AVIAN Documentation

Release 1.0.0

Mahmoud Pourmehrab

May 28, 2018

CONTENTS

CHAPTER

ONE

OVERVIEW

1.1 Introduction

The goal of this program is to simulate the performance of an isolated intersection with traffic from autonomous and conventional vehicles using a 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]). The methodology to use Genetic Algorithm (*GA*) and polynomial *trajectory* optimization is explained in-depth in[2]. The Min Cost Flow (*MCF*) signal optimization is elaborated in[3]. For comments and questions please email me at pourmehrab@gmail.com. For details on the AVIAN project visit AVIAN.ESSIE.UFL.EDU.

Note:

- The interpreter version requirement is set to `3.6.0`. 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)

[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).

[2] Pourmehrab, M., Elefteriadou, L., Ranka, S. (2018). *Dynamic Intersection Manager under Traffic of CAVs and Conventional Vehicles Using Hybrid Genetic Algorithm and Linear Programming*.

[3] Pourmehrab, M., Elefteriadou, L., Ranka, S. (2018). *Real-time Optimizing a Hyper-Intersection Performance under Traffic of Automated Vehicles and Conventional Vehicles*.

- **Outputs are stored under /log/<intersection name>/ (create the directories if they do not exist):**
  - The outputs are stored in CSV format and named by the format of <local timestamp>_<sc>_<content identifier>.csv (time is in mm-dd-yyyy_hh:mm:ss format).
  - <local timestamp>_<sc>_vehicle_level.csv includes input CSV plus the departure time, vehicle ID and elapsed time columns
  - <local timestamp>_<sc>_trj_point_level.csv includes the trajectory points
  - <local timestamp>_<sc>_sig_phase_level.csv includes signalization details
- 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 <local 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
  - *L*: length of vehicle
  - *maxAcc, maxDec*: maximum acceleration, deceleration rate vehicle can execute
  - *dest*: destination {0: right turn, 1: through, 2: left}
  - *curSpd*: detection speed
  - *desSpd*: desired speed
  - *dist*: detection distance

Table 1: A few rows of a sample input traffic CSV-file.

lane	type	arrival time	L	maxAcc	maxDec	dest	curSpd	desSpd	dist
4	1	33.2	4.6	2	-2.5	0	5.7	16.3	500
2	0	27.6	4.8	2	-2.5	1	5.5	18	500
6	1	35.2	4.8	2	-2.5	1	8.1	17.6	500
8	1	41.2	4.6	2	-2.5	2	9.7	17.5	500

Any arbitrary intersection can be added in the src/intersection/data.py. The list of all available intersections is:

- 13th16th: A physical intersection in Gainesville; google map it for the image and lane assignment detail.
- TERL: A physical intersection located at 2612 Springhill Road, Tallahassee, FL 32305; google map it for the image and lane assignment detail. 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`: Genetic Algorithms
- `pretimed`: pretimed control
- `MCF`: Min Cost Flow (under development)
- `actuated`: Actuated Control System (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*):

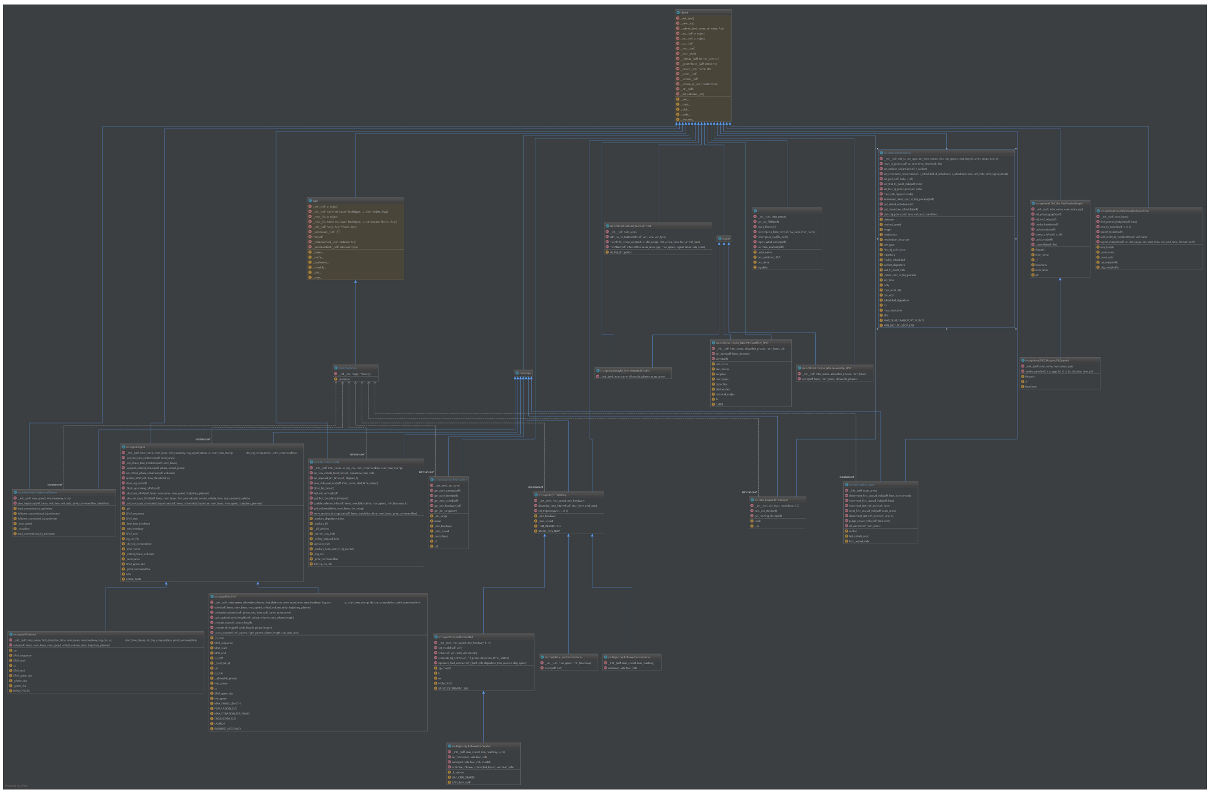

Fig. 1: The UML diagram of project.

1.2 Terms

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

SPaT Signal Phase and Timing. SPaT decision can be represented by the sequence and green duration of phases.

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 from the detection point to the stop bar.

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 in Gipps 1981 that models conventional vehicle movement. Read more here.

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

GA Genetic Algorithm used to optimize SPaTs decision. Read more here.

badness In the context of GA, badness is defined as the negative of the fitness of an individual. In other words, GA searches for an alternative with the least badness (equivalently, with the highest fitness) value.

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:

class `main.`**`Singleton`**

Only to make singleton classes.

The credit for this goes to this stackoverflow post.

`main.`**`run_avian`**(*inter_name*, *method*, *sc*, *start_time_stamp*, *tester*)

Note:

- Trajectories must end at the stop bar, i.e. the distance to stop bar converges to zero, even if they are temporarily assigned.
- list all the other assumptions here…

Parameters

- **`inter_name`** (*`str`*) – intersection name
- **`method`** (*`str`*) – pretimed, GA, …
- **`sc`** (*`int`*) – scenario number (*should match the appendix of the input CSV filename*)
- **`start_time_stamp`** – The UTC time stamp to name the CSV files

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

Organization University of Florida

CHAPTER

TWO

SIMULATION

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

Bases: object

Objectives:

- Keeps the time
- Moves the simulation clock forward

For time management we use seconds since the **Epoch** , or:

```
>>> import time
>>> time.time()
```

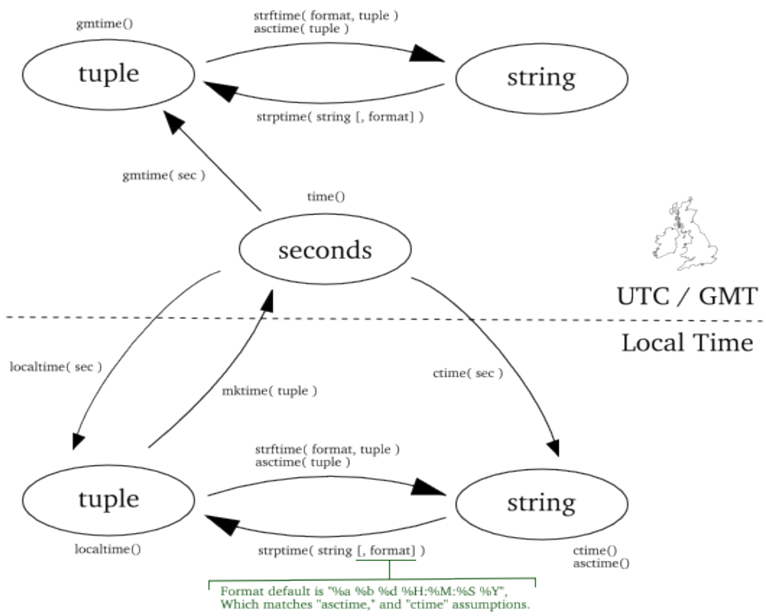

Fig. 1: Time management in python source.

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

CHAPTER

THREE

PYTHON DATA

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

Returns

- inter_name: intersection name
- max_speed: maximum speed in m/s
- min_headway: (s)
- det_range: detection range in m
- k, m:
- num_lanes: total number of incoming lanes
- phase_cover_set: a subset of mutually exclusive phases that cover all lanes for use in *`_set_non_base_scheduled_departures`*
- small_positive_num: small number that lower than that is approximated by zero
- large_positive_num: large number
- lag_on_green: The lag time from start of green when a vehicle can depart to allow vehicle cross after green (in seconds).
- max_num_traj_points: check if it's enough to preallocate the trajectory
- min_dist_to_stop_bar: lower than this (in m) do not update schedule
- do_traj_computation:
- trj_time_resolution: time difference between two consecutive trajectory points in seconds used in *`discretize_time_interval()`* (be careful not to exceed max size of trajectory)
- log_csv:
- print_commandline:

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.
- odd degree of polynomial is recommended: k to be even and **at least** 5
- `LARGE_NUM` is a large number to initialize badness of alternatives in GA. Make sure cannot be beaten by worst alternative.
- Make sure the `MAX_NUM_TRAJECTORY_POINTS` to preallocate the trajectories is enough for a given problem

---

> **Warning:** All the parameters defined here are required for running the program.

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, yellows, and all-reds based on traffic flow theory.

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

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

`data.data.`**`get_GA_parameters`**`(`*inter_name*`)`

- 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*
- max_iteration_per_phase:
- 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
- allowable_phases: subset of all possible phases to be used.

Returns

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

Returns a dictionary of sets where the keys are lane numbers and must be one-based. 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 want 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 considered as non-conflicting points. In the following figures, only cross and merge conflict points are indicated.

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

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

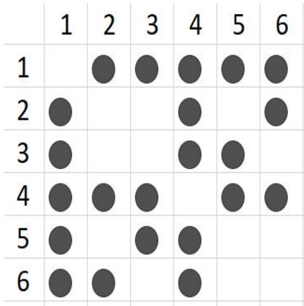

Fig. 1: The TERL facility.

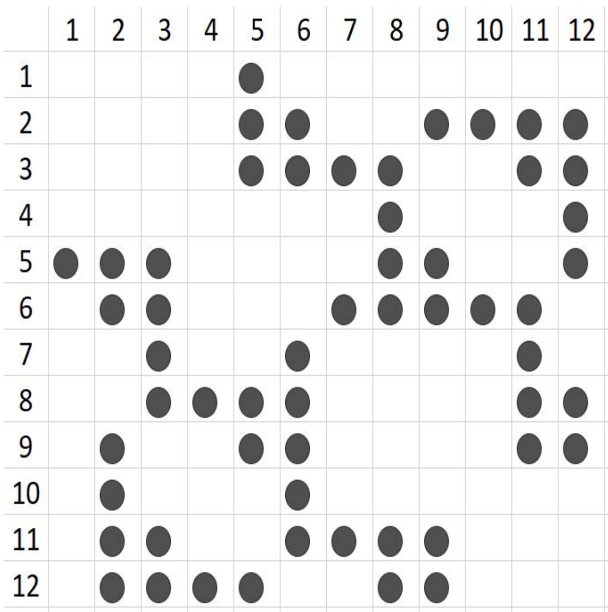

Fig. 2: The reservation-based intersection.

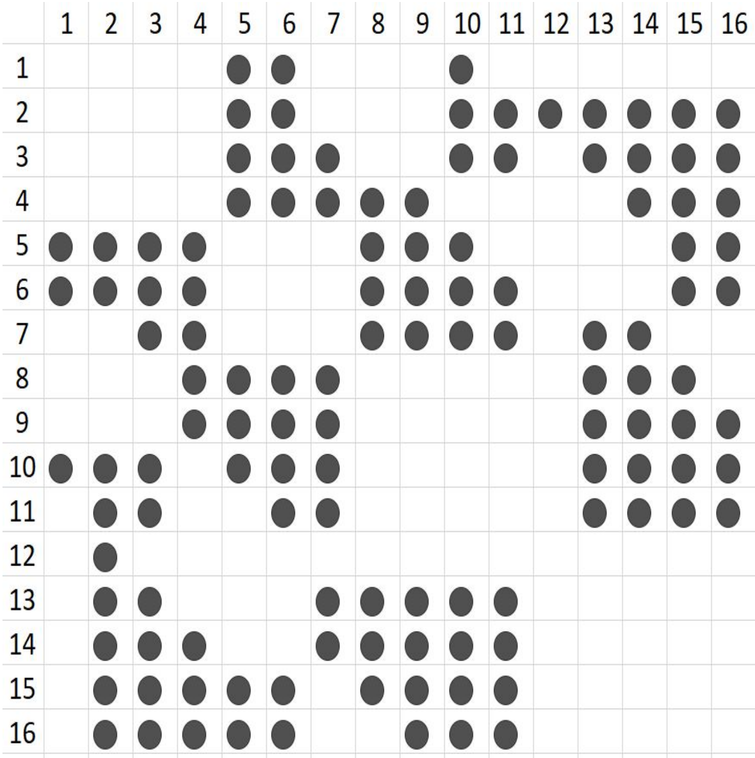

Fig. 3: The intersection of 13th and 16th, Gainesville, FL.

Returns A dictionary of sets. The key is the phase number and is one-based. The value to a key is a set of lanes included in that phase (lanes are also one-based).

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, and max green times are in seconds.

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

`data.data.`**`get_sig_ctrl_interface_params`**`(`*inter_name*`)`

Note:

- Account for SNMP lag time. Depending on the processor capability: [0.1s-0.9s]

Returns

- Proper phases to be called

Author Ash Omidvar <aschkan@ufl.edu>

Date May-2018

CHAPTER

FOUR

INTERSECTION

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

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

Bases: object

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

Objectives:

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

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

__init__(*intersection*)

Data Structure for keeping vehicles in order in the lanes in the form of a dictionary of arrays

Parameters **intersection** (Intersection) – keeps parameters related to the intersection

static refresh_earliest_departure_times(*lanes*, *intersection*)

" Computes the earliest departure time for all vehicles

decrement_first_unsrvd_indx(*lane*, *num_served*)

When vehicles get served, the first index to the unserved vehicle in a lane should change.

Parameters

- **n** – number of served vehicle
- **lane** – the lane in 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** – The index in which all vehicles with indices less than or equal to this get removed

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*, *intersection*)

Bases: `object`

Objectives:

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

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

__init__ (*det_id*, *det_type*, *det_time*, *speed*, *dist*, *des_speed*, *dest*, *length*, *amin*, *amax*, *indx*, *intersection*)

Initializes the vehicle object.

Attention:

- If the last trajectory point index is less than the first, no trajectory has been computed yet.
- The last trajectory index is set to -1 and the first to 0 for initialization purposes.
- 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* of an alternative.

Parameters

- **det_id** (`str`) – the *ID* assigned to 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
- **intersection** –
- **self.trajectory** – keeps the trajectory points as columns of a $3 \times n$ 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 the last trajectory point is stored.
- **self.poly** (*dict*) – keeps the reference time and the coefficients to reproduce trajectory of an AV
- **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 is ready to be 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 the specified size for trajectory array by `MAX_NUM_TRAJECTORY_POINTS` is enough to store all the trajectory points under the worst case.
- A vehicle may be open to being rescheduled but gets the same departure time; in that case, `freshly_scheduled` should hold False.

---

earliest_arrival_connected(*veh*, *max_speed*, *min_headway=0*, *t_earliest=0*)

Uses the latest departure time under the following cases to compute the earliest time the connected vehicle can re

- 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

- **veh** (Vehicle) –
- **max_speed** –
- **min_headway** –
- **t_earliest** – earliest timemap_veh_type2str of lead vehicle that is only needed if the vehicle is a follower vehicle

Returns The earliest departure time of the subject connected vehicle

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

earliest_arrival_conventional (*veh*, *max_speed*, *min_headway=0*, *t_earliest=0*)

Uses the latest departure time under the following cases to compute the earliest time the conventional vehicle can

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

Parameters

- **veh** (Vehicle) –
- **min_headway** –
- **t_earliest** – earliest time of lead vehicle that is only needed if the vehicle is a follower vehicle

Returns The earliest departure time of the subject conventional vehicle

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

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

Writes the trajectory points in the CSV file if the time stamp is before the `time_threshold` and then removes those points by updating the pointer to 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 meaningless.

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** – The CSV file to be written. It is initialized in *Traffic.__init__()* method, if `None`, this does not record points in CSV.

set_scheduled_departure (*t_scheduled*, *d_scheduled*, *s_scheduled*, *lane*, *veh_indx*, *intersection*)
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.
- Moves back the first trajectory point to make best use of limited size to store trajectory points

---

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 its 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

get_arrival_schedule ()

Returns The triple (t, d, s) corresponding to the arrival of subject vehicle

get_departure_schedule ()

Returns The triple (t, d, s) corresponding to the departure of subject vehicle

print_trj_points (*lane*, *veh_indx*, *identifier*)
Print the first and last trajectory point 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`**(*intersection*, *sc*, *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; saves 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 was 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__`(*intersection*, *sc*, *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*, *intersection*)

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
- **intersection** (Intersection) – intersection

static get_volumes(*lanes*, *intersection*)

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

Parameters

- **lanes** (Lanes) – includes all vehicles
- **intersection** –

Return volumes array of volume level per lanes

serve_update_at_stop_bar(*lanes*, *simulation_time*, *intersection*)

This looks for/removes the served vehicles.

Parameters

- **lanes** (Lanes) – includes all vehicles
- **simulation_time** – current simulation clock
- **intersection** (Intersection) –

class src.intersection.**TrajectoryPlanner**(*intersection*)

Bases: object

Plans trajectories of all type. This makes calls to trajectory classes' methods.

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

__init__(*intersection*)

Instantiates the **trajectory** classes

plan_trajectory(*lanes*, *veh*, *lane*, *veh_indx*, *intersection*, *tester*, *identifier*)

Parameters

- **tester** –
- **lanes** (Lanes) –
- **veh** (Vehicle) –
- **lane** –
- **veh_indx** –
- **intersection** –
- **identifier** – Shows type of assigned trajectory

CHAPTER

FIVE

SIGNAL PHASE AND TIMING (SPAT)

class src.signal.**Signal**(*intersection*, *sc*, *start_time_stamp*)

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

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

Note:

- The signal status is saved under `\log\<intersection name>\` directory.

Use Case:

Instantiate like:

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

Perform SPaT computation by:

```
>>> signal.solve(.)
```

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

__init__(*intersection*, *sc*, *start_time_stamp*)

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)

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`()

Sets the phase-phase incidence matrix of the intersection

`_append_extend_phase`(*phase*, *actual_green*, *intersection*)

Appends a phase to the *SPaT* (append/extend a phase and its green to the end of signal array)

Parameters

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

`update_SPaT`(*intersection*, *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`(*intersection*)

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*, *intersection*, *trajectory_planner*, *tester*)

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`) –
- **`intersection`** (`Intersection`) –
- **`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*, *intersection*)

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

- **intersection** (`Intersection`) –
- **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*, *trajectory_planner*, *intersection*, *tester*)

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.
- A cover phase set for all lanes is needed here.

Parameters

- **tester** –
- **lanes** (`Lanes`) –
- **scheduled_departure** –
- **trajectory_planner** (`TrajectoryPlanner`) –

class `src.signal.`**Pretimed** (*first_detection_time*, *intersection*, *sc*, *start_time_stamp*)

Bases: *`src.signal.Signal`*

Note: Assumptions: - The sequence and duration are pre-determined - Cycle length is pre-computed using the time budget concept in traffic flow theory

- min and max of 60 and 120 seconds bound the *cycle length*

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

`__init__` (*first_detection_time*, *intersection*, *sc*, *start_time_stamp*)
Initializes the pretimed SPaT

`solve` (*lanes*, *intersection*, *critical_volume_ratio*, *trajectory_planner*, *tester*)

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`) –
- **`intersection`** (`Intersection`) –

`_append_extend_phase` (*phase*, *actual_green*, *intersection*)
Appends a phase to the *SPaT* (append/extend a phase and its green to the end of signal array)

Parameters

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

`_do_base_SPaT` (*lanes*, *intersection*, *trajectory_planner*, *tester*)
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) –
- **intersection** (Intersection) –
- **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*, *intersection*)

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

- **intersection** (Intersection) –
- **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 (*intersection*)

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*, *trajectory_planner*, *intersection*, *tester*)

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.
- A cover phase set for all lanes is needed here.

Parameters

- **tester** –
- **lanes** (Lanes) –
- **scheduled_departure** –
- **trajectory_planner** (TrajectoryPlanner) –

_set_phase_lane_incidence ()

Sets the phase-phase incidence matrix of the intersection

close_sig_csv ()

Closes the signal csv file

update_SPaT (*intersection*, *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`**(*first_detection_time*, *intersection*, *sc*, *start_time_stamp*)

Bases: `src.signal.Signal`

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

> **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

`__init__`(*first_detection_time*, *intersection*, *sc*, *start_time_stamp*)

Parameters

- **inter_name** –
- **first_detection_time** –
- **intersection** (`Intersection`) –
- **sc** –
- **start_time_stamp** –

`solve`(*lanes*, *intersection*, *critical_volume_ratio*, *trajectory_planner*, *tester*)

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`) –
- **`intersection`** (`Intersection`) –
- **`critical_volume_ratio`** –
- **`trajectory_planner`** (`TrajectoryPlanner`) –

`_evaluate_badness` (*phase_seq*, *time_split*, *lanes*, *intersection*, *tester*)

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
- λ is weight factor in veh/s
- t is the average travel time in s, under the given *SPaT*.

Attention:

- A rough approximate for λ 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.
- Recursively computes the average travel time

Parameters

- **tester** –
- **phase_seq** –
- **time_split** –
- **lanes** (Lanes) – holds the traffic intended to be served
- **intersection** (Intersection) –

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

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

Appends a phase to the *SPaT* (append/extend a phase and its green to the end of signal array)

Parameters

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

_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.

_do_base_SPaT (*lanes*, *intersection*, *trajectory_planner*, *tester*)

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) –
- **intersection** (Intersection) –
- **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*, *intersection*)

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

- **intersection** (Intersection) –
- **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 (*intersection*)

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*, *trajectory_planner*, *intersection*, *tester*)

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.
- A cover phase set for all lanes is needed here.

Parameters

- **tester** –
- **lanes** (`Lanes`) –
- **scheduled_departure** –
- **trajectory_planner** (`TrajectoryPlanner`) –

_set_phase_lane_incidence ()

Sets the phase-phase incidence matrix of the intersection

close_sig_csv ()

Closes the signal csv file

update_SPaT (*intersection*, *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

SIX

TRAJECTORY

class `src.trajectory.Trajectory`(*intersection*)

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.

Author Mahmoud Pourmehrab <pourmehrab@gmail.com>

Date April-2018

discretize_time_interval(*start_time*, *end_time*)

Discretize the given time interval to an array of time stamps

Warning: It is inclusion-wise of the beginning and end of the interval.

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**(*intersection*)

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**(*intersection*)

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

static wiedemann99(*lead_d, lead_s, lead_a, lead_l, foll_d, foll_s, foll_s_des, cc0=1.35, cc1=1.1700000000000002, cc2=8.0, cc3=-12.0, cc4=-1.5, cc5=2.0999999999999996, cc6=0.0006, cc7=0.25, cc8=2.0, cc9=1.5*)

Parameters

- **lead_d** – lead vehicle distance to stop bar
- **lead_s** – lead vehicle speed
- **lead_a** –
- **lead_l** – length of lead vehicle
- **foll_d** – follower vehicle distance to stop bar
- **foll_s** – follower vehicle speed
- **foll_s_des** –
- **cc0** – Standstill Distance in m
- **cc1** – Spacing Time in s

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

- **cc2** – Following Variation (*max drift*) in m
- **cc3** – Threshold for Entering 'Following' in s
- **cc4** – Negative *Following* Threshold in m/s
- **cc5** – Positive *Following* Threshold in m/s
- **cc6** – Speed Dependency of Oscillation in $10^{-}4rad/s$
- **cc7** – Oscillation Acceleration in m/s^2
- **cc8** – Standstill Acceleration in m/s^2
- **cc9** – Acceleration at 80 km/h in m/s^2

Returns follower next acceleration rate

gipps(*lead_d*, *lead_s*, *lead_a*, *lead_l*, *foll_d*, *foll_s*, *foll_s_des*, *foll_a_min*, *foll_a_max*, *lead_a_min*, *dt*)

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

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

Fig. 1: Gipps car following formula.

Returns follower next acceleration rate

solve(*veh*, *lead_veh*)

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.
- Compared to W99 requires acc/dec on follower, dec on lead, dt, and does not need lead acc.

Parameters

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

static comp_speed_distance(*t0*, *d0*, *v0*, *a*, *t*, *foll_a_min*, *foll_a_max*)

If car-following models yielded unreasonable acceleration, fixes it.

Note:

- Speed should be positive
- Acceleration/deceleration constraints should be met.

Parameters

- **t0** –

- **d0** –
- **v0** –
- **a** –
- **t** –

Returns distance to stop bar and speed

class src.trajectory.**LeadConnected**(*intersection*)

> **Note:**
>
> - Trajectory function: $f(t) = \sum_{n=0}^{k-1} b_n \times (t/t_0)^n$
> - Negative of speed profile: $f'(t) = \sum_{n=1}^{k-1} n \times b_n \times (t/t_0)^{n-1}$
> - Negative of acceleration profile: $f''(t) = \sum_{n=2}^{k-1} n \times (n-1) \times b_n \times (t/t_0)^{n-2}$

Parameters

- **NUM_DIGS** – The accuracy to keep decimals
- **SPEED_DECREMENT_SIZE** – The final speed decrements from maximum to 0 by step-size defined by maximum speed divided 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*)

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

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

solve(*veh, lead_veh, model*)

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

Parameters

- **veh** (Vehicle) – subject vehicle
- **lead_veh** (Vehicle) – lead vehicle which could be *None* if no vehicle is in front.
- **model** (*CPLEX*) –

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

```
\ENCODING=ISO-8859-1
\Problem name: CAV trajectory optimization

Minimize
 obj: 44.05 b_0 + 22.025 b_1 + 14.6833333333333 b_2 + 11.0125 b_3 + 8.81 b_4
      + 7.34166666666667 b_5 + 6.29285714285714 b_6 + 5.50625 b_7
      + 4.89444444444444 b_8 + 4.405 b_9 + 4.00454545454545 b_10
      + 3.67083333333333 b_11 + 3.38846153846154 b_12 + 3.14642857142857 b_13
      + 2.93666666666667 b_14
Subject To
 det_dist:     b_0  = 499.871984004097
 det_speed:    b_1  = -238.274160236752
 dep_dist:     b_0 + b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 + b_8 + b_9 + b_10
               + b_11 + b_12 + b_13 + b_14  = 0
 dep_speed:    b_1 + 2 b_2 + 3 b_3 + 4 b_4 + 5 b_5 + 6 b_6 + 7 b_7 + 8 b_8
               + 9 b_9 + 10 b_10 + 11 b_11 + 12 b_12 + 13 b_13 + 14 b_14
                = -787.68448
 ub_speed_0:   0.0227014755959137 b_1 + 0.00216204529484893 b_2
               + 0.000154431806774923 b_3 + 0.00000980519408094751 b_4
               + 5.83642504818304e-7 b_5 + 3.33510002753317e-8 b_6
               + 1.85283334862954e-9 b_7 + 1.00834467952628e-10 b_8
               + 5.40184649746221e-12 b_9 + 2.85811983992709e-13 b_10
               + 1.49711039234276e-14 b_11 + 7.77719684333903e-16 b_12
               + 4.0120459906114e-17 b_13 + 2.05745948236482e-18 b_14
               >= -17.8816
 ub_speed_1:   0.0227014755959137 b_1 + 0.00432409058969785 b_2
               + 0.000617727227099693 b_3 + 0.0000784415526475801 b_4
               + 0.00000933828007709287 b_5 + 0.00000106723200881061 b_6
               + 1.1858133431229e-7 b_7 + 1.29068118979364e-8 b_8
               + 1.38287270335032e-9 b_9 + 1.46335735804267e-10 b_10
               + 1.53304104175899e-11 b_11 + 1.59276991351583e-12 b_12
               + 1.64333403775443e-13 b_13 + 1.68547080795326e-14 b_14
               >= -17.8816
 ub_speed_2:   0.0227014755959137 b_1 + 0.00648613588454678 b_2
               + 0.00138988626097431 b_3 + 0.000264740240185583 b_4
               + 0.0000472750428902827 b_5 + 0.0000081042930669056 b_6
               + 0.00000135071551115093 b_7 + 2.20524981412397e-7 b_8
               + 3.54415148698495e-8 b_9 + 5.6256372809285e-9 b_10
               + 8.84028715574478e-10 b_11 + 1.37770708920698e-10 b_12
               + 2.13216573329652e-11 b_13 + 3.28025497430233e-12 b_14
               >= -17.8816
 ub_speed_3:   0.0227014755959137 b_1 + 0.00864818117939571 b_2
               + 0.00247090890839877 b_3 + 0.000627532421180641 b_4
               + 0.000149412481233486 b_5 + 0.0000341514242819396 b_6
               + 0.00000758920539598658 b_7 + 0.00000165207192293585 b_8
               + 3.54015412057683e-7 b_9 + 7.49238967317848e-8 b_10
               + 1.56983402676121e-8 b_11 + 3.26199278288043e-9 b_12
               + 6.73109621864215e-10 b_13 + 1.38073768587531e-10 b_14
               >= -17.8816
 ub_speed_4:   0.0227014755959137 b_1 + 0.0108102264742446 b_2
               + 0.00386079516937308 b_3 + 0.00122564926011844 b_4
               + 0.00036477656551144 b_5 + 0.000104221875860411 b_6
               + 0.0000289505210723365 b_7 + 0.00000787769280879905 b_8
               + 0.00000211009628807117 b_9 + 5.58226531235761e-7 b_10
               + 1.46202186752223e-7 b_11 + 3.79745939616164e-8 b_12
               + 9.79503415676612e-9 b_13 + 2.51154721968362e-9 b_14 >= -17.8816
 ub_speed_5:   0.0227014755959137 b_1 + 0.0129722717690936 b_2
               + 0.00555954504389724 b_3 + 0.00211792192148466 b_4
               + 0.000756400686244522 b_5 + 0.000259337378140979 b_6
               + 0.0000864457927136597 b_7 + 0.0000282271976207868 b_8
               + 0.00000907302780668148 b_9 + 0.00000288032628783539 b_10
               + 9.05245404748266e-7 b_11 + 2.82154411869589e-7 b_12
               + 8.73335084358253e-8 b_13 + 2.68718487494847e-8 b_14 >= -17.8816
 ub_speed_6:   0.0227014755959137 b_1 + 0.0151343170639425 b_2
               + 0.00756715853197124 b_3 + 0.003363181569765 b_4
               + 0.00140132565406875 b_5 + 0.000560530261627499 b_6
               + 0.000217983990632916 b_7 + 0.0000830415202411111 b_8
               + 0.0000311405700904166 b_9 + 0.0000115335444779321 b_10
               + 0.0000042289663085751 b_11 + 0.00000153780593039094 b_12
               + 5.5531880819673e-7 b_13 + 1.99345213198826e-7 b_14 >= -17.8816
 ub_speed_7:   0.0227014755959137 b_1 + 0.0172963623587914 b_2
```

Fig. 2: Part of a sample CPLEX model.

compute_trj_points (*f*, *f_prime*, *departure_time_relative*)
Converts the polynomial trajectory to the trajectory points :param f: :param f_prime: :param departure_time_relative: span of the trajectory :return: t, d, s

optimize_lead_connected_trj (*veh*)
Computes a linear trajectory for the vehicle. This case should not happen except for the case the LP has no solution.

Parameters **veh** (Vehicle) – subject vehicle

Returns trajectory of subject lead CAV

class src.trajectory.**FollowerConnected** (*intersection*)
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, .)
```

Parameters

- **GAP_CTRL_STARTS** – This is the relative time when gap control constraints get added
- **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 ($0.0 \geq -1.0$ is always true)

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

solve (*veh*, *lead_veh*, *model*)
The only reason this class method exist is to access *optimize_follower_connected_trj* method.

Parameters

- **veh** (Vehicle) – subject vehicle
- **lead_veh** (Vehicle) – the vehicle in front of the subject
- **model** – the follower's CPLEX model

optimize_follower_connected_trj(*veh*, *lead_veh*)
Works based on the concept of hypothetical trajectory.

Parameters

- **veh** (Vehicle) – subject vehicle
- **lead_veh** (Vehicle) – lead vehicle which could be *None* if no vehicle is in front.

Returns trajectory of the subject follower AV in case the LP has no solution.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

T

U

V

W