Fine-grained Trajectory Reconstruction by Microscopic Traffic Simulation with Dynamic Data-driven Evolutionary Optimization

Htet Naing

I. Introduction

This document contains the supplementary materials of our paper, *"Fine-grained Trajectory Reconstruction by Microscopic Traffic Simulation with Dynamic Data-driven Evolutionary Optimization"*, submitted to the IEEE Transactions on Intelligent Transportation Systems (T-ITS). The paper provides supplementary information regarding the parameters of different optimization models used in our experiments. All our models were implemented in Python. SUMO [1] was adopted as our microscopic traffic simulation (MTS) model. Its Python interface, TraCI API, was used for communicating with SUMO during the simulation run-time. Parallel multi-processing with N CPU cores was used to evaluate N sets of simulation parameters simultaneously. Pytorch[1] was used to implement our surrogate model.

II. Simulation Parameters

The parameters of the SUMO's behavioral models – namely *Intelligent Driver Model (IDM)* [2] for car-following, and *LC2013* [3] for lane-changing, each having 5 and 11 constant parameters – are to be calibrated for trajectory reconstruction. Thus, this defines the simulation parameters λ with 16 dimensions. The measurement units used were primarily in second(s) and meter(m). In Table I, more information about these parameters is described with reference to SUMO's documentation[2]. The names of the parameters as well as the descriptions are directly derived from the documentation to avoid any confusion. For each parameter, their valid ranges are also provided in the table. During optimization, the optimal solution is searched over the feasible region defined by these ranges under a limited computing budget. Some other parameters involved in the car-following model were set as follows: delta=4; emergencyDecel=9; stepping=1.

III. Parameters for Dynamic Data-driven Evolutionary Optimization (D3EO)

A. Parameters for D3EO Algorithm

This section corresponds to Algorithm 1 in our paper. The parameters involved in the D3EO Algorithm defined in our paper were set as follows: Maximum evaluation budget B=100,000; Number of replication runs r=5; Number of Latin Hypercube Sampling (LHS) samples N_{LHS} = 10,000; Population size N is dependent on a chosen model; Elite size k=1.

B. Parameters for Genetic Algorithm

This section corresponds to Algorithm 2 in our paper. The parameters involved in the Genetic Algorithm defined in our paper were set as follows: Number of parents N_{pr}=0.5N where N is the population size; Number of offspring N_{op}=N; Crossover rate R_{cx}=0.5; Mutation rate R_m=0.5; Parameter change step size γ=0.1.

As for selection operation, a 2-way tournament selection with replacement was used as follows. First, two solutions in the population were randomly selected. Then, the one with the higher fitness score was declared the winner, and thus it was selected as a parent. This selection operation was executed until N_{pr} parents were already selected for subsequent operations.

As for crossover operation, a 1-point crossover was used as follows. First, two parents were randomly selected. Then, with R_{cx} probability, the crossover between the two parents would take place. In the case of crossover occurrence, a single crossover point was randomly determined. After that, the first portion of the chromosome before the crossover point would be taken from the first parent while the second portion after the crossover point, from the second parent. In the case of no crossover occurrence, one of the parents was randomly chosen as the offspring. The crossover operation was executed until N_{op} offspring were already produced.

As for mutation operation, each gene in the chromosome of an offspring was selected for mutation with R_m probability. In the case of mutation, the uniform random noise generated within a pre-defined range would be added to the selected genes. Otherwise, no changes were needed for a particular gene. After the mutation operation, some solutions may result in an invalid solution range, and thus, such solutions are clipped to be within the range.

C. Parameters for Surrogate

Artificial Neural Network (ANN) was chosen as our surrogate model. Its architecture was implemented as follows.

- Input: A feature vector of size 16.
- Layer 1: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.

Htet Naing is with the School of Computer Science & Engineering, Nanyang Technological University, Singapore (email: htetnain001@e.ntu.edu.sg)

[1]https://pytorch.org/

[2]https://sumo.dlr.de/docs/index.html

TABLE I: Simulation model parameters selected for calibration

No.	Parameter Name	Parameter Type	Description	Range [LB, UB]
1	maxSpeed	Car-following	The vehicle's (technical) maximum velocity. (in ms^{-1})	[10, 33.3333]
2	tau	Car-following	The driver's desired (minimum) time headway. (s)	[0.3, 6]
3	minGap	Car-following	Minimum gap when standing. (m)	[1, 5]
4	accel	Car-following	The acceleration ability of vehicles of this type. (in ms^{-2})	[0.28, 3.41]
5	decel	Car-following	The deceleration ability of vehicles of this type. (in ms^{-2})	[0.47, 3.41]
6	lcStrategic	Lane-changing	The eagerness for performing strategic lane changing. Higher values result in earlier lane-changing.	[0, 100]
7	lcCooperative	Lane-changing	The willingness for performing cooperative lane changing. Lower values result in reduced cooperation.	[0, 1]
8	lcSpeedGain	Lane-changing	The eagerness for performing lane changing to gain speed. Higher values result in more lane-changing.	[0, 100]
9	lcKeepRight	Lane-changing	The eagerness for following the obligation to keep right. Higher values result in earlier lane-changing.	[0, 100]
10	lcOvertakeRight	Lane-changing	The probability for violating rules against overtaking on the right.	[0, 1]
11	lcLookaheadLeft	Lane-changing	Factor for configuring the strategic lookahead distance when a change to the left is necessary.	[0, 100]
12	lcSpeedGainRight	Lane-changing	Factor for configuring the threshold asymmetry when changing to the left or to the right for speed gain.	[0, 100]
13	lcSpeedGainLookahead	Lane-changing	Lookahead time in seconds for anticipating slow down.	[0, 100]
14	lcKeepRightAcceptanceTime	Lane-changing	Time threshold for changing the willingness to change right. The value is compared against the anticipated time of unobstructed driving on the right. Lower values will encourage keepRight changes.	[0, 100]
15	lcCooperativeSpeed	Lane-changing	Factor for cooperative speed adjustments.	[0, 1]
16	lcAssertive	Lane-changing	Willingness to accept lower front and rear gaps on the target lane.	[1, 100]

- Layer 2: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *Linear* layer with 32 neurons, activated by *ReLU* function.
- Output: A scalar fitness score.

The training data were taken from 10,000 samples (generated via the LHS method) with their corresponding fitness scores. The data were first randomly shuffled and then, 90% of the data were used as the training data, and 10% were used as the validation data. This validation data were used to identify the model with the lowest validation error throughout all training epochs to be used as a surrogate. The training hyperparameters were set as follows: Number of training epochs=3000; Batch size=128; Objective function=Mean squared error loss; Optimizer=Adam with a learning rate of 0.0001.

D. Parameters for SAnE Algorithm

This section corresponds to Algorithm 3 in our paper. The parameters involved in the SAnE Algorithm defined in our paper were set as follows: Number of SAnE Sampling times S is dependent on a chosen model; Number of surrogate-assisted evolution steps n=10. The same training hyperparameters were used for both offline and online surrogate training.

IV. Parameters for Model Comparison

The following parameter settings were used for different models compared in our experiments:

- **DEFAULT**: Refer to our paper.
- **RANDOM**: Refer to our paper.
- **GA**: Follow the setting described in Section III-B on top of the information already provided in our paper.
- **DDPG**: Refer to Section IV-A below on top of the information already provided in our paper.
- **D3GA**: S=1, n=10 with the GA setting in Section III-B and the ANN setting in Section III-C on top of the information already provided in our paper.
- **D3GA++**: Same as D3GA except S=30 on top of the information already provided in our paper.

A. Deep Deterministic Policy Gradient (DDPG)

The DDPG is an off-policy actor-critic deep reinforcement learning model. Its implementation closely follows the original work [4]. It was used for parameter calibration as follows. At each iteration, a population of parameters was fed to the actor as the states. Each solution in the population was separately treated as one state vector for the agent. Then, the actor determined corresponding changes (perturbed with exploratory uniform-random noises) to these parameters as its output actions. The changes were then added to respective parameters in the population. After that, any invalid parameter values were clipped to be within their valid ranges. These new solutions were considered as the new states seen by the agent. Then, the updated population with new solutions was evaluated with the real objective function evaluation by running the simulation. Their fitness scores were treated as rewards for the agent. Then, through elitism, the worst solution in the current population was replaced with the best solution found so far. Next, the new training samples in the form of (states, actions, new states, rewards) were added to the replay memory. The replay memory was initially pre-filled with the same samples generated via the LHS method for training the ANN surrogate. Finally, before proceeding to the next iteration, both actor and critic networks' parameters were updated by learning through the samples collected in the replay memory. The learning involves selecting a batch of samples randomly from the replay memory, updating the critic network using these samples first, and then updating the actor network using both the updated critic network and the samples. This learning loop was executed 10 times at each iteration. The target networks were updated by directly copying the parameters of the actor and critic networks every five iterations. This iterative process continued until the limited computing budget ran out, after which the best solution found was used as the final calibrated parameters for the simulation model.

The training hyperparameters were set as follows: Discount factor=0.25; Optimizer=Adam with a learning rate of 0.0005 for both actor and critic networks; Batch size=128; Replay memory size=7000; Exploration noise=Same as mutation

noise used in Section III-B.

The actor network was implemented with the following architecture:

- Input: A feature vector of size 16.
- Layer 1: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *Linear* layer with 32 neurons, activated by *Tanh* function.
- Output: Parameter changes $\in [-1, 1]$ (with dimension=16).
- Scaled Output: Scale the output towards the range $[-\Delta\lambda, \Delta\lambda]$; where $\Delta\lambda = 0.1\left[\frac{\lambda_{UB}-\lambda_{LB}}{2}\right]$.

The critic network was implemented with the following architecture:

- Input: A feature vector of size 16 (The input is the sum of state and action vectors instead of the usual concatenated state and action vectors).
- Layer 1: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 2: *Linear* layer with 32 neurons, followed by *layer normalization*, activated by *ReLU* function.
- Layer 3: *Linear* layer with 32 neurons.
- Output: A scalar Q value.

REFERENCES

[1] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo–simulation of urban mobility: an overview," in *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[2] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[3] J. Erdmann, "Lane-changing model in sumo," *Proceedings of the SUMO2014 modeling mobility with open data*, vol. 24, pp. 77–88, 2014.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.