AVIAN Documentation

Release 1.0.0

Mahmoud Pourmehrab

Apr 29, 2018

CONTENTS

1 Overview 1

2 Simulator/Data 5

3 Intersection 7

4 Signal Phase and Timing (SPaT) 13

5 Trajectory 37

6 Indices and tables 41

Python Module Index 43

Index 45

CHAPTER

ONE

OVERVIEW

Note: The program is to simulate the performance of an isolated intersection under traffic of AV and conventional vehicles with variety of signal control methods.

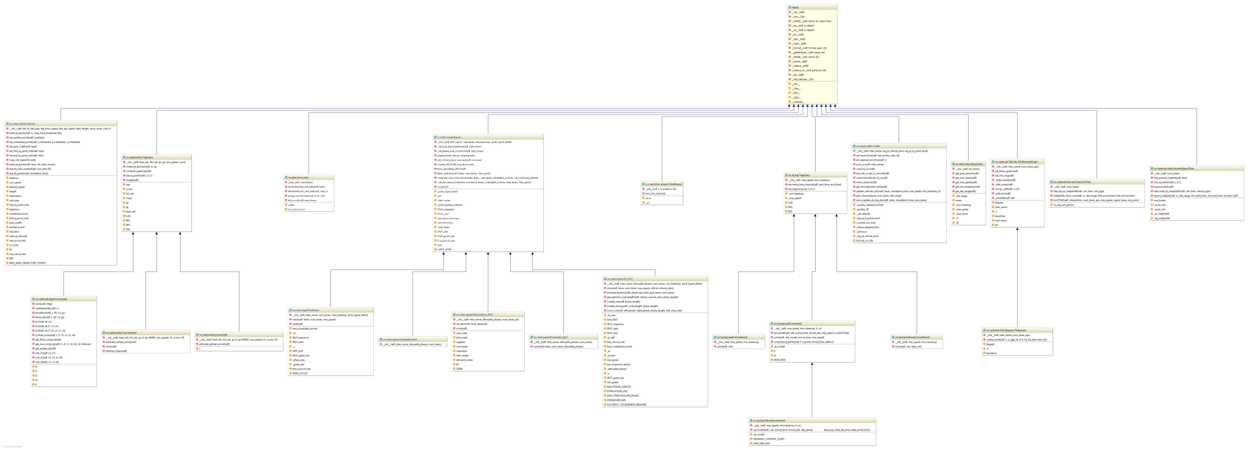
Coded by: Mahmoud Pourmehrab (mpourmehrab@ufl.edu)

Visit the project website here

Warning:

- The Python requirement is set to 3.5.2
- Install packages using `pip3 install -r requirements.txt`
- **For simulation, the directory `\data\` shall includes `<intersection name>.csv` which has the scenarios** to be tested. Note the filename should match the intersection name.
- SI units(speed in m/s, length in m, time in s, acceleration in m/s^2)
- **Run `python main <intersection_name> <optimization_algo>`**
  - intersection name could be `13th16th` or `reserv`
  - optimization algorithm could be `GA, pretimed` or `MCF`
- **Outputs are populated under `\log`:**
  - **`<intersection name>_vehicle_level` includes input csv plus the `departure time` and** `elapsed time` columns.
  - `<intersection name>_trj_point_level.csv` includes the trajectory points.

The **UML diagram** of the project is as the following (*Eagle eyes required, otherwise right-click and save as the image so you can zoom in*):



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

`main.`**`check_py_ver`**`()`

checks the python version to meet the requirement (`ver 3.5.2`)

`main.`**`run_avian`**`(`*inter_name*, *method*, *sc*, *do_traj_computation*, *log_at_vehicle_level*, *log_at_trj_point_level*, *print_clock*, *print_signal_detail*, *print_trj_info*, *test_time*, *print_detection*, *print_departure*`)`

For simulating a 12 lane four leg intersection (reservation intersection) with pretimed control do:

```
>>> python reserv pretimed simulation
```

For simulating intersection of 13th and 16th in Gainesville with GA do:

```
>>> python 13th16th GA simulation
```

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

- reserv
- 13th16th

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 (*each has certain requirements though*):

- simulation
- realtime

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: keeps lane-lane and phase-lane incidence dictionaries
  - 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**
  - Simulator:
- **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`** – Pretimed, GA, . . .
- **`sc`** – scenario number (*should match the appendix of the input csv filename*)
- **`do_traj_computation`** –
- **`log_at_vehicle_level`** –
- **`log_at_trj_point_level`** –
- **`print_clock`** – prints the simulation clock
- **`print_signal_detail`** –
- **`print_trj_info`** –
- **`test_time`** – in seconds from start of simulation

CHAPTER TWO

SIMULATOR/DATA

class src.time_keeper.**TimeKeeper**(*t*, *resolution=20.0*)

Bases: object

Goals:

1. Keeps the time
2. Move forward the time

Note: Set the **simulation resolution** in s at the default value of `TimeKeeper.__init__()`

__init__(*t*, *resolution=20.0*)

Clock keeps the simulation starting time in seconds (it gets reset for every scenario) Simulation resolution: the time steps to move the simulation forward in seconds

next_sim_step()

Move simulation clock forward

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

Returns a dictionary of sets The keys are lane numbers and must be coded in 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)

Available intersections:

1. 13th16th: a physical one, google map it in Gainesville for the image and lane assignment detail
2. reserv: reservation based model intersection: 12 incoming lanes (3 per approach and all lanes are exclusive).

Note: Assumes three exclusive discharge lanes (http://www.cs.utexas.edu/~aim/)

---

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

Returns max speed (m/s), min_headway (s), detection range (m), k, m, number of lanes

- 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 can be used for generic computations.

> **Warning:** Required for trajectory optimization

`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 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

`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.
- Compute green splits and yellows, all-reds based on traffic flow theory.

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

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

CHAPTER

THREE

INTERSECTION

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

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

Bases: object

Goals:

1. Keeps intersection parameters

__init__(*int_name*)

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

get_det_range()

get_max_speed()

get_min_headway()

get_num_lanes()

get_poly_params()

Returns K and M

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

Bases: object

__init__(*num_lanes*)

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

Goals:

1. Keeps vehicles in order
2. Keep track of index of last vehicle in each lane (useful for applications in Signal())
3. Remove served vehicles
4. Check if all lanes are empty

Parameters **num_lanes** – number of lanes

all_served(*num_lanes*)

Returns True if all lanes are empty, False otherwise

decrement_last_veh_indx(*lane*, *n*)

increment_last_veh_indx(*lane*)

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

class `src.intersection.`**Traffic**(*inter_name*, *sc*, *log_at_vehicle_level*, *log_at_trj_point_level*, *print_detection*)

Bases: `object`

Goals:

1. Add new vehicles from the csv file to the lanes.vehlist structure
2. Append travel time, ID, and elapsed time columns and save csv
3. Manages scenario indexing, resetting, and more
4. Compute volumes in lanes
5. remove/record 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.

__init__(*inter_name*, *sc*, *log_at_vehicle_level*, *log_at_trj_point_level*, *print_detection*)

Goals:

1. Set the logging behaviour for outputting requested CSV files and auxiliary output vectors
2. Import the CSV file that includes the traffic and sorts it
3. Initialize the first scenario number to run

close_trj_csv()

Closes trajectory CSV file

get_first_detection_time()

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

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

Unit of volume in each lane is veh/sec/lane Volume = Density x Space Mean Speed

Parameters

- **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

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.

---

save_csv(*inter_name*)
Set the recorded values and save the CSV at vehicle level

serve_update_at_stop_bar(*lanes*, *simulation_time*, *num_lanes*, *print_departure*)
This looks for/removes the served vehicles

Parameters

- **lanes** – includes all vehicles
- **simulation_time** – current simulation clock
- **num_lanes** – number of lanes

set_departure_time_for_csv(*departure_time*, *indx*, *id*)
Sets the departure time of an individual vehicle that is just served :param departure_time: departure time in seconds :param indx: row index in the sorted CSV file that has list of all vehicles :param id: ID of the vehicle being recorded

set_elapsed_sim_time(*t*)
Sets the elapsed time for one simulation of scenario :param t: elapsed time in seconds

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

Goals

1. Add vehicles from the csv file to lanes.vehlist
2. Assign their earliest arrival time

Parameters

- **lanes** (`dictionary of array as of 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.

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

Goals: 1) Defines the vehicle object that keeps all necessary information

1-1) Those which are coming from fusion 1-2) Those which are defined to be decided in the program: *trajectory[time, distance, speed], earliest_arrival, scheduled_arrival, poly_coeffs, _do_trj*

2. Update/record the trajectory points once they are expired
3. Keep trajectory indexes updated

4. Print useful info once a plan is scheduled
5. Decides if a trajectory re-computation is needed
6. Quality controls the assigned trajectory

Note:

1. Make sure the MAX_NUM_TRAJECTORY_POINTS to preallocate the trajectories is enough for given problem

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*)

Data Structure for an individual vehicle .. note:

```
- 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 :math:`3 *n` where :math:`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.
```

Warning:

- **The vehicle detection time shall be recorded in `init_time`. GA uses this field to compute travel time** when computing *badness* if an individual.

Parameters

- **det_id** – the *id* assigned to this vehicle by radio
- **det_type** – 0: Conventional, 1: Connected and Automated Vehicle
- **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

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

`print_trj_points`(*lane*, *veh_indx*)

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
- **`source`** – specifies which method has called the print

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

Writes 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_arrival`(*t_earliest*)

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

`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

`set_poly_coeffs`(*beta*)

Sets the coefficients that define the polynomial that defines trajectory of a connected vehicle

`set_scheduled_arrival`(*t_scheduled*, *d_scheduled*, *s_scheduled*, *lane*, *veh_indx*, *print*)

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`, avoid appending the schedule.

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** – `True` if we want to print schedule

test_trj_points(*simulation_time*)

Verifies the trajectory points for following cases:

1. Non-negative speed (threshold is set to -3 m/s)
2. Non-negative distance (threshold is set to -3 m)
3. Expired trajectory point is not removed

todo add more tests

Parameters **simulation_time** – the current simulation clock

test_trj_redo_needed(*min_dist=50*)

Checks if the trajectory model should be run (returns True) or not (False). Cases:

1. if last trajectory point is not assigned yet, do the trajectory.
2. if vehicle is closer than a certain distance, do NOT update the trajectory.

Parameters **min_dist** – for lower than this (in meters), no trajectory optimization or car following will be applied

Returns

CHAPTER FOUR

SIGNAL PHASE AND TIMING (SPAT)

class src.signal.**ActuatedControl**(*inter_name*, *allowable_phases*, *num_lanes*)

Bases: *src.signal.Signal*

todo: main problem is how to schedule the departures

LAG = 1

LARGE_NUM = 999999

__init__(*inter_name*, *allowable_phases*, *num_lanes*)

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 end of each list

_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.

_schedule_unserved_vehicles(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)

Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **`lanes`** –
- **`num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters **`num_lanes`** –

`_set_non_base_scheduled_arrival`(*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)

Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **`lanes`** –
- **`scheduled_arrivals`** –
- **`num_lanes`** –
- **`max_speed`** – by default the departure speed is maximum allowable speed in m/s

`_set_phase_lane_incidence`(*num_lanes*)

Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

`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

`base_badness`(*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

`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*)

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

Note:

- 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*

class src.signal.**Enumerate_SpaT**(*inter_name*, *allowable_phases*, *num_lanes*)

Bases: *src.signal.Signal*

Gives all phases equal chance but picks the one with highest throughput Similar to GA functionality (UNDER-DEVELOPMENT) :return:

LAG = 1

LARGE_NUM = 999999

__init__(*inter_name*, *allowable_phases*, *num_lanes*)

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 end of each list

_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.

_schedule_unserved_vehicles(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)

Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **`lanes`** –
- **`num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters **`num_lanes`** –

`_set_non_base_scheduled_arrival`(*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)

Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **`lanes`** –
- **`scheduled_arrivals`** –
- **`num_lanes`** –
- **`max_speed`** – by default the departure speed is maximum allowable speed in m/s

`_set_phase_lane_incidence`(*num_lanes*)

Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

`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

`base_badness` (*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

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

solve (*lanes*, *num_lanes*, *allowable_phases*)

update_SPaT (*time_threshold*)

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

Note:

- 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*

class src.signal.**GA_SPaT** (*inter_name*, *allowable_phases*, *num_lanes*, *min_headway*, *print_signal_detail*)
Bases: *src.signal.Signal*

Assumptions:

- The sequence and duration is decided optimally by a Genetic Algorithms

Parameters **allowable_phases** – subset of all possible phases to be used.

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

ACCURACY_OF_BADNESS_MEASURE = 100

CROSSOVER_SIZE = 10

LAG = 1

LARGE_NUM = 999999

MAX_ITERATION_PER_PHASE = 10

MAX_PHASE_LENGTH = 4

POPULATION_SIZE = 20

`__init__`(*inter_name*, *allowable_phases*, *num_lanes*, *min_headway*, *print_signal_detail*)

Parameters

- **`inter_name`** –
- **`allowable_phases`** (`tuple`) –
- **`num_lanes`** –
- **`min_headway`** –
- **`print_signal_detail`** –

`_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.

`_schedule_unserved_vehicles`(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)

Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **`lanes`** –
- **`num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters **`num_lanes`** –

_set_non_base_scheduled_arrival (*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)
Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **lanes** –
- **scheduled_arrivals** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in m/s

_set_phase_lane_incidence (*num_lanes*)
Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

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

base_badness (*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

`cross_over` (*left_parent*, *right_parent*, *phase_length*, *half_max_indx*)

Parameters

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

Returns

`evaluate_badness` (*phase_seq*, *time_split*, *lanes*, *num_lanes*)

Parameters

- **`phase_seq`** –
- **`time_split`** –
- **`lanes`** –
- **`num_lanes`** –

Returns

get_optimal_cycle_length (*critical_volume_ratio*, *phase_length*)
Uses the time budget concept $C = (n * ar)/(1 - V_{cr})$.

Refer to HCM 2010 for values.

Parameters

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

Returns

mutate_seq (*phase_length*)
Randomize the sequence # todo: if two same phases follow each other, re-sample carefully with replacement

Parameters **phase_length** –

Returns

mutate_timing (*cycle_length*, *phase_length*)
Creates the random phase split Valid timing should respect the min/max green requirement unless it conflicts with the cycle length which in that case we should adjust the maximum green to avoid the slack in time

note each timing is between `g_min+y+ar and g_max+y+ar`

Parameters

- **cycle_length** –
- **phase_length** –

Returns

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

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

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.

..note::

- We define `badness` as the measure that less of it is preferred for choosing a SPaT. Here we used travel time time `ACCURACY_OF_BADNESS_MEASURE` as the `badness` measure but any other measure can be simply used.
- GA has access to only subset of phases defined by `allowable_phases` from the full set in `data.py` file.
- 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`** –
- **`num_lanes`** –
- **`max_speed`** –
- **`critical_volume_ratio`** –

`update_SPaT`(*time_threshold*)

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

Note:

- 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

class `src.signal.`**`MinCostFlow_SPaT`**(*inter_name*, *allowable_phases*, *num_lanes*, *pli*)

Bases: *`src.signal.Signal`*

(UNDERDEVELOPMENT) This class is meant to slove the min cost flow problem that is set up for phase selection

Code is written in Python 3 Install the list of packages in the Pipfile using PyEnv solver by Google: https://goo.gl/jFncvj

NODES: Head of phase-selection arcs: from 0 to **|p|**-1 Head of sink arcs: from **|p|** to 2|p|-1 Head of lane-assignment arcs: from 2|p|+1 to 2|p|+|L|

ARCS: Phase-selection arcs are from 0 to **|p|**-1

cost 1 unit / cap of M

Phase-activator arcs are from |p| to 2|p|-1 cost 1 unit / cap of 1

Sink arcs are from 2|p| to 3|p|-1 cost 0 unit / cap of M

Lane-assignment arcs are from 3|p| to len(A) cost 0 unit / cap of M

(Note M is the large constant implemented as self.M)

CMIN = 1

LAG = 1

LARGE_NUM = 999999

M = 999

__init__(*inter_name*, *allowable_phases*, *num_lanes*, *pli*)

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 end of each list

_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.

_schedule_unserved_vehicles(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)

Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **lanes** –
- **num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters num_lanes –

_set_non_base_scheduled_arrival(*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)
Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **lanes** –
- **scheduled_arrivals** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in m/s

_set_phase_lane_incidence(*num_lanes*)
Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

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

base_badness(*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

`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

set_dem(*lanes_demand*)

solve()

update_SPaT(*time_threshold*)

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

Note:

- 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*

class src.signal.**Pretimed**(*inter_name*, *num_lanes*, *min_headway*, *print_signal_detail*)
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.

LAG = 1

LARGE_NUM = 999999

NUM_CYCLES = 2

__init__(*inter_name*, *num_lanes*, *min_headway*, *print_signal_detail*)
Initialize the pretimed 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.

_schedule_unserved_vehicles(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)
Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **`lanes`** –
- **`num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters **`num_lanes`** –

`_set_non_base_scheduled_arrival`(*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)

Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **`lanes`** –
- **`scheduled_arrivals`** –
- **`num_lanes`** –
- **`max_speed`** – by default the departure speed is maximum allowable speed in m/s

_set_phase_lane_incidence (*num_lanes*)
Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

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

base_badness (*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

`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

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

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`** –
- **`num_lanes`** –
- **`max_speed`** –

`update_SPaT`(*time_threshold*)

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

Note:

- 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*

class src.signal.**Signal**(*inter_name*, *num_lanes*, *min_headway*, *print_signal_detail*)

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.

LAG = 1

LARGE_NUM = 999999

__init__(*inter_name*, *num_lanes*, *min_headway*, *print_signal_detail*)

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 end of each list

_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.

_schedule_unserved_vehicles(*lanes*, *num_lanes*, *served_vehicle_time*, *any_unserved_vehicle*)

Sometimes 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 `redo_trj_allowed` to `False`.

Parameters

- **lanes** –
- **num_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_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 $|L| * |L|$ where element ij is 1 if i and j are conflicting movements

Parameters **num_lanes** –

_set_non_base_scheduled_arrival (*lanes*, *scheduled_arrivals*, *num_lanes*, *max_speed*)

Sets the scheduled departure in the trajectory of the vehicle.

Note:

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

Parameters

- **lanes** –
- **scheduled_arrivals** –
- **num_lanes** –
- **max_speed** – by default the departure speed is maximum allowable speed in m/s

_set_phase_lane_incidence (*num_lanes*)

Sets the phase-phase incidence matrix of the intersection #todo automate phase enumerator :param num_lanes: :return:

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

`base_badness` (*lanes*, *num_lanes*, *max_speed*)

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 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.

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.redo_trj_allowed` 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).

`self.first_unsrvd_indx` and setting the schedule of any possible served vehicles make the main result of this method. The `self.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 `redo_trj_allowed` 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 `redo_trj_allowed` 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.

Parameters

- **`lanes`** –
- **`num_lanes`** –
- **`max_speed`** –

Returns The `self.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

`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*)

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

Note:

- 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

CHAPTER

FIVE

TRAJECTORY

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

set_model(*veh*, *arrival_time*, *arrival_dist*, *dep_speed*, *lead_poly*, *lead_det_time*, *lead_arrival_time*)
Sets the LP model using the extra constraints to enforce the safe headway

Parameters

- **veh** – follower connected vehicle that the trajectory model is constructed for
- **arrival_time** – scheduled arrival time for this vehicle
- **arrival_dist** – scheduled arrival distance for this vehicle
- **dep_speed** – scheduled arrival speed for this vehicle
- **lead_poly** – the lead vehicle polynomial to regenerate necessary info at the control points
- **lead_det_time** – lead vehicle departure time
- **lead_arrival_time** – scheduled arrival time for lead vehicle

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

class src.trajectory.**FollowerConventional**(*max_speed*, *min_headway*)
Computes the trajectory for a follower conventional vehicle assuming a car following model.

solve(*veh*, *lead_veh*)
Gipps car following model is assumed here. It is written in-place (does not call set_trajectory)

Refer to: Gipps, Peter G. "A behavioural car-following model for computer simulation." Transportation Research Part B: Methodological 15.2 (1981): 105-111.

Note the only trajectory point index that changes is follower's last one

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

Note:

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

See also:

- Refer to `IBM(R) ILOG CPLEX Python API Reference Manual` for CPLEX usage using Python
- Docs for solver status codes

set_model (*veh*, *arrival_time*, *arrival_dist*, *dep_speed*, *is_lead=False*)
Overrides the generic coefficients to build the specific model

Parameters

- **veh** – vehicle object that its trajectory is meant to be computed
- **arrival_time** – time vehicle is scheduled to reach the stop bar
- **arrival_dist** – distance vehicle is scheduled to reach the stop bar
- **dep_speed** – speed vehicle is scheduled to reach the stop bar

solve (*veh*, *model*, *arrival_time*, *max_speed*)
Solves for connected vehicle (both lead and follower)

Parameters

- **veh** –
- **model** –
- **arrival_time** –
- **max_speed** –

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

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.

solve (*veh*)
Constructs the trajectory of a lead conventional vehicle assuming the driver maintains its speed

Parameters **veh** – the lead conventional vehicle

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

Abstract class for computing the trajectory points. Four subclasses inherited from Trajectory():

- LeadConventional
- FollowerConnected
- LeadConnected
- FollowerConventional

Note if want to omit the trajectory planning, there are two options:

- If a particular vehicle is intended to be skipped, simply invoke veh.set_redo_trj_false() whenever needed
- If the whole simulation is intended to be run without trajectory planer, set do_traj_computation in main.py

to False.

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

Parameters

- **LAG** – the lag time from start of green when a vehicle can depart
- **RES** – time difference between two consecutive trajectory points in second (be careful not to exceed max size of trajectory
- **EPS** – small number that lower than that is approximated by zero

discretize_time_interval(*start_time*, *end_time*)

Discretize the given time interval to a numpy array of time stamps The resolution is equal to the :param RES: (sec)

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

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

Parameters

- **veh** – 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)

src.trajectory.**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:

1. Accelerate/Decelerate to the maximum allowable speed and maintain the speed till departure
2. Distance is short, it accelerates/decelerated to the best speed and departs
3. 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

src.trajectory.**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:

1. Maintain the detected speed till departure
2. Depart 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

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

data.data, **??**

m

main, **??**

s

src.intersection, **??**
src.signal, **??**
src.time_keeper, **??**
src.trajectory, **??**

INDEX

Symbols

__init__() (src.intersection.Intersection method), 7
__init__() (src.intersection.Lanes method), 7
__init__() (src.intersection.Traffic method), 8
__init__() (src.intersection.Vehicle method), 10
__init__() (src.signal.ActuatedControl method), 13
__init__() (src.signal.Enumerate_SpaT method), 16
__init__() (src.signal.GA_SPaT method), 19
__init__() (src.signal.MinCostFlow_SPaT method), 25
__init__() (src.signal.Pretimed method), 28
__init__() (src.signal.Signal method), 32
__init__() (src.time_keeper.TimeKeeper method), 5
_flush_upcoming_SPaTs() (src.signal.ActuatedControl method), 13
_flush_upcoming_SPaTs() (src.signal.Enumerate_SpaT method), 16
_flush_upcoming_SPaTs() (src.signal.GA_SPaT method), 20
_flush_upcoming_SPaTs() (src.signal.MinCostFlow_SPaT method), 25
_flush_upcoming_SPaTs() (src.signal.Pretimed method), 28
_flush_upcoming_SPaTs() (src.signal.Signal method), 32
_schedule_unserved_vehicles() (src.signal.ActuatedControl method), 13
_schedule_unserved_vehicles() (src.signal.Enumerate_SpaT method), 16
_schedule_unserved_vehicles() (src.signal.GA_SPaT method), 20
_schedule_unserved_vehicles() (src.signal.MinCostFlow_SPaT method), 25
_schedule_unserved_vehicles() (src.signal.Pretimed method), 28
_schedule_unserved_vehicles() (src.signal.Signal method), 32
_set_lane_lane_incidence() (src.signal.ActuatedControl method), 14
_set_lane_lane_incidence() (src.signal.Enumerate_SpaT method), 17
_set_lane_lane_incidence() (src.signal.GA_SPaT method), 20
_set_lane_lane_incidence() (src.signal.MinCostFlow_SPaT method), 26
_set_lane_lane_incidence() (src.signal.Pretimed method), 29
_set_lane_lane_incidence() (src.signal.Signal method), 33
_set_non_base_scheduled_arrival() (src.signal.ActuatedControl method), 14
_set_non_base_scheduled_arrival() (src.signal.Enumerate_SpaT method), 17
_set_non_base_scheduled_arrival() (src.signal.GA_SPaT method), 20
_set_non_base_scheduled_arrival() (src.signal.MinCostFlow_SPaT method), 26
_set_non_base_scheduled_arrival() (src.signal.Pretimed method), 29
_set_non_base_scheduled_arrival() (src.signal.Signal method), 33
_set_phase_lane_incidence() (src.signal.ActuatedControl method), 14
_set_phase_lane_incidence() (src.signal.Enumerate_SpaT method), 17
_set_phase_lane_incidence() (src.signal.GA_SPaT method), 21
_set_phase_lane_incidence() (src.signal.MinCostFlow_SPaT method), 26
_set_phase_lane_incidence() (src.signal.Pretimed method), 29
_set_phase_lane_incidence() (src.signal.Signal method), 33

A

ACCURACY_OF_BADNESS_MEASURE (src.signal.GA_SPaT attribute), 19
ActuatedControl (class in src.signal), 13
all_served() (src.intersection.Lanes method), 7
append_extend_phase() (src.signal.ActuatedControl method), 14

append_extend_phase() (src.signal.Enumerate_SpaT method), 17
append_extend_phase() (src.signal.GA_SPaT method), 21
append_extend_phase() (src.signal.MinCostFlow_SPaT method), 26
append_extend_phase() (src.signal.Pretimed method), 30
append_extend_phase() (src.signal.Signal method), 33

B

base_badness() (src.signal.ActuatedControl method), 14
base_badness() (src.signal.Enumerate_SpaT method), 18
base_badness() (src.signal.GA_SPaT method), 21
base_badness() (src.signal.MinCostFlow_SPaT method), 26
base_badness() (src.signal.Pretimed method), 30
base_badness() (src.signal.Signal method), 34

C

check_py_ver() (in module main), 2
close_trj_csv() (src.intersection.Traffic method), 8
CMIN (src.signal.MinCostFlow_SPaT attribute), 25
cross_over() (src.signal.GA_SPaT method), 22
CROSSOVER_SIZE (src.signal.GA_SPaT attribute), 19

D

data.data (module), 5
decrement_last_veh_indx() (src.intersection.Lanes method), 7
discretize_time_interval() (src.trajectory.Trajectory method), 39

E

earliest_arrival_connected() (in module src.trajectory), 39
earliest_arrival_conventional() (in module src.trajectory), 39
Enumerate_SpaT (class in src.signal), 16
EPS (src.intersection.Vehicle attribute), 10
evaluate_badness() (src.signal.GA_SPaT method), 22

F

FollowerConnected (class in src.trajectory), 37
FollowerConventional (class in src.trajectory), 37

G

GA_SPaT (class in src.signal), 19
get_conflict_dict() (in module data.data), 5
get_det_range() (src.intersection.Intersection method), 7
get_first_detection_time() (src.intersection.Traffic method), 8
get_general_params() (in module data.data), 5
get_max_speed() (src.intersection.Intersection method), 7
get_min_headway() (src.intersection.Intersection method), 7
get_num_lanes() (src.intersection.Intersection method), 7
get_optimal_cycle_length() (src.signal.GA_SPaT method), 22
get_phases() (in module data.data), 6
get_poly_params() (src.intersection.Intersection method), 7
get_pretimed_parameters() (in module data.data), 6
get_signal_params() (in module data.data), 6
get_volumes() (src.intersection.Traffic static method), 8

I

increment_last_veh_indx() (src.intersection.Lanes method), 7
Intersection (class in src.intersection), 7

L

LAG (src.signal.ActuatedControl attribute), 13
LAG (src.signal.Enumerate_SpaT attribute), 16
LAG (src.signal.GA_SPaT attribute), 19
LAG (src.signal.MinCostFlow_SPaT attribute), 25
LAG (src.signal.Pretimed attribute), 28
LAG (src.signal.Signal attribute), 32
Lanes (class in src.intersection), 7
LARGE_NUM (src.signal.ActuatedControl attribute), 13
LARGE_NUM (src.signal.Enumerate_SpaT attribute), 16
LARGE_NUM (src.signal.GA_SPaT attribute), 19
LARGE_NUM (src.signal.MinCostFlow_SPaT attribute), 25
LARGE_NUM (src.signal.Pretimed attribute), 28
LARGE_NUM (src.signal.Signal attribute), 32
last_veh_arrived() (src.intersection.Traffic method), 8
LeadConnected (class in src.trajectory), 37
LeadConventional (class in src.trajectory), 38

M

M (src.signal.MinCostFlow_SPaT attribute), 25
main (module), 2
map_veh_type2str() (src.intersection.Vehicle static method), 10
MAX_ITERATION_PER_PHASE (src.signal.GA_SPaT attribute), 19
MAX_NUM_TRAJECTORY_POINTS (src.intersection.Vehicle attribute), 10
MAX_PHASE_LENGTH (src.signal.GA_SPaT attribute), 19
MIN_DIST_TO_STOP_BAR (src.intersection.Vehicle attribute), 10
MinCostFlow_SPaT (class in src.signal), 24
mutate_seq() (src.signal.GA_SPaT method), 23
mutate_timing() (src.signal.GA_SPaT method), 23

N

next_sim_step() (src.time_keeper.TimeKeeper method), 5
NUM_CYCLES (src.signal.Pretimed attribute), 28

P

POPULATION_SIZE (src.signal.GA_SPaT attribute), 19
Pretimed (class in src.signal), 28
print_trj_points() (src.intersection.Vehicle method), 11
purge_served_vehs() (src.intersection.Lanes method), 7

R

reset_trj_points() (src.intersection.Vehicle method), 11
run_avian() (in module main), 2

S

save_csv() (src.intersection.Traffic method), 9
serve_update_at_stop_bar() (src.intersection.Traffic method), 9
set_critical_phase_volumes() (src.signal.ActuatedControl method), 15
set_critical_phase_volumes() (src.signal.Enumerate_SpaT method), 19
set_critical_phase_volumes() (src.signal.GA_SPaT method), 23
set_critical_phase_volumes() (src.signal.MinCostFlow_SPaT method), 27
set_critical_phase_volumes() (src.signal.Pretimed method), 31
set_critical_phase_volumes() (src.signal.Signal method), 35
set_dem() (src.signal.MinCostFlow_SPaT method), 28
set_departure_time_for_csv() (src.intersection.Traffic method), 9
set_earliest_arrival() (src.intersection.Vehicle method), 11
set_elapsed_sim_time() (src.intersection.Traffic method), 9
set_first_trj_point_indx() (src.intersection.Vehicle method), 11
set_last_trj_point_indx() (src.intersection.Vehicle method), 11
set_model() (src.trajectory.FollowerConnected method), 37
set_model() (src.trajectory.LeadConnected method), 38
set_poly_coeffs() (src.intersection.Vehicle method), 11
set_scheduled_arrival() (src.intersection.Vehicle method), 11
set_trajectory() (src.trajectory.Trajectory static method), 39
Signal (class in src.signal), 32
solve() (src.signal.Enumerate_SpaT method), 19
solve() (src.signal.GA_SPaT method), 23
solve() (src.signal.MinCostFlow_SPaT method), 28
solve() (src.signal.Pretimed method), 31
solve() (src.trajectory.FollowerConventional method), 37
solve() (src.trajectory.LeadConnected method), 38
solve() (src.trajectory.LeadConventional method), 38
src.intersection (module), 7
src.signal (module), 13
src.time_keeper (module), 5
src.trajectory (module), 37

T

test_trj_points() (src.intersection.Vehicle method), 12
test_trj_redo_needed() (src.intersection.Vehicle method), 12
TimeKeeper (class in src.time_keeper), 5
Traffic (class in src.intersection), 8
Trajectory (class in src.trajectory), 38

U

update_SPaT() (src.signal.ActuatedControl method), 16
update_SPaT() (src.signal.Enumerate_SpaT method), 19
update_SPaT() (src.signal.GA_SPaT method), 24
update_SPaT() (src.signal.MinCostFlow_SPaT method), 28
update_SPaT() (src.signal.Pretimed method), 31
update_SPaT() (src.signal.Signal method), 35
update_vehicles_info() (src.intersection.Traffic method), 9

V

Vehicle (class in src.intersection), 9