The schedule keeps the earliest departures at the stop bars of each lane and gets updated when a signal decision goes permanent. It is made by a dictionary of arrays (key is lane, value is sorted earliest departures).

`lanes.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `lanes.first_unsrvd_indx` will be used after this to avoid reserving and double-counting those already served with base SPaT. This also returns `any_unserved_vehicle` array that has True if any lane has vehicles that could not be unserved with base SPaT.

---

**Note:**

- Since base SPaT never gets changed (for safety and practical reasons), any vehicle served by it has to get `reschedule_departure` value set to `False`.
- It is feasible that if fusion algorithm updates the info on this vehicle and wants an update on trajectory, it rolls back the `reschedule_departure` to be `True`. However, this should be decided outside this method.
- The reason that this does not return schedule of departures is because they are already set inside this method. Late, the set method skips these.
- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.
- all-red from the end and `LAG` time from the beginning of a phase are note utilizes by any vehicle.
- The `veh.reschedule_departure` is set to False for vehicles that get schedules here, however if decided a vehcile needs to be rescheduled, make it True wherever that decision is being made.

---

**Parameters**

- **lanes** (`Lanes`) –
- **num_lanes** –
- **max_speed** –
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**Returns** The `lanes.first_unsrvd_indx` array that keeps index off the first unserved vehicle in each lane, is initialized to zero before calling this method and gets updated by the end of this call. It also returns `served_vehicle_time` that shows the schedule

**_do_non_base_SPaT**(*lanes*, *num_lanes*, *first_unsrvd_indx*, *served_vehicle_time*, *any_unserved_vehicle*)

Most of times the base SPaT prior to running a `solve()` method does not serve all vehicles. However, vehicles require trajectory to be provided. One way to address this is to assign them the best temporal trajectory which only has some of general qualities necessary for continuation of program. In this method we do the followings to compute the `departure times` of such trajectories:

- Without use of phases, schedule vehicles one after the other at minimum headway restricted by the saturation headway. This gives an overestimate of teh departure time since one vehicle gets served by intersection at a time, while having allowing to depart in phases let multiple simultaneous departures.
- This may be called after a signal `solve()` method decided to complete those that did not get served.
- Also this assumes min headway after green starts instead of `LAG` time which is a simplification.
- If a vehicle gets a schedule and has more than one trajectory point, the last index should reset to the first index so when the trajectory is set there would be two points.

**Warning:**

- Since the departure times are definitely temporal, DO NOT set `reschedule_departure` to `False`.

- The `lanes.first_unsrvd_indx` cannot be used since it does not keep GA newly served vehicles. However, it would work for pretimed since the method is static.

**Parameters**

- **lanes** (Lanes) –
- **num_lanes** –
- **first_unsrvd_indx** – keeps the index of first unserved vehicle in lanes.
- **served_vehicle_time** – includes schedule of departures for those served by base SPaT
- **any_unserved_vehicle** – *Has 'False'* for the lane that has all vehicles scheduled through base SPaT and the `solve()`, `True` otherwise.

**Returns** `served_vehicle_time` that now includes the schedules of all vehicle except those served through base SPaT

**_flush_upcoming_SPaTs**()

Just leaves the first SPaT and flushes the rest. One more severe variant to this is to even reduce the the green time of first phase.

**_set_lane_lane_incidence**(*num_lanes*)

This converts a dictionary of the form: key is a lane and value is *set* of lanes that are in conflict with key (note numbering starts from 1 not 0) to `lane_lane_incidence` which includes the conflict matrix $|L| \times |L|$ where element $ij$ is 1 if $i$ and $j$ are conflicting movements

**Parameters** **num_lanes** –

**_set_non_base_scheduled_departures**(*lanes*, *scheduled_departure*, *num_lanes*, *max_speed*, *trajectory_planner*)

Sets the scheduled departure in the trajectory of the vehicle and plans trajectory of vehicle

**Note:**

- Departure schedule of those which were served by base SPaT is set in `base_badness()` and not here.

**Parameters**

- **lanes** (Lanes) –
- **scheduled_departure** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in $m/s$
- **trajectory_planner** (`src.intersection.TrajectoryPlanner`) –

**_set_phase_lane_incidence**(*num_lanes*)

Sets the phase-phase incidence matrix of the intersection

**Parameters** **num_lanes** –

**Returns**

**close_sig_csv**()
Closes the signal csv file

**set_critical_phase_volumes**(*volumes*)
Not used in GA since the phasing configuration is unknown prior to cycle length formula that is derived from time budget concept

> **Warning:** Do not call this on a signal method that does not take `allowable_phases` as input

**Parameters** **volumes** –

**Returns**

**update_SPaT**(*time_threshold*, *sc*)

**Performs two tasks to update SPaT based on the given clock:**

- Removes terminated phase (happens when the all-red is passed)
- Checks for SPaT to not get empty after being updated

> **Attention:**
>
> - If all phases are getting purged, either make longer SPaT decisions or reduce the simulation steps.

**Parameters**

- **time_threshold** – Normally the current clock of simulation or real-time in $s$
- **sc** – scenario number to be recorded in CSV

CHAPTER

FIVE

# TRAJECTORY

**class** `src.trajectory.Trajectory`(*max_speed*, *min_headway*)

**Is the abstract class for computing the trajectory points. Four subclasses inherited from this parent class:**

- *LeadConventional*
- *FollowerConnected*
- *LeadConnected*
- *FollowerConventional*

Any solve method under each class shall invoke *set_trajectory* method at the end or does the assignment in-place.

---

**Note:** If want to limit the trajectory planning, there are two options: - If a particular vehicle is intended to be skipped, simply set `vehicle.reschedule_departure` to `False` - If the whole simulation is intended to be run without trajectory planer, set `vehicle.reschedule_departure` in `main.py` to False.

---

**Parameters**

- **RES** – time difference between two consecutive trajectory points in seconds used in *discretize_time_interval()* (be careful not to exceed max size of trajectory)
- **EPS** – small number that lower than that is approximated by zero

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**discretize_time_interval**(*start_time*, *end_time*)

Discretize the given time interval to a numpy array of time stamps

**static set_trajectory**(*veh*, *t*, *d*, *s*)

Sets trajectory of the vehicle and updates the first and last trajectory point index.

---

**Note:** An assigned trajectory always is indexed from zero as the `veh.set_first_trj_point_indx`.

---

**Parameters**

- **veh** (`Vehicle`) – the vehicle object that is owns the trajectory

- **t** – time stamps (seconds from the reference time)
- **d** – distances at each time stamp (in meters from the stop bar)
- **s** – speed at each time stamp (in $m/s$)

**class** src.trajectory.**LeadConventional**(*max_speed*, *min_headway*)

Computes the trajectory for a lead conventional vehicle assuming the vehicle tends to maintain its arrival speed.

Use Case:

Instantiate like:

```
$ lead_conventional_trj_estimator = LeadConventional(.)
```

Perform trajectory computation by:

```
$ lead_conventional_trj_estimator.solve(veh)
```

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**solve**(*veh*)

Constructs the trajectory of a lead conventional vehicle assuming the driver maintains its speed

**Parameters** **veh** (Vehicle) – the lead conventional vehicle

**class** src.trajectory.**FollowerConventional**(*max_speed*, *min_headway*)

Estimates the trajectory for a follower conventional vehicle assuming a car following model. In the current implementation, Gipps car-following model[1] is used.

Use Case:

Instantiate like:

```
$ follower_conventional_trj_estimator = FollowerConventional(.)
```

Perform trajectory computation by:

```
$ follower_conventional_trj_estimator.solve(veh, .)
```

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**solve**(*veh*, *lead_veh*)

Gipps car following model is implemented. It is written in-place (does not call *set_trajectory*)

**Note:**

- The only trajectory point index that changes is follower's last one.
- This method relies on the fact that lead vehicle's first trajectory point is current.
- Assumed the gap to lead vehicle cannot get lower than half length of the lead vehicle.

[1] Gipps, Peter G. *A behavioural car-following model for computer simulation*. Transportation Research Part B: Methodological 15.2 (1981): 105-111 (link).

$$v_{n_l}(t+\Delta t) = \min\left\{v_{n_l}(t) + 2.5a_{n_l}^{acc} \times \Delta t \times (1 - \frac{v_{n_l}(t)}{V_{n_l}^{des}}) \times \sqrt{0.025 + \frac{v_{n_l}(t)}{V_{n_l}^{des}}},\right.$$
$$\left. a_{n_l}^{dec} \times \Delta t + \sqrt{a_{n_l}^{dec} \times \left(2 \times \left(d_{(n-1)_l}(t) - d_{n_l}(t) + L_{n_l}\right) + \Delta t \times \left(a_{n_l}^{dec} \times \Delta t + v_{n_l}(t)\right) + \frac{v_{(n-1)_l}(t)^2}{a_{n_l}^{dec}}\right)}\right\}$$

Fig. 1: Gipps car following formula.

**Parameters**

- **veh** (Vehicle) – The follower conventional vehicle
- **lead_veh** (Vehicle) – The vehicle in front of subject conventional vehicle

**class** src.trajectory.**LeadConnected**(*max_speed*, *min_headway*, *k*, *m*)

> **Attention:**
>
> - Trajectory function: $f(t) = \sum_{n=0}^{k-1} \beta_n \times t^n$
> - Negative of speed profile: $f'(t) = \sum_{n=1}^{k-1} n \times \beta_n \times t^{n-1}$
> - Negative of acceleration profile: $f''(t) = \sum_{n=2}^{k-1} n \times (n-1) \times \beta_n \times t^{n-2}$

**param NUM_DIGS** The accuracy to keep decimals

**param SPEED_DECREMENT_SIZE** The final speed decrements from maximum to 0 by stepsize defined by maximum speed devided by this

Use Case:

Instantiate like:

```
$ lead_connected_trj_optimizer = LeadConnected(.)
```

Perform trajectory computation by:

```
$ lead_conventional_trj_estimator.solve(veh)
```

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**set_model**(*veh*, *is_lead=False*)

Overrides the generic coefficients to build the specific *LP* model for the AV trajectory.

This model solves an *LP* model to compute trajectory of AVs.

**Parameters**

- **veh** (Vehicle) – vehicle object that its trajectory is meant to be computed
- **is_lead** –

**Returns** cplex LP model. Should return the model since the follower optimizer adds constraints to this model

```
Minimize
 obj: 28.8338319973405 beta_0 + 415.694933825427 beta_1
      + 7990.71858937861 beta_2 + 172802.278008126 beta_3
      + 3986041.48227522 beta_4 + 95777375.3619614 beta_5
      + 2367110357.4283 beta_6 + 59721254569.5955 beta_7
      + 1530660107493.44 beta_8 + 39721316746047.2 beta_9
Subject To
 match_det_dist:  beta_0  = 499.871984004097
 match_det_speed: beta_1  = -8.09141058767781
 match_dep_dist:  beta_0 + 28.8338319973405 beta_1 + 831.389867650854 beta_2
                  + 23972.1557681358 beta_3 + 691209.112032505 beta_4
                  + 19930207.4113761 beta_5 + 574664252.171769 beta_6
                  + 16569772501.9981 beta_7 + 477770036556.764 beta_8
                  + 13775940967440.9 beta_9  = 0
 match_dep_speed: beta_1 + 57.6676639946809 beta_2 + 2494.16960295256 beta_3
                  + 95888.6230725434 beta_4 + 3456045.56016252 beta_5
                  + 119581244.468257 beta_6 + 4022649765.20238 beta_7
                  + 132558180015.985 beta_8 + 4299930329010.87 beta_9
                   = -17.8816
 ub_speed_0:      beta_1 + 2.88338319973405 beta_2 + 6.23542400738141 beta_3
                  + 11.9860778840679 beta_4 + 21.6002847510158 beta_5
                  + 37.3691388963302 beta_6 + 62.8539025812872 beta_7
                  + 103.561078137488 beta_8 + 167.966028476987 beta_9
                  >= -17.8816
 ub_speed_1:      beta_1 + 5.62259723948139 beta_2 + 23.7101997880678 beta_3
                  + 88.8752692506281 beta_4 + 312.318652216717 beta_5
                  + 1053.62519507536 beta_6 + 3455.73089941259 beta_7
                  + 11102.9617231016 beta_8 + 35115.4585880879 beta_9
                  >= -17.8816
 ub_speed_2:      beta_1 + 8.36181127922873 beta_2 + 52.4399159020776 beta_3
                  + 292.328453514533 beta_4 + 1527.74709989832 beta_5
                  + 7664.83975904325 beta_6 + 37386.9670712117 beta_7
                  + 178641.578829835 beta_8 + 840244.032449181 beta_9
                  >= -17.8816
 ub_speed_3:      beta_1 + 11.1010253189761 beta_2 + 92.4245723494109 beta_3
                  + 684.005011830897 beta_4 + 4745.72309665083 beta_5
                  + 31609.4353516622 beta_6 + 204689.999591781 beta_7
                  + 1298439.35314832 beta_8 + 8107879.57563064 beta_9
                  >= -17.8816
 ub_speed_4:      beta_1 + 13.8402393587234 beta_2 + 143.664169130068 beta_3
                  + 1325.56432535484 beta_4 + 11466.3297176848 beta_5
                  + 95218.0487152807 beta_6 + 768740.341535875 beta_7
                  + 6079743.0466076 beta_8 + 47331618.1900783 beta_9
                  >= -17.8816
 ub_speed_5:      beta_1 + 16.5794533984708 beta_2 + 206.158706244048 beta_3
                  + 2278.66577524147 beta_4 + 23611.8956445664 beta_5
                  + 234883.394093186 beta_6 + 2271639.00042486 beta_7
                  + 21521447.3975387 beta_8 + 200707781.734762 beta_9
                  >= -17.8816
 ub_speed_6:      beta_1 + 19.3186674382181 beta_2 + 279.908183691351 beta_3
                  + 3604.96874264592 beta_4 + 43526.9951652174 beta_5
                  + 504530.126509057 beta_6 + 5685662.34051117 beta_7
                  + 62765382.8127635 beta_8 + 682055750.920628 beta_9
                  >= -17.8816
 ub_speed_7:      beta_1 + 22.0578814779654 beta_2 + 364.912601471978 beta_3
                  + 5366.13260872329 beta_4 + 73978.4481739149 beta_5
                  + 979084.705046415 beta_6 + 12597978.3888015 beta_7
                  + 158791265.235516 beta_8 + 1970211885.91942 beta_9
                  >= -17.8816
 ub_speed_8:      beta_1 + 24.7970955177128 beta_2 + 461.171959585929 beta_3
                  + 7623.8167546287 beta_4 + 118155.320171292 beta_5
                  + 1757945.25612808 beta_6 + 25428629.5848188 beta_7
                  + 360317803.828166 beta_8 + 5025844686.52098 beta_9
                  >= -17.8816
 ub_speed_9:      beta_1 + 27.5363095574601 beta_2 + 568.686258033203 beta_3
                  + 10439.6805615173 beta_4 + 179668.922264337 beta_5
                  + 2968451.43679561 beta_6 + 47681781.9741032 beta_7
                  + 750274462.108702 beta_8 + 11621131785.9462 beta_9
                  >= -17.8816
 ub_speed_10:     beta_1 + 30.2755235972075 beta_2 + 687.4554968138 beta_3
                  + 13875.3834105441 beta_4 + 262552.811166394 beta_5
                  + 4769354.29798878 beta_6 + 84230240.8454513 beta_7
```

Fig. 2: Part of a sample CPLEX model.

**solve** (*veh*, *model*, *max_speed*)

Solves an *LP* model for connected vehicle (both lead and follower)

**Parameters**

- **veh** (Vehicle) –
- **model** (*cplex*) –
- **max_speed** –

**Returns** coefficients of the polynomial for the veh object and trajectory points to the trajectory attribute of it

**class** src.trajectory.**FollowerConnected** (*max_speed*, *min_headway*, *k*, *m*)

Optimizes the trajectory of a follower CAV.

Use Case:

Instantiate like:

```
$ follower_connected_trj_optimizer = FollowerConnected(.)
```

Perform trajectory computation by:

```
$ model = follower_connected_trj_optimizer.set_model(.)
$ follower_connected_trj_optimizer.solve(veh, .)
```

**param GAP_CTRL_STARTS** This is the relative time when gap control constraints get added

**param SAFE_MIN_GAP** The minimum safe distance to keep from lead vehicles (in $m$) [*can be speed dependent*]

**Author** Mahmoud Pourmehrab <pourmehrab@gmail.com>

**Date** April-2018

**set_model** (*veh*, *lead_veh*)

**Sets the LP model using the extra constraints to enforce the safe headway. Either three cases happen here:**

- **The lead is a *CNV* and its trajectory overlaps and it has enough trajectory points**
  - Enforce the constraint on last m trajectory points of the lead vehicle
- **The lead is a *CAV* and its trajectory overlaps**
  - Evaluate the polynomial over m points as defined in the paper
- **Otherwise**
  - Relax the constraints

**Parameters**

- **veh** (Vehicle) – follower connected vehicle that the trajectory model is constructed for
- **lead_veh** (Vehicle) – the vehicle in front

**Returns** the cplex LP model to be solved by solve() method

CHAPTER

SIX

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## d



## m



## s

# INDEX

## Symbols



## A



## B



## C



## D



## E

## F



## G



## I



## L



## M



## N



## P



## R



## S

# T



# U



# V