CAP5830 Final Project: Implementing Ant-Colony Route Optimization with SUMO

Quentin Goss *Graduate Student of Applied Computer Science*
Florida Polytechnic University
Lakeland, Florida
quentingoss0323@floridapoly.edu

I. Introduction

In this paper, we demonstrate our knowledge of Modeling, Simulation, and Discrete Events System Specification (DEVS) with an implementation of the Ant Colony route finding approach to solve the Traveling Salesman problem.

The ant colony approach was first introduced by Dorigo et. al in 1996 [1] in which they propose a solution to the traveling salesman problem influenced by the route-finding approach of the familiar biting insects. Ants will explore in a direction at random, leaving a pheromone trail behind them where they travel. When food is located, they turn around and follow the trail back. The closer food is to the ant colony, the more the ant will travel the same path, increasing the amount of pheromone on that trail. The other ants, incidentally prefer trails with more pheromone and will likely choose the path with the strongest pheromone as optimal route. Additionally, there is still a need to explore for more food, which results in a small likelihood to travel in another direction.

In transportation systems (TS), vehicles must have a route to travel around the TS. Vehicles, like ants, may be modeled as agents in an agent based simulation. Additionally, vehicles travels along paths in a street network (SN) of a TS, much like ants travel along the paths of pheromone trails. Thus, the ant colony approach is applicable to vehicles in a SN of a TS where the ants (vehicles) traverse trails (connections) to find food (points of interest (POIS)). We implement the ant colony approach using the traffic simulation software SUMO [2], the advanced Python 3.x interface TraCI [3], the Python 3.x graph module NetworkX [4], and a realistic street network obtained from OpenStreetMaps (OSM) [5].

The approach that we recreate is summed up concisely by the Wikipedia article on the Ant Colony approach [6]:

1) It must visit each city exactly once;
2) A distant city has less chance of being chosen (the visibility);
3) The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen;
4) Having completed its journey, the ant deposits more pheromones on all edges it traversed, if the journey is short;
5) After each iteration, trails of pheromones evaporate.

The code for this project may be found on github at https://github.com/QuentinGoss/CAP5830-Final.

II. Process

Algorithm 1 Preprocessing

Retrieve a SN from OSM
Generate a SUMO SN from the OSM SN
Generate the NetworkX Graph of the SUMO SN
Select 10 POIs to Visit
Read/Load the SUMO edges from XML to Dict
Read/Load the SUMO nodes from XML to Dict
Generate a 2D matrix of Connections for every (POI + start) to every other (POI + start)
return

Algorithm 2 Hop (Each Vehicle)

$i \in \{1, \ldots, \text{nodes2visit}\}$
Update C where $Ci = \text{cost} * \text{avg([from][to].trails)}$
Update p where $p_i = Ci/\text{sum}(C)$ where $\text{sum}(p) = 1$
Pick a destination i at random given p.
conn[from][to].visits += 1
conn[from][to].trails.append(d_t) where $dt =$ Time it takes to travel along the connection.
return

III. Models

A. Vehicle

$$
\begin{align}
\text{Vehicle} &= (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta) \tag{1a}\\
R &= r \forall R \in \text{POIs that need to be visited.} \tag{1b}\\
\text{start} &= \text{starting location} \tag{1c}\\
E &= \{\text{True,False}\}\text{Vehicle exists in the SUMO simulation.} \tag{1d}\\
\text{inPorts} &= \text{"in"} \tag{1e}\\
t_d &= \text{Time vehicle depart.} \tag{1f}\\
dt &= \text{Trip time.} \tag{1g}\\
X &= E \tag{1h}\\
\text{outPorts} &= r \text{for} r \in R \tag{1i}\\
Y &= \text{this (This vehicle model.)} \tag{1j}\\
S &= \{\text{"not departed","departed","arrived","finished"}\} * R * E * t_d * dt \tag{1k}\\
\delta_{ext}((phase, R, E, T_d, dt), e, x) &= \begin{cases} ((\text{"not departed"}, R, True, t_d = \text{now}(), dt), 1, x) & phase = \text{"arrived"} \,\&\, E = \text{False} \\ ((\text{"departed"}, \text{random}(R), True, t_d, dt), 0, x) & phase = \text{"not departed"} \,\&\, E = \text{True} \\ ((\text{"arrived"}, r, False, t_d, dt = \text{now}() - t_d), 0, x) & phase = \text{"departed"} \,\&\, E = \text{False} \,\& \\ \cdots & R.\text{size}() > 0 \\ ((\text{"finished"}, \text{start}, False, t_d, dt = \text{now}() - t_d), 0, x) & phase = \text{"departed"} \,\&\, E = \text{False} \,\& \\ \cdots & R.\text{size}() = 0 \end{cases} \tag{1l}\\
\delta_{int}(phase, \sigma) &= (phase, \sigma) \tag{1m}\\
\lambda(phase, \sigma) &= \text{this} \tag{1n}
\end{align}
$$

B. Connection

$$
\begin{align}
\text{Connection} &= (X, Y, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta) \tag{2a}\\
V &= \text{Number of visits} \in \mathbb{R}^+ \tag{2b}\\
T &= \text{Pheromone Trail weight} \in [0, 1] \tag{2c}\\
X &= \text{Vehicle} \tag{2d}\\
Y &= \text{Vehicle} \tag{2e}\\
S &= \{\text{"Active","Passive"}\} * V * T \tag{2f}\\
\delta_{ext}((phase, V, T), 1, x) &= ((\text{"Active"}, V{+}{+}, T.\text{add}(x.dt)), e, x) \tag{2g}\\
\delta_{int}(phase, \sigma) &= (\text{"passive"}, \sigma) \tag{2h}\\
(phase, \sigma) &= X.\text{bag.pop}() \tag{2i}
\end{align}
$$

IV. Acknowledgments

Street network obtained from OpenStreetMaps (c) OSM contributors.

References

[1] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29–41, Feb 1996.

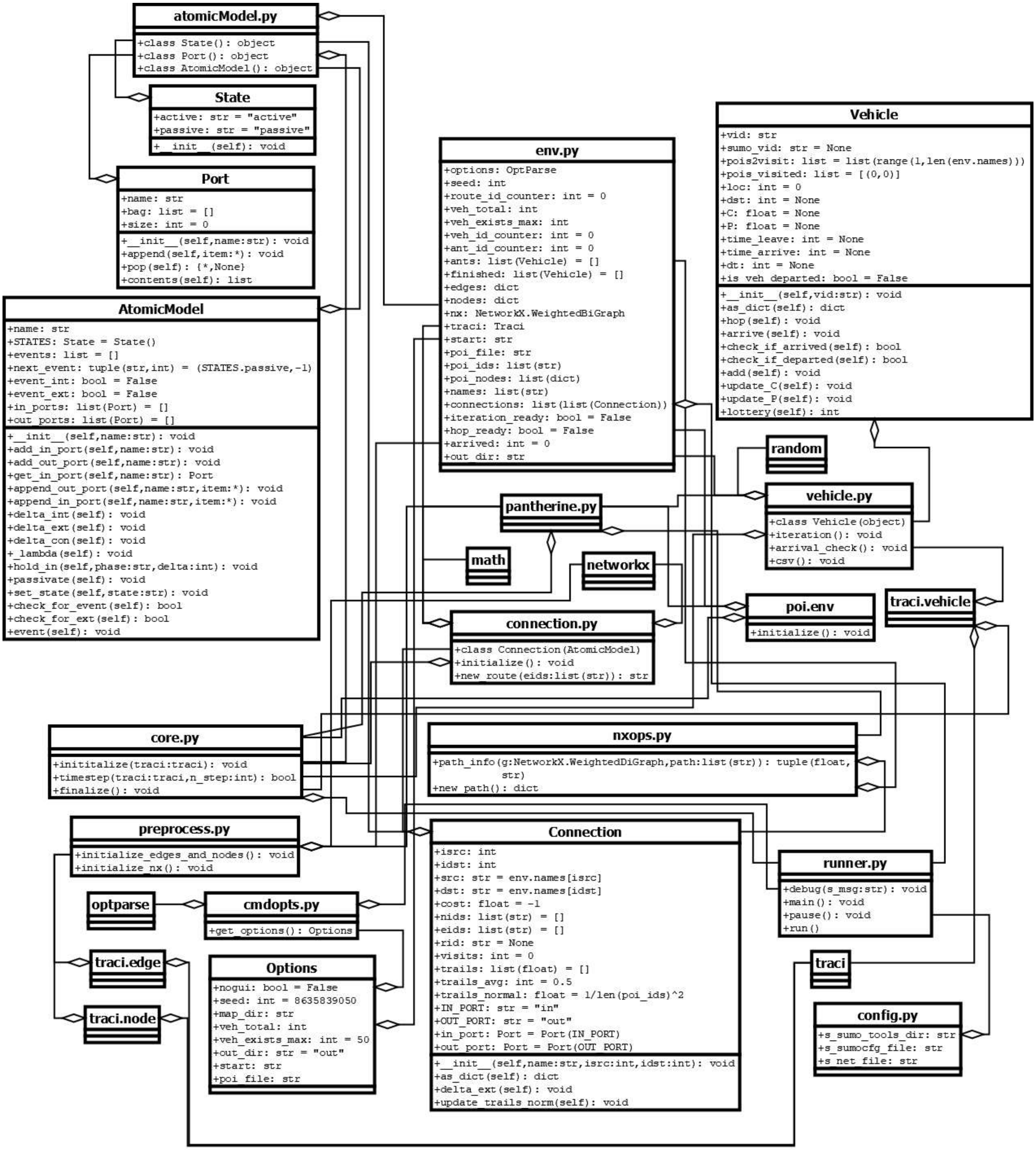
[2] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, December 2012.

[3] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux, "Traci: An interface for coupling road traffic and network simulators," in *Proceedings of the 11th Communications and Networking Simulation Symposium*, ser. CNS '08. New York, NY, USA: ACM, 2008, pp. 155–163. [Online]. Available: http://doi.acm.org/10.1145/1400713.1400740

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[5] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2019.

[6] Ant colony optimization algorithms. [Online]. Available: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms


atomicModel.py
+class State(): object
+class Port(): object
+class AtomicModel(): object
State
+active: str = "active"
+passive: str = "passive"
+__init__(self): void
Port
+name: str
+bag: list = []
+size: int = 0
+__init__(self,name:str): void
+append(self,item:*): void
+pop(self): {*,None}
+contents(self): list
AtomicModel
+name: str
+STATES: State = State()
+events: list = []
+next_event: tuple(str,int) = (STATES.passive,-1)
+event_int: bool = False
+event_ext: bool = False
+in_ports: list(Port) = []
+out_ports: list(Port) = []
+__init__(self,name:str): void
+add_in_port(self,name:str): void
+add_out_port(self,name:str): void
+get_in_port(self,name:str): Port
+append_out_port(self,name:str,item:*): void
+append_in_port(self,name:str,item:*): void
+delta_int(self): void
+delta_ext(self): void
+delta_con(self): void
+_lambda(self): void
+hold_in(self,phase:str,delta:int): void
+passivate(self): void
+set_state(self,state:str): void
+check_for_event(self): bool
+check_for_ext(self): bool
+event(self): void
env.py
+options: OptParse
+seed: int
+route_id_counter: int = 0
+veh_total: int
+veh_exists_max: int
+veh_id_counter: int = 0
+ant_id_counter: int = 0
+ants: list(Vehicle) = []
+finished: list(Vehicle) = []
+edges: dict
+nodes: dict
+nx: NetworkX.WeightedBiGraph
+traci: Traci
+start: str
+poi_file: str
+poi_ids: list(str)
+poi_nodes: list(dict)
+names: list(str)
+connections: list(list(Connection))
+iteration_ready: bool = False
+hop_ready: bool = False
+arrived: int = 0
+out_dir: str
Vehicle
+vid: str
+sumo_vid: str = None
+pois2visit: list = list(range(1,len(env.names)))
+pois_visited: list = [(0,0)]
+loc: int = 0
+dst: int = None
+C: float = None
+P: float = None
+time_leave: int = None
+time_arrive: int = None
+dt: int = None
+is_veh_departed: bool = False
+__init__(self,vid:str): void
+as_dict(self): dict
+hop(self): void
+arrive(self): void
+check_if_arrived(self): bool
+check_if_departed(self): bool
+add(self): void
+update_C(self): void
+update_P(self): void
+lottery(self): int
random
vehicle.py
+class Vehicle(object)
+iteration(): void
+arrival_check(): void
+csv(): void
pantherine.py
math
networkx
poi.env
+initialize(): void
traci.vehicle
connection.py
+class Connection(AtomicModel)
+initialize(): void
+new_route(eids:list(str)): str
core.py
+inititalize(traci:traci): void
+timestep(traci:traci,n_step:int): bool
+finalize(): void
nxops.py
+path_info(g:NetworkX.WeightedDiGraph,path:list(str)): tuple(float, str)
+new_path(): dict
preprocess.py
+initialize_edges_and_nodes(): void
+initialize_nx(): void
Connection
+isrc: int
+idst: int
+src: str = env.names[isrc]
+dst: str = env.names[idst]
+cost: float = -1
+nids: list(str) = []
+eids: list(str) = []
+rid: str = None
+visits: int = 0
+trails: list(float) = []
+trails_avg: int = 0.5
+trails_normal: float = 1/len(poi_ids)^2
+IN_PORT: str = "in"
+OUT_PORT: str = "out"
+in_port: Port = Port(IN_PORT)
+out_port: Port = Port(OUT_PORT)
+__init__(self,name:str,isrc:int,idst:int): void
+as_dict(self): dict
+delta_ext(self): void
+update_trails_norm(self): void
runner.py
+debug(s_msg:str): void
+main(): void
+pause(): void
+run()
optparse
cmdopts.py
+get_options(): Options
Options
+nogui: bool = False
+seed: int = 8635839050
+map_dir: str
+veh_total: int
+veh_exists_max: int = 50
+out_dir: str = "out"
+start: str
+poi_file: str
traci.edge
traci.node
traci
config.py
+s_sumo_tools_dir: str
+s_sumocfg_file: str
+s_net_file: str