Multi Agents in A city traffic Simulation High Level Design

ALon ReicHer, Liad gam

‏2023

תוכן

[Preface 2](#_Toc142058236)

[Notes for current version 2](#_Toc142058237)

[Simulation Concept – Model Description 2](#_Toc142058238)

[Environment 2](#_Toc142058239)

[Agent 2](#_Toc142058240)

[GUI 3](#_Toc142058241)

[Simulation Parameters 3](#_Toc142058242)

[Imported Data 3](#_Toc142058243)

[Fixed parameters 3](#_Toc142058244)

[Configurable parameters 3](#_Toc142058245)

[Exported Data 3](#_Toc142058246)

[Software Architecture 5](#_Toc142058247)

[Class Diagram (Unified Modeling Language) 5](#_Toc142058248)

[Data Structures 6](#_Toc142058249)

[Class: Route (Abstract Base Class) 6](#_Toc142058250)

[Class: Q\_Learning\_Route (Route) 9](#_Toc142058251)

[Class: Shortest\_path\_route (Route) 11](#_Toc142058252)

[Class: Car 12](#_Toc142058253)

[Class: Road 16](#_Toc142058254)

[Timing Diagram 20](#_Toc142058255)

[Bibliography 20](#_Toc142058256)

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
|  | 4.8.23 |  |  |
| 0.1 |  |  | Creation |

## Preface

### Notes for current version

# Simulation Concept – Model Description

## Environment

The environment is made up of one-way roads, each road starts at an intersection and ends at another intersection. Each road will include its length, the current speed on it, the roads you can pass to from it. As soon as an agent reaches the road, the agent will calculate with the help of the road data the time it will take him to complete the trip on this road.

## Agent

An agent is a car driving on the roads on the way to a predetermined destination. The agent will be able to use different ways to determine the mode of movement. It can be randomly oriented, according to the shortest distance from the target, or a reinforcement-based learning algorithm (Q-learning).

An agent's state is defined by the current time (time slice and day of the week), current vertex and destination vertex. An agent will take information from the relevant table for the day of the week from a table updated for that day and the time slice of the previous week, according to its current location, the current time slice, and the target vertex.

For example: a car in curve 1 that wants to reach curve 3, at 10:00 on Sunday, will check last week's table - on Sunday at 10:00, when the current curve is 1 and the destination is curve 3, which next curve (continuation of curve 1) will give the maximum return.

In the simulation we will have a Q table from each source vertex to each destination vertex on the simulation graph, the return for each arc is negative, and is defined in direct proportion to the time it takes to cross it.

An agent will also contribute to filling the distance table: when an agent uses the shortest path algorithm - if the data for his trip is not in the distance table, he will update the distance table according to his origin and destination. Otherwise, the relevant data for the agent already exists in the table, and he will take it from there.

● In the event that an agent uses the shortest path algorithm, and encounters a block in an arc he wanted to enter - he will choose an arc from among the other optional arcs, and will continue to act according to the same principle.

● For all algorithms - if an agent cannot move forward due to blocking of all arcs, it will wait/stop completely

## GUI

At the end of the simulation, the map of the area will be printed with two main features:

1. Tracking the travel path of specific agents on the map.

2. Color the edges colors based on road's speeds.

# Simulation Parameters

### Imported Data

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Calculated value** |
| G | Graph of the roads, as imported with the OSM API. | --- |
| Speed\_Json | Data (in a JSON format) about cars speed on the roads in the imported OSM graph. |  |
|  |  |  |

### Fixed parameters

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Value** |
| INTERVAL\_TIME | The time intervals of the speed data. | 10 (minutes) |
|  |  |  |

### Configurable parameters

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Valid values** |
| location | The simulated location. A city/ part of a city. | City names\coordinates |
| routing\_algorithm | The algorithm which is used by agents to determine the next road/edge of their route. | "random"/"shortest path"/"q learning" |
| Speed\_noise\_dist | The distribution used to generate synthetic movement data from the real data. The generation method will apply noise to the real data, using gaussian distribution or uniform distribution between -10 to 10 (with no negative speeds). | "gauss"/"uni" |

### Exported Data

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Calculated value** |
| route | Routes of the agents that finished their navigation | route |
| Q\_table | Calculated Q table |  |

# Software Architecture

## Class Diagram (Unified Modeling Language)[[1]](#footnote-1)תמונה שמכילה תרשים התיאור נוצר באופן אוטומטי



## Data Structures

# Class: Route (Abstract Base Class)

This abstract base class defines the interface and common methods for different routing strategies.

**Method: decide\_first\_road (source\_node, road\_network)**

**Input:**

* **source\_node** (int): The starting node ID.
* **road\_network** (Road\_Network): The road network object.

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It is responsible for deciding the first road that a car should take.
* Returns the selected first road.

**Method: get\_next\_road (source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Input:**

* **source\_road** (Road): The current road the car is on.
* **destination\_node** (int): The destination node ID.
* **adjacency\_list** (list of Road): List of adjacent roads to the current road.
* **road\_network** (Road\_Network): The road network object.
* **time** (datetime.timedelta): The current time.

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It determines the next road that the car should take based on the current road and other parameters.
* Returns the selected next road.

**Method: get\_alt\_road (self, source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Input:**

* **source\_road** (Road): The current road the car is on.
* **destination\_node** (int): The destination node ID.
* **adjacency\_list** (list of Road): List of adjacent roads to the current road.
* **road\_network** (Road\_Network): The road network object.
* **time** (datetime.timedelta): The current time.

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It determines an alternative road when the chosen road is blocked or unavailable.
* Returns the selected alternative road.

**Class: Random\_route (Route)**

This class represents a random routing strategy for car simulation.

**Method: decide\_first\_road (self, source\_node, road\_network)**

**Functionality:**

* Selects the first road randomly from roads connected to the source node.

**Output:**

* (road): the selected first road.

**Method: get\_next\_road (self, source\_node, destination\_node, adjacency\_list, road\_network, time)**

**Functionality:**

* Selects the next road randomly from the list of adjacent roads.
* Ensures that the selected road is not blocked and has adjacent roads.

**Output:**

* (road): the selected road.

**Method: get\_alt\_road (self, source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Input and Functionality:**

This method is not implemented in this class.

# Class: Q\_Learning\_Route (Route)

This class represents a Q-learning based routing strategy for car simulation.

**Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network, start\_time)**

**Input:**

* src\_node (int): The source node ID.
* dst\_node (int): The destination node ID.
* road\_network (Road\_Network): The road network object.
* start\_time (datetime.datetime): The starting time of the simulation.

**Functionality:**

* Initializes the Q-learning route with source, destination, and other parameters.
* Creates an instance of the QLearning class to perform Q-learning.
* Loads or trains the Q-table based on previous experience or initializes a new one.
* Defines the agent's path using Q-learning.

**Method: decide\_first\_road (self, source\_node, road\_network)**

**Functionality:**

* Chooses the first road using Q-learning based on the source node.

**Output:**

* (road): the selected first road.

**Method: get\_next\_road (self, source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Functionality:**

* This method uses Q-learning to select the next road based on the current road's destination node.

**Output:**

* (road): the selected road.

**Method: get\_alt\_road(self, source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Input and Functionality:**

This method is not fully implemented and can be enhanced to provide alternative road options.

# Class: Shortest\_path\_route (Route)

This class represents a shortest path-based routing strategy for car simulation.

**Method: decide\_first\_road (self, source\_node, road\_network)**

**Functionality:**

* Determines the first road using a shortest path algorithm.

**Output:**

* (road): the selected first road.

**Method: get\_next\_road (self, source\_node, destination\_node, adjacency\_list, road\_network, time)**

**Functionality:**

* Determines the next road using a shortest path algorithm.

**Output:**

* (road): the selected road.

**Method: get\_alt\_road (self, source\_road, destination\_node, adjacency\_list, road\_network, time)**

**Input and Functionality:**

* This method calculates an alternative road based on minimum distance from adjacent roads.
* Returns the selected alternative road (Road object) or None if no alternative is found.

# Class: Car

**\_\_init\_\_(self, id: int, source\_node: int, destination\_node: int, starting\_time: datetime, road\_network: Road\_Network, route\_algorithm = 'random')**

**Input:**

* id (int): The unique identifier for the car.
* source\_node (int): The source node from which the car starts its journey.
* destination\_node (int): The destination node where the car aims to arrive.
* starting\_time (datetime): The time when the car's journey begins.
* road\_network (Road\_Network): The road network within which the car operates.
* route\_algorithm (str, optional): The algorithm the car will use to decide its route (default: 'random').

**Functionality:**

* Initializes a new instance of the Car class with the provided attributes.
* Sets up various attributes to manage the car's state, route, and movement within the simulation.

**Returns:**

* None

**decide\_route\_algorithm(self, route\_algorithm: str, source\_node: int, destination\_node: int) -> Route**

**Input:**

* route\_algorithm (str): A string representing the desired route algorithm.
* source\_node (int): The source node for the route.
* destination\_node (int): The destination node for the route.

**Functionality:**

* Decides and creates a route object based on the specified route algorithm.
* The supported route algorithms are Q-learning, shortest path, and random route.

**Returns:**

* A route object that corresponds to the selected route algorithm.

**start\_car(self) -> Road**

**Functionality:**

* Moves the car to the first road based on the starting node.
* Updates the car's time until the next road.
* Records the source node in the list of past nodes.

**Returns:**

* The first road the car will travel on.

**decide\_next\_road(self) -> Road**

**Functionality:**

* Decides the next road the car will travel to based on its route.
* Determines the next road by considering the destination node and adjacent roads.

**Returns:**

* A road object representing the next road the car will travel on.

**decide\_alt\_road(self) -> Road**

**Functionality:**

* Decides an alternative road choice if the current road is blocked.

**Returns:**

* A road object representing an alternative road choice, or None if all roads are blocked.

**move\_next\_road(self, time: float) -> Union[None, str, Road]**

**Input:**

* time (float): The time increment for the car's movement.

**Functionality:**

* Moves the car to the next road based on its route's next node.
* Updates the car's time until the next road.
* Handles cases where the car finishes its journey or encounters blocked roads.
* Updates the car's distance traveled, past nodes, and past roads.

**Returns:**

* None if the car finishes its journey or if all roads are blocked.
* "blocked" if the car is blocked by all roads.
* A road object representing the next road the car will travel on.

**check\_if\_finished(self) -> bool**

**Functionality:**

* Checks if the car has reached its destination road.

**Returns:**

* True if the car has reached its destination road, otherwise False.

**force\_finish(self) -> None**

**Functionality:**

* Forces the car to finish its journey.
* Records the current road and time in the list of past roads.

**Returns:**

* None

**update\_time\_until\_next\_road(self, road: Road) -> None**

**Input:**

* road (Road): The road for which to calculate the time until the next road.

**Functionality:**

* Updates the car's time until the next road based on the given road's properties.

**Returns:**

* None

**update\_travel\_time(self, time: float) -> None**

**Input:**

* time (float): The time increment to update the total travel time.

**Functionality:**

* Updates the car's total travel time and time until the next road.

**Returns:**

* None

# Class: Road

**\_\_init\_\_(self, id: int, source\_node, destination\_node, length, max\_speed)**

Constructor for the Road class.

**Inputs:**

* id (int): Unique identifier for the road.
* source\_node (Node): The starting node of the road (class Node).
* destination\_node (Node): The ending node of the road (class Node).
* length (float): Length of the road segment in meters.
* max\_speed (int): Maximum speed limit of the road in km/h.

**block(self)**

**Blocks the road, indicating it is unavailable for traffic.**

**Inputs: (None)**

**Functionality:**

* Sets the is\_blocked attribute to True, indicating the road is blocked and not accessible for traffic.

**Output: (None)**

**Block\_road():**

Inputs: none

Outputs: none.

Method:

block the road for passing of agents. Agent won't be able to choose this road.

* Sets attributes of is\_blocked to true and current\_speed to 0.

**unblock\_road():**

Inputs: none

Outputs: none.

Method:

Revert the action of block\_road(). The road will be back to normal use in the next update of speed.

* Sets attributes of is\_blocked to false and current\_speed will be updated normally according to the JSON of the speeds.

**Road\_Network**

**Update\_roads\_speeds(time):**

Inputs: current time

Outputs: none.

Method:

for each road:

* go to the json file that saves speed of the road based on the time of the day.
* use the current time in order to get to the right sub dictionary.
* calculate modified speed with modify\_speed\_parameter() and modify\_speed\_noise() (the value returned from the former multiplied by the parameter in the latter.
* update the road's current speed.

if the calculated speed is 0 or less, the road will be blocked. if it was blocked and the calculated speed turned to be other than 0, the road will be unblocked. if the calculated speed is above the maximum, it will be capped to the maximum speed.

**Check\_and\_set\_reachble\_roads():**

Inputs: none

Outputs: for every road update the adjacent roads.

Method:

This function will work before the simulation starts and after we load the map and initialize all the roads in the simulation.

At the end every road will have a list of his adjacent roads.

The method will use a map from node id (representing a destination node) to edges array (that have the destination as their source).

* for every road in the simulation:
  + if the destination node is not in the map keys:
    - add to the map
  + if the source node is in the map keys:
    - add the road (\*or destination node\*) to that key

(1,2) {{2:(2,4)},{3:(3,4)},{4:}}

(1,3)

(2,4)

(3,4)

**Block\_road(road index):**

Inputs: int-road index

Outputs: none.

Method:

Activate the function **block\_road()** in Road class for the requested road.

**unblock\_road(road index):**

Inputs: none

Outputs: none.

Method:

Revert the action of block\_road(). The road will be back to normal use in the next update of speed.

**modify\_speed\_parameter(road index):**

Inputs: Road ID.

Outputs: speed multiplier.

Method:

The method chooses a random parameter between {1,0.5,0}, where 0 has a low probability to be chosen, and multiplies the current speed with it.

**modify\_speed\_noise(road index):**

Inputs: Road ID.

Outputs: fixed speed.

Method:

Using the relevant function (Gaussian distribution or uniformal distribution), this method generates a slight change in the road’s speed in order to add a white noise.

**Simulation\_Manager**

**Start\_simulation(location, running time):**

Inputs: location, running time

Outputs: none.

Method:

Start the simulation of the road network in the location chosen with the help of Open Street Maps. Speed\_Json is required for the location to get the speed of the roads. Agents will appear at a random time and random places. The simulation will stop after the clock will get to the value of running time.

Before running the simulation, the simulation manager will load the relevant Q table for the relevant area, and day of the week.

the method will receive from the console (inputted by the user) the parameter for noise and the routing algorithm. the

**Car\_Manager**

**add\_agent(source, destination):**

Inputs: int that represents the road id of the source, int that represents the road id of the destination

Outputs: none.

Method:

Creates an instance of agent with the source and destination. Add it to the simulation.

**remove\_agent(id):**

Inputs: agent ID to remove

Outputs: none.

Method:

The method will remove the agent from the list of active agents, extract its statistics.

**how to save the statistics? and what statistics?**

## Timing Diagram

1.call simulation manger() -> 2. Simulation manager will create road\_network with all the road in it -> 3. Random agents will be created at different times -> 4. Every agent will use Route class to navigate to its destination ->5. Simulation will stop after certain amount of time and the ending result will be shown.

## Bibliography

|  |  |
| --- | --- |
| [1] | IEEE-SA, *IEEE 802.11n-2009—Amendment 5: Enhancements for Higher Throughput,* 2009. |
| [2] | IEEE-SA, *IEEE 802.11-2007,* 2007. |

1. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> [↑](#footnote-ref-1)