Transportation Research Part C 128 (2021) 103091

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

TrajGAIL: Generating urban vehicle trajectories using generative adversarial imitation learning☆,☆☆,☆☆☆,☆☆☆☆

Seongjin Choi [a], Jiwon Kim [b], Hwasoo Yeo [a,*]

[a] Department of Civil and Environmental Engineering, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon, Republic of Korea
[b] School of Civil Engineering, The University of Queensland, St Lucia, Queensland, Australia

ARTICLE INFO

Keywords:
Urban vehicle trajectories
Trajectory data analysis
Trajectory data generation
Generative model
Generative adversarial imitation learning

ABSTRACT

Recently, an abundant amount of urban vehicle trajectory data has been collected in road networks. Many studies have used machine learning algorithms to analyze patterns in vehicle trajectories to predict location sequences of individual travelers. Unlike the previous studies that used a discriminative modeling approach, this research suggests a generative modeling approach to learn the underlying distributions of urban vehicle trajectory data. A generative model for urban vehicle trajectories can better generalize from training data by learning the underlying distribution of the training data and, thus, produce synthetic vehicle trajectories similar to real vehicle trajectories with limited observations. Synthetic trajectories can provide solutions to data sparsity or data privacy issues in using location data. This research proposes *TrajGAIL*, a generative adversarial imitation learning framework for the urban vehicle trajectory generation. In TrajGAIL, learning location sequences in observed trajectories is formulated as an imitation learning problem in a partially observable Markov decision process. The model is trained by the generative adversarial framework, which uses the reward function from the adversarial discriminator. The model is tested with both simulation and real-world datasets, and the results show that the proposed model obtained significant performance gains compared to existing models in sequence modeling.

1. Introduction

Rapid advancements in location sensing and wireless communication technology enabled us to collect and store a massive amount of spatial trajectory data, which contains geographical locations of moving objects with their corresponding passage times (Lee and Krumm, 2011). Over the last decade, considerable progress is made in collecting, pre-processing, and analyzing trajectory data. Also, the trajectory data analysis is applied in various research areas, including behavioral ecology (De Cáceres et al., 2019), transportation engineering (Wu et al., 2018), and urban planning (Laube, 2014).

☆ This research was supported by the Ministry of Land, Infrastructure, and Transport (MOLIT, KOREA) Project ID: 21TLRP-B146733-04, Project Name: Connected and Automated Public Transport Innovation (National R&D Project).☆☆ All implementations of the model and the training algorithm is available at https://github.com/benchoi93/TrajGAIL.☆☆☆ The authors appreciate Seoul Metropolitan Government and Dr. Min Ju Park for providing the taxi DTG data of Seoul city.☆☆☆☆ This article belongs to the Virtual Special Issue on IG005572: VSI:Machine learning.

* Corresponding author.
E-mail addresses: benchoi93@kaist.ac.kr (S. Choi), jiwon.kim@uq.edu.au (J. Kim), hwasoo@kaist.ac.kr (H. Yeo).

https://doi.org/10.1016/j.trc.2021.103091
Received 21 August 2020; Received in revised form 11 March 2021; Accepted 14 March 2021
Available online 3 May 2021
0968-090X/Crown Copyright © 2021 Published by Elsevier Ltd. All rights reserved.

In transportation engineering, urban vehicle trajectory data are collected based on the location sensors installed inside vehicles or at the roadside and analyzed with various methods. The high-resolution mobility data of individual users in urban road networks offer unprecedented opportunities to understand vehicle movement patterns in urban traffic networks. It provides rich information on both aggregated flows and disaggregated travel behaviors. The aggregated flows include the origin-destination (OD) matrix and cross-sectional link traffic volumes. The disaggregated travel behaviors include user-centric travel experiences, namely, speed profile, link-to-link route choice behavior and travel time experienced by individual vehicles, as well as system-wide spatio-temporal mobility patterns, such as origin-destination pairs, routing pattern distributions, and network traffic states (Kim and Mahmassani, 2015).

Most of the studies in the vehicle trajectory data analysis use machine learning methods. The recurrent neural network, for example, is used by many previous researchers due to its ability to learn sequential information in trajectory data. In machine learning, there are mainly two approaches to modeling: the *discriminative* and the *generative* modeling. A discriminative model learns a direct map from input X to output (label) Y or posterior probability $P(Y|X)$, which is the conditional probability of each label Y given the input variable X. It only learns the decision boundaries between labels and does not care about the underlying distribution of data. In contrast, a generative model captures the underlying probability distribution, i.e., joint probability $P(X, Y)$, from which $P(Y|X)$ can be computed. One advantage of a generative model is that we can generate new (synthetic) data similar to existing data by sampling from $P(X, Y)$.

Synthetic data generation based on generative models has gained increasing importance as the data generation process plays a significant role in various research fields in an era of data-driven world (Popić et al., 2019). It is mainly used to serve two purposes. The first purpose is to deal with the lack of real data. In many research fields, data collection is costly, and, therefore, it is often difficult to collect enough data to properly train and validate models. In this case, it is useful to generate synthetic data that are similar to the real observations to increase training and test samples. The second purpose is to address the issue with the privacy and confidentiality of real data. Many types of data contain personal information, such as gender, name, and credit card usage. Synthetic data can be combined with or replace such privacy-sensitive data with a reasonable level of similarity, thereby protecting privacy while serving the intended analysis.

Urban vehicle trajectory analysis has both challenges: data sparsity and data privacy issues. Although the sources and availability of urban trajectory data are increasing, most of the currently available trajectory datasets cover only a portion of all vehicles in the network. From network management and operations perspectives, there is a desire to infer vehicle trajectories that represent the whole population to have a more complete view of traffic dynamics and network performance. Moreover, urban vehicle trajectory data may contain personal information of individual drivers, which poses serious privacy concerns in relation to the disclosure of private information to the public or a third party (Chow and Mokbel, 2011). The ability to generate synthetic trajectory data that can realistically reproduce the population mobility patterns is, therefore, highly desirable and expected to be increasingly beneficial to various applications in urban mobility.

While synthetic trajectory data generation is a relatively new topic in transportation research communities, there are several existing research areas that have addressed similar problems. One example is *trajectory reconstruction*. When two points in a road network are given as an initial point (treated as sub-origin) and a target point (treated as sub-destination), the models reconstruct the most plausible route between the two points. The trajectory reconstruction can be considered as generating trajectories between sub-origins and sub-destinations. Previous studies such as Chen et al. (2011) and Hu et al. (2018) investigated on discovering the most popular routes between two locations. Chen et al. (2011) first constructs a directed graph to simplify the distribution of trajectory points and used the Markov chain to calculate the transfer probability to each node in the directed graph. The transfer probability is used as an indicator to reflect how popular the node is as a destination. The route popularity is calculated from the transfer probability of each node. Hu et al. (2018) also used a graph-based approach to constructing popular routes. The check-in records which contain the route's attributes are analyzed to divide the whole space into zones. Then, the historical probability is used to find the most plausible zone sequences. Also, Feng et al. (2015) and Rao et al. (2018) estimated origin-destination patterns by using trajectory reconstruction. Both studies used particle filtering to reconstruct the vehicle trajectory between two points in automatic vehicle identification data. The reconstructed vehicle trajectory is then used to estimate the real OD matrix of the road network. Another problem that is relevant to trajectory generation is the *next location prediction* problem, where the "next location" of a subject vehicle is predicted based on the previously visited locations of the subject vehicle. Monreale et al. (2009), for example, presented a decision tree to predict the next location based on the previously visited locations. Decision-tree based models, however, occasionally overfit the training dataset and lack the generalization ability to produce diverse trajectory patterns. Gambs et al. (2012) used Mobility Markov chain (MMC) to predict the next location among the clustered points or Point-of-Interests (POIs). The POIs considered in Gambs et al. (2012) are home, work, and other activity locations to model human activity trajectories throughout the day, rather than vehicle movement trajectories reflecting link-to-link vehicle driving behavior considered in this study. Choi et al. (2019b) used a feed-forward neural network to predict the next intersection in a grid-structured road network. A set of intersections in Brisbane, Australia are treated as POI's to capture the link-to-link route choice behavior. Jin et al. (2019) used an augmented-intention recurrent neural network model to predict locations of vehicle trajectories of individual users. Jin et al. (2019) incorporated additional information on individual users' historical records of frequently visited locations into a next location prediction model. The past visited locations in historical records are represented as edge-weighted graph, and graph convolution network is used to incorporate this information into trajectory prediction. In Choi et al. (2018), an urban road network is partitioned into zones based on the clustering of trajectory data points, and the prediction model based on recurrent neural network (RNN) is proposed to predict the zone that the subject vehicle would visit. Choi et al. (2019a) extended the idea of predicting the next zone and used network traffic state information to improve the RNN model's performance.

In fact, the existing models developed for the next location prediction problem can be applied for synthetic trajectory data generation. By sequentially applying the next location predictions, a synthetic vehicle trajectory can be generated. However, most of the existing models for next location prediction adopt a discriminative modeling approach, where the next locations are treated as labels and the model is trained to predict one or two next locations. The discriminative models have limitations in generating full trajectories, especially when sample trajectory data are sparse. it is only the decision boundaries between the labels that the models are trained to predict, not the underlying distributions of data that allow proper generalization for sampling realistic trajectories. As a result, it is necessary to develop a model based on the generative modeling approach to successfully perform synthetic trajectory data generation.

In this paper, we apply *imitation learning* to develop a generative model for urban vehicle trajectory data. Imitation learning is a subdomain of reinforcement learning for learning sequential decision-making behaviors or "policies". Unlike reinforcement learning that uses "rewards" as signals for positive and negative behavior, imitation learning directly learns from sample data, so-called "expert demonstrations," by imitating and generalizing the expert' decision-making strategy observed in the demonstrations. If we consider an urban vehicle trajectory as a sequence of decisions for choosing road links along a travel path, imitation learning can be applied to develop a generator that can reproduce synthetic data by imitating the decision-making process (i.e., driver' route choice behavior) demonstrated in the observed trajectory dataset. One approach to imitation learning is called *Inverse Reinforcement Learning* (IRL), which aims to recover a reward function that explains the behavior of an expert from a set of demonstrations. Using the recovered expert reward function as feedback signals, the model can generate samples similar to the expert' decisions through reinforcement learning. Ziebart et al. (2008a) and Ziebart et al. (2008b) used maximum entropy IRL (MaxEnt) to generate synthetic trajectories similar to a given taxi dataset. One of the advantages of using IRL is that the model generates trajectories using both current states and expected returns of future states to determine an action—as opposed to considering only the knowledge up to the current state (e.g., previous visited locations)—, thereby enabling a better generalization of travel behavior along the whole trajectory.

Recently, there have been remarkable breakthroughs in generative models based on deep learning. In particular, Goodfellow et al. (2014) introduced a new generative model called Generative Adversarial Networks (GAN), which addressed inherent difficulties of deep generative models associated with intractable probabilistic computations in training. GANs use an adversarial *discriminator* to distinguish whether a sample is from real data or from synthetic data generated by the *generator*. The competition between the generator and the discriminator is formulated as a minimax game. As a result, when the model is converged, the optimal generator would produce synthetic sample data similar to the original data. The generative adversarial learning framework is used in many research fields such as image generation (Radford et al., 2015), audio generation (Oord et al., 2016), and molecular graph generation (De Cao and Kipf, 2018).

GANs have been also applied in transportation engineering. Zhang et al. (2019a) proposed trip travel time estimation framework called *T-InfoGAN* based on generative adversarial networks. They used a dynamic clustering algorithm with Wasserstein distance to make clusters of link pairs with similar travel time distribution, and they applied Information Maximizing GAN (InfoGAN) to travel time estimation. Xu et al. (2020) proposed Graph-Embedding GAN (GE-GAN) for road traffic state estimation. Graph embedding is applied to select the most relevant links for estimating a target link and GAN is used to generate the road traffic state data of the target link. In Li et al. (2020), GAN is used as a synthetic data generator for GPS data and travel mode label data. To solve the sample size problem and the label imbalance problem of a real dataset, the authors used GAN to generate fake GPS data samples of each travel mode label to obtain a large balanced training dataset. The generative adversarial learning framework is also used for synthetic trajectory generation. Liu et al. (2018) proposed a framework called trajGANs. Although this paper does not include specific model implementations, it discusses the potential of generative adversarial learning in synthetic trajectory generation. Inspired by Liu et al. (2018),Rao et al. (2020) proposed LSTM-TrajGAN with specific model implementations. The generator of LSTM-TrajGAN is similar to RNN models adopted in the next location prediction studies.

This study proposes *TrajGAIL*, a generative adversarial imitation learning (GAIL) model for urban vehicle trajectory data. GAIL, proposed by Ho and Ermon (2016), uses a combination of IRL's idea that learns the experts' underlying reward function and the idea of the generative adversarial framework. GAIL effectively addresses a major drawback of IRL, which is high computational cost. However, the standard GAIL has limitations when applied to the vehicle trajectory generation problem because it is based on the IRL concept that only considers a vehicle's current position as states in modeling its next locations (Ziebart et al., 2008a; Ziebart et al., 2008b; Zhang et al., 2019b), which is not realistic as a vehicle's location choice depends on not only the current position but also the previous positions. To overcome these limitations, this study proposes a new approach that combines a partially-observable Markov decision process (POMDP) within the GAIL framework. POMDP can map the sequence of location observations into a latent state, thereby allowing more generalization of the state definition and incorporating the information of previously visited locations in modeling the vehicle's next locations. In summary, the generation procedure of urban vehicle trajectories in TrajGAIL is formulated as an imitation learning problem based on POMDP, which can effectively deal with sequential data, and this imitation learning problem is solved using GAIL, which enables trajectory generation that can scale to large road network environments.

This paper is organized as follows. Section 2 describes the methodology of this paper. A detailed problem formulation is presented in Section 2.1, and the proposed framework of TrajGAIL is presented in Section 2.2. Section 3 describes how the performance of the proposed model is evaluated. Section 3.1 introduces the data used in this study, and Section 3.2 introduces the baseline models for performance comparison. In Section 3.3, the evaluation results are presented at both trajectory-level and dataset-level. Finally, Section 4 presents the conclusions and possible future research.

2. Methodology

The objective of TrajGAIL is to generate location sequences in urban vehicle trajectories that are similar to real vehicle travel paths observed in a road traffic network. Here, the “similarity” between the real vehicle trajectories and the generated vehicle trajectories can be defined from two different perspectives. First, the *trajectory-level* similarity measures the similarity of an individual trajectory to a set of reference trajectories. For instance, the probability of accurately predicting the next locations—single or multiple consecutive locations as well as the alignment of the locations—are examples of trajectory-level similarity measures. Second, the *dataset-level* similarity measures the statistical or distributional similarity over a trajectory dataset. This type of measure aims to capture how closely the generated trajectory dataset matches the statistical characteristics such as OD and route distributions in the real vehicle trajectory dataset. In this section, we present the modeling framework of TrajGAIL, where the procedure of driving in a road network is formulated as a partially observable Markov decision process to generate realistic synthetic trajectories, taking into account the similarities defined above.

2.1. Problem formulation

Let $Traj = \{(x_1, y_1, t_1), \cdots, (x_N, y_N, t_N)\}$ be an urban vehicle trajectory, where (x_i, y_i, t_i) is the (x, y)-coordinates and timestamp t for the i^{th} point of the trajectory, and $LocSeq = \{(x_1, y_1), \cdots, (x_N, y_N)\}$ be the location sequence of $Traj$. When location points (x, y) are continuous latitude and longitude coordinates, it is necessary to pre-process these coordinates and match them to a predefined set of discrete locations. Previous studies used different ways of defining discrete locations. For instance, Choi et al. (2018),Choi et al. (2019a), and Ouyang et al. (2018) used partitioned networks, so-called cells or zones, while Choi et al. (2019b) and Ziebart et al. (2008a) used road links to represent trajectories. In this paper, we represent a trajectory as a sequence of links to model link-to-link route choice behaviors in urban road networks. The location sequence of each vehicle trajectory is, thus, transformed to a sequence of link IDs by link matching function f_M:

$$LinkSeq = (l_1, \cdots, l_M) = f_M(LocSeq = \{(x_1, y_1), \cdots, (x_N, y_N)\}) \tag{1}$$

where l_j is the link ID of the j^{th} visited link along the trajectory. The goal of this study is to generate the link sequence of a trajectory by modeling and learning the probability distribution of $LinkSeq, P(LinkSeq) = P(L_1 = l_1, \cdots, L_M = l_M)$ for a discrete random variable L in all possible set of link IDs. Modeling this joint probability distribution is, however, extremely challenging, as also noted in the previous studies (Choi et al., 2018; Ouyang et al., 2018). A way to resolve this problem is to use a sequential model based on the Markov property, which decomposes the joint probability to the product of conditional probabilities as follows:

$$\begin{aligned} P(LinkSeq) &= P(L_1 = l_1, \cdots, L_M = l_M) \\ &= P(L_M = l_M | L_{M-1} = l_{M-1}, \cdots, L_1 = l_1) \times \cdots \times P(L_2 = l_2 | L_1 = l_1) \cdot P(L_1 = l_1) \\ &= P(L_M = l_M | L_{M-1} = l_{M-1}) \times \cdots \times P(L_2 = l_2 | L_1 = l_1) \times P(L_1 = l_1) \end{aligned} \tag{2}$$

The problem of modeling vehicle trajectories using this Markov property can be formulated as a Markov Decision Process (MDP). An MDP is a discrete-time stochastic control process based on the Markov property (Howard, 1960). This process provides a mathematical framework for modeling sequential decision making of an agent. An MDP is defined with four variables: (S, A, T, R), where S is a set of states that the agent encounters, A is a set of possible actions, $T(s, a, s')$ is a transition model determining the next state $(s' \in S)$ given the current state $(s \in S)$ and action $(a \in A)$, and $R(s, a)$ is a reward function that gives the agent the reward value (feedback signal) of its action given the current state. If the transition is stochastic, transition model $T(s, a, s')$ can also be denoted as $P(s'|s, a)$. A policy (π_θ) is defined as a θ-parameterized function that maps states to an action in the deterministic case $(\pi_\theta(s) \to a)$, or a function that calculates the probability distribution over actions $(\pi_\theta(s) = P(a|s))$ in the stochastic case. The objective of MDP's optimization is to find the optimal policy that maximizes the expected cumulative rewards, which is expressed as:

$$\pi_{\theta^*} = \underset{\theta}{\operatorname{argmax}} \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t \cdot R(s, a)\right] \quad a \sim Categorical(\pi_\theta(s) = P(a|s)) \tag{3}$$

where π_{θ^*} is the optimal policy with parameter θ^*, a is a sampled action from $\pi_\theta(s) = P(a|s)$ (in this study, we use discrete action space, so actions are sampled from categorical distribution), and γ is the discount rate of future rewards.

How to define the four variables of MDP is critical to the successful training of a policy model. The states (s) should incorporate enough information so that the next action (a) is determined based only on the current state (s), and the transition model T should correctly reflect the transition of states in the environment it models. Finally, the reward function R should give a proper training signal to the agent to learn the optimal policy.

In TrajGAIL, the vehicle movement in a road network is formulated as an MDP. We set road segments or links as states and transitions between links as actions. In this case, the transition model can be defined as a deterministic mapping function that gives the next link (s') given the current link (s) and the link-to-link movement choice (a), i.e., $T : (s, a) \mapsto s'$. The policy represents a driver's route choice behavior associated with selecting the next link at each intersection. This road network MDP, thus, produces vehicle trajectories—more specifically, link sequences—as a result of sequential decision making modeled by this policy.

As mentioned above, MDP assumes that the action (a) is determined based only on the current state (s). However, it is likely that vehicles' link-to-link movement choice is affected by not only the current location but also the previous locations. Moreover, vehicle movements in a road network is a result of complex interactions between a large number of drivers and road environment such as the generation and distribution of trips and the assignment of the routes and, therefore, the link choice action cannot be determined solely by road segment information alone as a state. The model needs more information such as origin, destination, trip purpose, and the prevailing traffic state. Incorporating all such information in the state definition, however, makes the problem intractable due to an extremely large state space. It is, thus, desirable to relax the assumption such that the action is determined based on the current state as well as some *unobservable* states.

This can be achieved by employing a partially observable MDP (POMDP). A POMDP assumes that an MDP determines the model dynamics, but the agent cannot directly observe the underlying states. Instead of directly using the states as MDP, POMDP uses a surrogate state such as probability distribution over the set of possible states (Kaelbling et al., 1998) and belief state (Rao, 2010). Fig. 1 shows the graphical model of POMDP. It is assumed that there exist latent unobservable states $s^* \in S^*$. We can only partially observe s^* through observation $o \in O$. Using o or sequence of o's, latent state s^* is estimated and this estimated latent state, or the belief state, is represented as $s \in S$.

As a result, instead of four variables of MDP, POMDP uses five variables (O, S, A, T, R), where O represents the set of possible observations. The belief state at time state t,s_t, is estimated based on the sequence of o representing all observations up to the current time t, which is assumed to be the estimate of the latent unobservable state, $\hat{s}_t^*$, as follows:

$$s_t = f(o_1, \ldots, o_t) = \hat{s}_t^* \qquad s \in S, o \in O \tag{4}$$

In TrajGAIL, the observation space, O, is defined as the ID of links in the road network and two virtual tokens representing the start and the end of a trip (*Start*,*End*). Actions A are transitions between links. In Ziebart et al. (2008a), the set of actions includes all possible link transitions. However, this can lead to a very large action space even with a moderate-sizes network with hundreds of links, requiring high computational cost. To reduce the computational complexity, we instead define a set of common actions that represent possible movements between two connected links, namely, [*Straight*,*Left*,*Right*,*Terminate*], where *Straight*,*Left*, and *Right* represent the movement direction at the end of each link (at intersections) and *Terminate* represents the termination of a trip (i.e., a vehicle reached its destination). These four actions are sufficient for our current study as we consider a grid-structured network, where all intersections are four-way intersections. However, it is also possible to model general networks with more diverse intersection structures such as five-way or T-shape intersections as we can apply a "mask" that allows flexibility to further define specific actions available for each link, which would be a subset of the network-wide common action set. For instance, one can define six actions for a network with the maximum intersection size of six and specify only a subset of available actions for each link if it has less than six connected roads.

To summarize, we formulate a partially observable Markov Decision Process to develop a generative model for vehicle trajectories, which produces the optimal policy describing optimal actions given a sequence of observations.

2.2. Model framework

2.2.1. Preliminaries and background - imitation learning

In this study, the imitation learning framework is used to develop a generative model represented in POMDP formulation. Imitation learning is a learning problem that aims to train a model that can act like a given expert. Usually, demonstrations of decisions of the expert are given as a training dataset. In this study, a real vehicle trajectory dataset serves as expert demonstrations so that the model learns the decision-making process of vehicle movements in a road network observed in the given dataset. There are mainly two categories of approaches in imitation learning: *behavior cloning* and *inverse reinforcement learning*.

Behavior cloning considers the imitation learning problem as a supervised learning problem. In behavior cloning, given the expert demonstrations, the state and action sequence is divided into independent state-action pairs and a model is trained to directly learn the relationship between input (state) and output (action) based on these sample pairs. The biggest advantage of behavior cloning is simplicity. However, because of its simplicity, the model fails to make proper generalization in complex tasks. Simple generative models based on Markov Chain Gambs et al. (2010) and Recurrent Neural Networks Choi et al. (2018, 2019a, 2016) can be classified into this category of imitation learning.

The inverse reinforcement learning (IRL) uses an indirect approach. The objective of IRL is to find the reward function that the agent is optimizing given the measurements of agents' behavior and sensory inputs to the agents (Russell, 1998). It is assumed that the experts follow certain rules known as a reward function. The main idea of IRL is to learn this reward function to imitate the experts

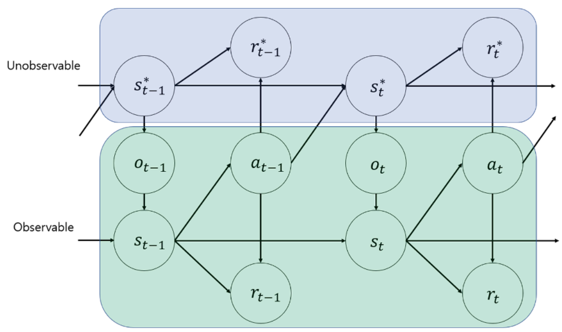


Fig. 1. Partially observable Markov Decision Process.

based on the history of experts' behaviors in certain situations. It is called "inverse" reinforcement learning because it learns the reward function that represents the experts' decisions from their states and actions, whereas the reinforcement learning (RL) learns to generate states and actions from a given reward function. Some of the key papers on IRL problems include Ng et al. (2000), Abbeel and Ng (2004), Ziebart et al. (2008a), Wulfmeier et al. (2015), Ho and Ermon (2016), which readers are referred to for more details on IRL.

Given expert policy π_E, the objective of IRL is to find a reward function (r) that maximizes the difference between the expected rewards from the expert and the RL agent ($\mathbf{E}_{\pi_E}[r(s,a)] - RL(r)$) such that the expert performs better than all other policies (Ho and Ermon, 2016), where an expectation with respect to a policy, $\mathbf{E}_{\pi}[r(s,a)]$, is used to denote an expectation with respect to the trajectory it generates (i.e., γ-discounted cumulative reward), $\mathbf{E}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\right]$. This is achieved by minimizing the expected reward from RL agent ($RL(r)$) and by maximizing the expected reward from the expert ($\mathbf{E}_{\pi_E}[r(s,a)]$), while minimizing the reward regularizer ($\psi(r)$). On the other hand, when a reward function (r) is given, the objective of RL is to find a policy (π) that maximizes the expected reward ($\mathbf{E}_{\pi}[r(s,a)]$) while maximizing the entropy of the policy ($H(\pi)$).

$$IRL\left(\pi_E\right) = \arg\min_{r\in\mathbf{R}}(\psi(r) + RL(r) - \mathbf{E}_{\pi_E}[r(s,a)]) \tag{5}$$

where,

$$RL(r) = \max_{\pi\in\Pi}(H(\pi) + \mathbf{E}_{\pi}[r(s,a)]) \tag{6}$$

where $\mathbf{R}$ is the largest possible set of reward functions ($\mathbf{R} = \{r : S\times A \longrightarrow \mathbb{R}\}$), $\psi(r)$ is the convex reward function regularizer, and $H(\pi) = \mathbf{E}_{\pi}[-\log\pi(a|s)]$ is the causal entropy of the policy π (Ho and Ermon, 2016).

It is interesting to investigate the relationship between RL and IRL in Eq. (5) and Eq. (6). RL tries to find the optimal policy π that maximizes the expected rewards, and IRL tries to find the optimal reward function that maximizes the difference between expert policy (π_E) and RL's policy (π). In some sense, RL can be interpreted as a generator that creates samples based on the given reward, and IRL can be interpreted as a discriminator that distinguish the expert policy from RL's policy. This relationship is similar to the framework of Generative Adversarial Networks (GAN). GANs use an adversarial *discriminator* (D) that distinguishes whether a sample is from real data or from synthetic data generated by the *generator* (G). The competition between generator and discriminator is formulated as a minimax game. As a result, when the model is converged, the optimal generator would produce synthetic sample data similar to the original data. Eq. (7) shows the formulation of minimax game between D and G in GANs.

$$\min_G\max_D\left(\mathbf{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbf{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]\right) \tag{7}$$

With a proper selection of the regularizer $\psi(r)$ in IRL formulation in Eq. (5), Ho and Ermon (2016) proposed *generative adversarial imitation learning* (GAIL). The formulation of minimax game between the discriminator (D) and the policy (π) is shown in Eq. (5),

$$\min_D\max_\pi(\mathbf{E}_{\pi}[\log D(s,a)] + \mathbf{E}_{\pi_E}[\log(1 - D(s,a))] - \lambda H(\pi)) \tag{8}$$

Eq. (8) can be solved by finding a saddle point (π,D). To do so, it is necessary to introduce function approximations for π and D since both π and D are unknown functions and it is very difficult, if not impossible, to define a exact function form for them. Nowadays, deep neural networks are widely used for function approximation. By computing the gradients of the objective function with respect to the corresponding parameters of π and D, it is possible to train both generator and discriminator through backpropagation. In the implementation, we usually take gradient steps for π and D alternatively until both networks converge.

While GAIL provides a powerful solution framework for synthetic data generation, the original GAIL model Ho and Ermon (2016) could not be directly used for our problem of vehicle trajectory generation. From our experiments, we found that the standard GAIL tends to produce very long trajectories with many loops, indicating that vehicles are constantly circulating in the network. This is because the generator in GAIL tries to maximize the expected cumulative rewards and creating a longer trajectory can earn higher expected cumulative rewards as there is no penalty of making a trajectory longer.

There are several ways to address this issue. Possible approaches include giving a negative reward whenever a link is visited to penalize a long trajectory or using positional embedding (the number of visited links) as state. However, a better approach would be to let the model know the vehicle's visit history (a sequence of links visited so far) and learn that it is unrealistic to visit the same link over and over. Our proposed TrajGAIL framework achieves this and addresses the limitation of GAIL in trajectory generation by assuming POMDP and using RNN embedding layer.

2.2.2. TrajGAIL: generative adversarial imitation learning framework for vehicle trajectory generation

TrajGAIL uses POMDP to formulate vehicle trajectory generation as a sequential decision making problem and GAIL to perform imitation learning on this POMDP to learn patterns in observed trajectories to generate synthetic vehicle trajectories similar to real trajectories. In vehicle trajectory generation, it is important to take into account not only the previous locations of the trajectories, but also the expected future locations that the trajectory is expected to visit. By using POMDP, TrajGAIL considers how realistic the previously visited locations are. By using GAIL, TrajGAIL can also consider how realistic future locations would be because the imitation learning framework in GAIL uses an objective function to maximize the expected cumulative future rewards when generating new actions, which captures how realistic the remaining locations will be. It is noted that, in this study, TrajGAIL focuses on generating location sequences (link sequences) of trajectories without considering time components. Throughout the paper, we use the term *trajectory generation* to refer to the generation of link sequences representing trajectory paths for the sake of brevity. Fig. 2 shows the model framework of TrajGAIL. As in GAIL, TrajGAIL consists of the discriminator and the generator, where the discriminator gives reward feedback to the vehicle trajectories generated by the generator until both converge. The generator works as a reinforcement learning agent, and the discriminator works as an inverse reinforcement learning agent. Below we provide more details on each of these two modules.

The Generator of TrajGAIL. The primary role of the generator is to make realistic synthetic vehicle trajectories. The generator creates N trajectories by a policy roll-out, or an execution of a policy from initial state (*Start* of trip) to terminal state (*End* of trip). A trajectory starts with the virtual token *Start*. By sequentially applying the policy generator until the current observation reaches the other virtual token *End*, the generator produces a whole vehicle trajectory. As our problem is formulated as a POMDP, we need to map the sequence of observations into the latent states. Rao (2010) suggested that the belief states (s_t), the estimate of probability distribution over latent states, can be computed recursively over time from the previous belief state. The posterior probability of state i at the t-th observation, denoted by $s_t(i)$ can be calculated as follows:

$$\begin{aligned} s_t(i) &= P\left(s_t^* = i | o_t, a_{t-1}, o_{t-1}, \cdots, a_0, o_0\right) \\ &\propto P\left(o_t | s_t^* = i\right) \cdot P\left(s_t^* = i | a_{t-1}, o_{t-1}, \cdots, a_0, o_0\right) \\ &\propto P\left(o_t | s_t^* = i\right) \cdot \sum_j T\left(j, a_{t-1}, i\right) s_{t-1}(j) \end{aligned} \tag{9}$$

where s_t^* is the latent states at the t-th observation including both observable and unobservable variables, $P(o|s^*)$ is the probability of observation $o$given s^*, and $T(s, a, s')$ is the transition model that maps the current state (s) and the action (a) to the next state (s').

Eq. (9) indicates that the current belief state vector (s_t) is a combination of the information from the current observation ($P(o_t|s_t^*)$) and the feedback from the previous computation of the belief state (s_{t-1}). In Rao (2010), the author recognized the similarity between the structure of this equation and recurrent neural networks (RNN) and suggested using RNN for belief state estimation. Many previous studies on the next location prediction problem suggest that RNNs show great performance in embedding the sequence of locations into

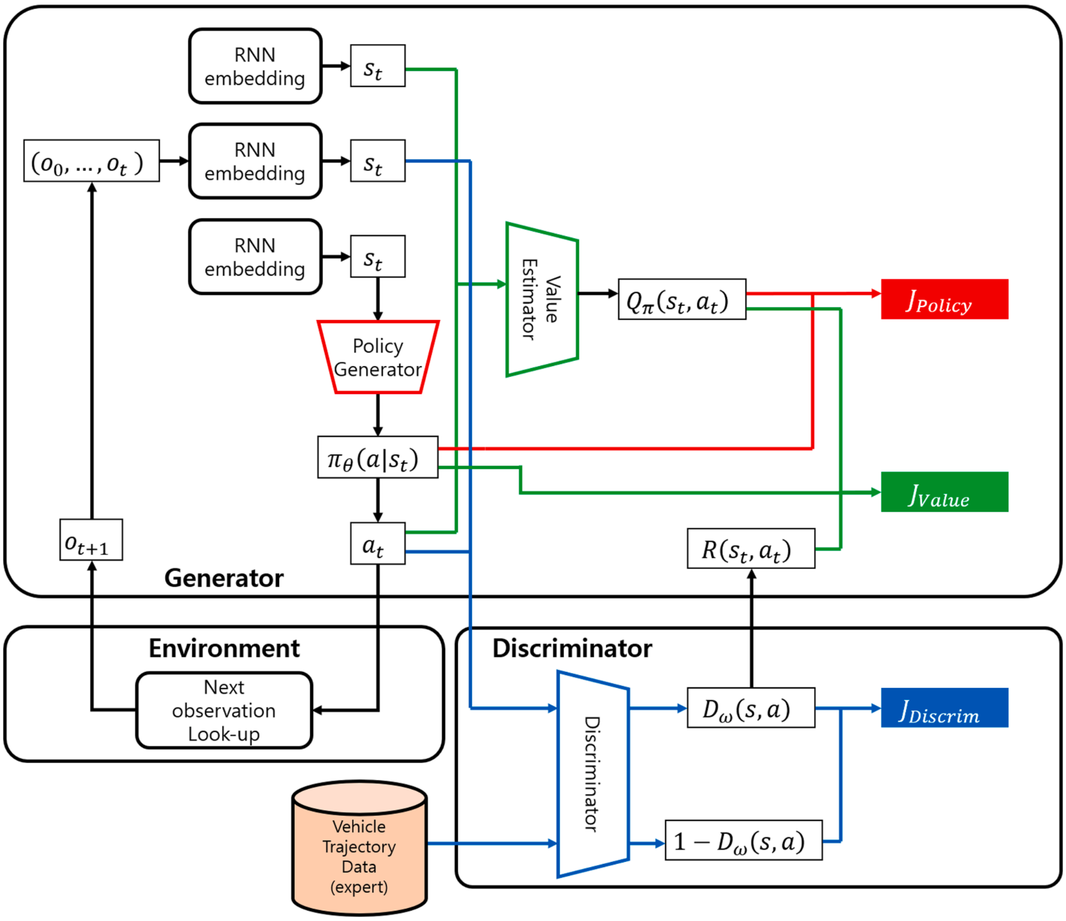


Fig. 2. The model framework of TrajGAIL.

a vector (Choi et al., 2018; Choi et al., 2019a; Feng et al., 2018). Accordingly, we use an RNN embedding layer to map the sequence of observations (link IDs) to a belief state vector. Since the entire historical sequence is embedded in the current (belief) state via RNN embedding and the actions are still determined based only on the current state, the Markov assumption in MDP is not violated, while sequential information can be effectively captured within the model. In the implementation of RNN embedding layer, the size of input tensor (observation sequence) is $[B \times L]$ and the size of output tensor (belief state vector) is $[B \times H]$, where B is the batch size, L is the maximum observation sequence length in the batch, and H is the number of hidden neurons.

Based on the belief state vector (s_t), the *policy generator* within the TrajGAIL generator module calculates the probability of the next action ($\pi(a|s_t)$). The policy $\pi(a|s_t)$ has a size of $[B \times A]$, where A is the size of action space. The next action is sampled from a multinomial distribution with the probability ($\pi(a|s_t)$). The next observation is determined by the *next observation look-up* table of the road network environment, $T_o(o_t, a_t, o_{t+1})$. The next observation loop-up table maps the current observation (o_t) and the action (a_t) to the next observation (o_{t+1}). This should be defined based on the map geometry data which contains information on connections between links. This process continues until the current observation reaches the virtual token *End*.

In reinforcement learning, a *value function* is often used to calculate the expected return of the actions at the current state. Here, we use a state-action value function $Q_\pi(s,a)$, which is estimated via the *value estimator* in TrajGAIL's generator. The state-action value function, $Q_\pi(s,a)$, has size of $[B \times 1]$ since it represents a value scalar of each input observation sequence. The value estimator has a separate RNN embedding layer to process the sequence of observations into the belief state. Based on the processed belief state and a given action, the value estimator calculates the expected return of the action at current belief state. The estimated value, or the expected return, is used as a coefficient when updating the policy generator. If the estimated value of a given action is large, the policy generator model is reinforced to give the similar actions more often. This value estimator is also modeled as a deep neural network, which is trained to minimize the value objective function, J_{Value}, defined as follows:

$$\begin{aligned} J_{Value} &= \mathbf{E}\left[(Q_\pi(s_t,a_t) - G(s_t,a_t))^2\right] \\ &= \mathbf{E}\left[\left(Q_\pi\left(s_t,a_t\right) - \left(R\left(s_t,a_t\right) + \gamma\cdot\mathbf{E}\left[R\left(s_{t+1},a_{t+1}\right)\right] + \gamma^2\cdot\mathbf{E}\left[R\left(s_{t+2},a_{t+2}\right)\right] + \ldots\right)^2\right] \\ &= \mathbf{E}\left[(Q_\pi(s_t,a_t) - (R(s_t,a_t) + \gamma(\mathbf{E}[R(s_{t+1},a_{t+1})] + \gamma\cdot\mathbf{E}[R(s_{t+2},a_{t+2})] + \ldots))^2\right] \\ &= \mathbf{E}\left[(Q_\pi(s_t,a_t) - (R(s_t,a_t) + \gamma\cdot\mathbf{E}[Q_\pi(s_{t+1},a_{t+1})]))^2\right] \\ &= \mathbf{E}\left[\left(Q_\pi\left(s_t,a_t\right) - \left(R\left(s_t,a_t\right) + \gamma\cdot\sum\pi\left(a_{t+1}\middle|s_{t+1}\right)\cdot Q_\pi\left(s_{t+1},a_{t+1}\right)\right)\right)^2\right] \end{aligned} \tag{10}$$

where a mean squared error (MSE) loss between the value estimate ($Q_\pi(s,a)$) and the actual γ discounted return ($G(s,a)$) is used.

The objective of the policy update is to maximize the expected cumulative reward function as shown in Eq. (3). We define the policy objective as $J(\theta)$ and we maximize the policy objective to improve the policy generator at every iterations. In order to compute the gradient of the policy objective, we use the Policy Gradient Theorem (Sutton et al., 2000). In the Policy Gradient Theorem, for any differentiable θ-parameterized policy π_θ, the policy gradient of policy objective $\nabla_\theta J_{PG}(\theta)$ is given as:

$$\nabla_\theta J_{PG}(\theta) = \mathbf{E}\left[\nabla_\theta \log\pi_\theta(a|s)\cdot Q_{\pi_\theta}(s,a)\right] = \nabla_\theta J_{Policy} \tag{11}$$

where the last equality indicates that the gradient of J_{PG} is equal to the gradient of J_{Policy} given by $J_{Policy} = \mathbf{E}\left[\log\pi_\theta(a|s)\cdot Q_{\pi_\theta}(s,a)\right]$ although J_{PG} and J_{Policy} themselves are different. As their gradients are the same, we use J_{Policy} in the training procedure.

Additionally, we add an entropy maximization objective (Ziebart et al., 2008a; Ho and Ermon, 2016) to the policy objective $J(\theta)$ in order to prevent the policy from converging to a local optimal policy. Often the global optimal policy is difficult to learn because of the sparsity of reward function. In this case, the policy would converge to a local optimal policy that only generates a limited variety of trajectories without considering the underlying distribution of actions. The entropy maximization objective helps counteract this tendency, guiding the policy to learn the underlying distribution of actions and eventually learn the underlying distribution of trajectories. As a result, we use the following equation to update the parameters of the policy generator to maximize the policy objective.

$$\begin{aligned} &J(\theta) = J_{PG}(\theta) + \lambda H(\pi_\theta) \\ &\nabla_\theta J\left(\theta\right) = \nabla_\theta J_{PG}\left(\theta\right) + \lambda\nabla_\theta H\left(\pi_\theta\right) = \nabla_\theta J_{Policy} + \lambda\nabla_\theta H\left(\pi_\theta\right) \end{aligned} \tag{12}$$

The Discriminator of TrajGAIL. The discriminator solves the classification problem by distinguishing real vehicle trajectories from generated vehicle trajectories. As the generator gets improved to create more realistic vehicle trajectories, the discriminator's ability to classify the generated trajectories from the real trajectories is also improved through iterative parameter updates and fine-turning. This competition of two neural networks is the fundamental concept of the generative adversarial learning framework. In the discriminator update step, the samples from the real vehicle trajectory dataset are labeled as 0, and the samples from the generator are labeled as 1. For both real and generated vehicle trajectories, we put the sequence observation and the action taken at the last observation as an input, process the sequence of observation into a belief state through an RNN embedding layer, and calculate the probability ($D_\omega(s,a)$) that the given sequence of observations and the action are from the generator. The discriminator probability, $D_\omega(s, a)$, has size of $[B \times 1]$ similar to $Q_\pi(s,a)$. The parameters of the discriminator are updated to minimize the binary cross-entropy loss. The ω-parameterized discriminator is updated to minimize the discriminator objective, $J_{Discrim}$, with the following gradient term.

$$\nabla_\omega J_{Discrim} = \left[\mathbf{E}_{(s,a)\sim\pi_\theta}[\nabla_\omega \log(D_\omega(s,a))] + \mathbf{E}_{(s,a)\sim\pi_E}[\nabla_\omega \log(1 - D_\omega(s,a))]\right] \tag{13}$$

The discriminator gives the training signal to the generator through the reward function ($R(s,a)$) as shown in Fig. 2. The generator is trained to maximize the binary cross-entropy loss of the discriminator. As the second term of Eq. (13) is irrelevant to the parameters of the generator, the objective of the generator is to maximize the first term of Eq. (13). As a result, the reward function is defined as follows:

$$R(s,a) = -\log(D_\omega(s,a)) \tag{14}$$

RNN embedding layer. In Fig. 2, it is noted that the policy generator, the value estimator, and the discriminator use their own RNN embedding layers to embed the sequence of observations to the latent states. Using a separate RNN embedding layer for each of these three modules enables each module to interpret the sequence of observations and update RNN parameters in such a way as to maximize its performance in the task given to the module. For example, the discriminator might have a different interpretation on the sequence states from the policy generator, because the discriminator might focus more on the information on the whole sequence such as trajectory length to execute a discriminative task, while the policy generator might focus more on the current position to decide the next link-to-link transition.

It is possible to implement TrajGAIL with a single shared RNN embedding layer. We implemented and tested this structure, but the results showed that using a shared RNN embedding layer produced a high variance in the three objectives in the early stage of training, which can lead to the failure of training. This is because the three objectives are updating the RNN embedding layer in different directions so that updating one objective is affecting the other objectives in unintended ways.

Backpropagation. Fig. 3 shows the schema of backpropagation to update parameters in the whole TrajGAIL framework. There are three different objective functions, J_{Policy}, J_{Value}, and $J_{Discrim}$, used to update parameters associated with policy generator, value estimator, and discriminator, respectively. Each objective is backpropagated to update only the related parameters in the deep neural networks. The backpropagation route of each objective function is indicated in different colors, i.e., red arrows for policy objective (J_{Policy}), green arrows for value objective (J_{Value}), and blue arrows for discriminator objective ($J_{Discrim}$). The backpropagation routes through the RNN embedding layers in Fig. 3 represent that each of these three modules updates its own RNN embedding layer. These RNN embedding layers play an integral role in combining the POMDP concept within the GAIL framework, which are the main distinction between TrajGAIL and stardard GAIL models.

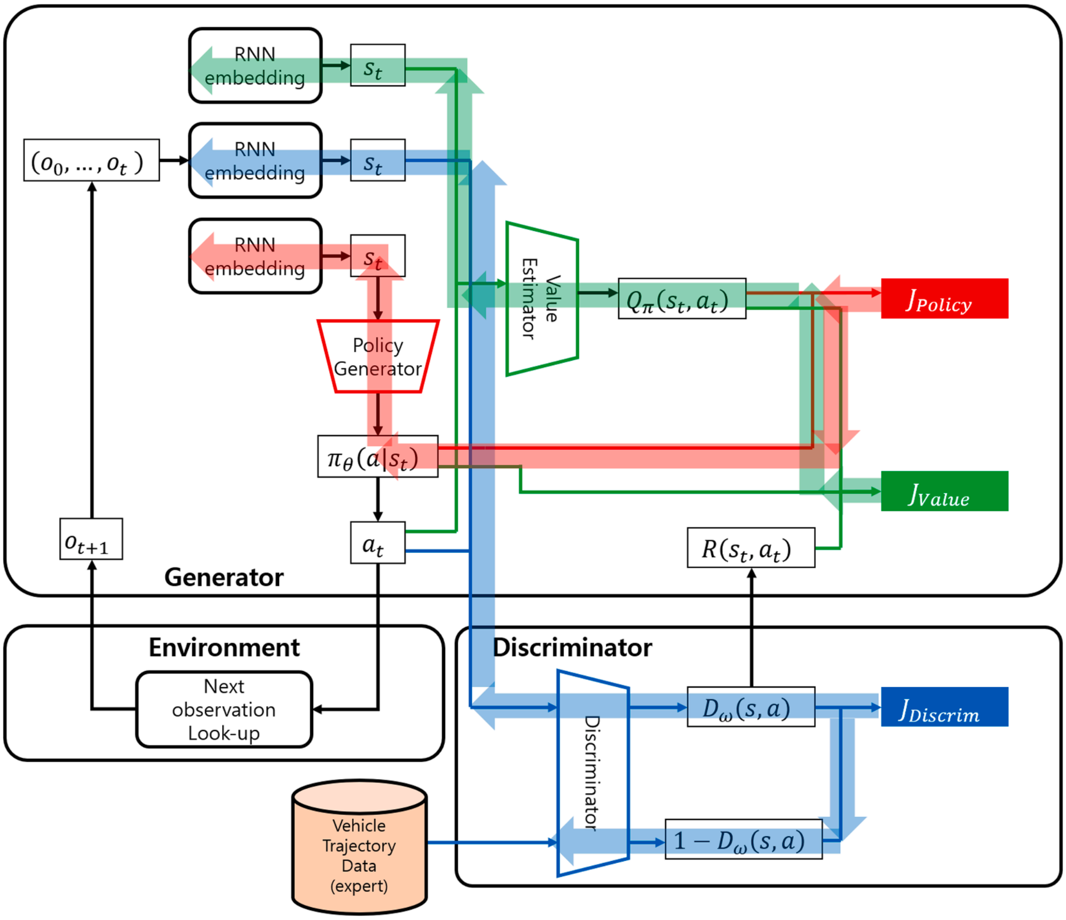


Fig. 3. Backpropagation schema of TrajGAIL (red arrows for policy objective, green arrows for value objective, and blue arrows for discriminator objective). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Training Techniques. During the implementation of generative adversarial network algorithms, there are several techniques that can be used to avoid training failure and facilitate model convergence. The first technique used in this study is to maintain similar learning levels between the discriminator and the generator while training. If one overpowers the other, a proper competition cannot be formed and, thus, the model cannot learn from the competition mechanism of generative adversarial networks. Options to balance the learning levels include setting different learning rates and/or using different numbers of update-steps for training the generator and the discriminator. In this study, we used the same learning rate for both discriminator and generator and used a different number of update-steps. At each iteration, the generator is updated six times, while the discriminator is updated twice, because the discriminator usually learned faster in this study. The second technique is to use a sufficient number of trajectories generated for training. At each iteration, a certain number of trajectories are created and used to update the parameters of the model. If the number of generated trajectories is small, the model may result in a problem called "mode collapse." The mode collapse is defined as a case where the generator collapses, producing a limited variety of data (Dumoulin et al., 2016; Lin et al., 2018). Sometimes the generator oscillates among a few data points without converging to the equilibrium. In our case, the mode collapse leads the generator to produce trajectories for a few specific routes only. Using a sufficient number of sample trajectories at each training iteration solved this problem in our case. The details of the sample size used in this study are provided in the description of the evaluation results below.

3. Performance evaluation

3.1. Data

The performance of TrajGAIL was evaluated based on two different datasets. The first dataset is a virtual vehicle trajectory dataset generated by a microscopic traffic simulation model, AIMSUN. The second dataset includes the data collected by the digital tachograph (DTG) installed in the taxis operating in Gangnam District in Seoul, South Korea.

AIMSUN uses dynamic traffic assignment (Barceló and Casas, 2005) to select the appropriate route for each vehicle. We can select five different route choice models: Binomial, C-Logit, Proportional, Multinomial Logit, and Fixed. The first four algorithms use a predefined cost function and sample a route with the corresponding random distribution. The last algorithm only considers the travel time in free-flow condition and makes greedy choices, in which most vehicles use the route with shortest travel time for each OD.

The first dataset consists of data with three different demand patterns. The first demand pattern is called "**Single-OD**" pattern. The Single-OD pattern has only one origin source and one destination sink as shown in Fig. 4. The origin source is connected to the Link 252, and the destination sink is connected to the Link 442. There are six possible shortest path candidates.

The second and third demand patterns use multiple origins and destinations. In these cases, the vehicle sources are connected to all 12 links that are at the boundary of the network, whose directions are towards the inside of the network (Link 252, 273, 298, 302, 372, 443, 441, 409, 430, 392, 321, 245). The remaining 12 boundary links, whose directions are towards the outside of the network, (Link 253, 276, 301, 299, 376, 447, 442, 400, 420, 393, 322, 246) are connected to the vehicle sinks. There can be 132 origin-destination pairs, excluding direct U-turns from the origin such as Link 245 to Link 246. The second demand pattern is called "**One-way Multi-OD**"

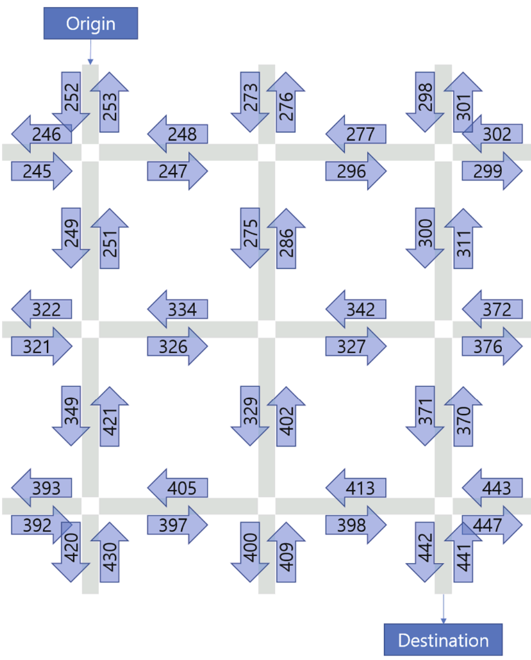


Fig. 4. Simulated traffic network in AIMSUN environment. The origin and destination location is for Single-OD pattern.

pattern, where we assume there are major demand flows from the origin links on the left (Link 245, 321, 392) to the destination links on the right (Link 299, 376, 447) as shown in the Fig. 5a. The major flows include all combinations of the origins (left) and destinations (right). The third demand pattern is called "**Two-way Multi-OD**" pattern, shown in Fig. 5b, where major demand flows from the origin links on the right (Link 302, 372, 443) to the destination inks on the left (Link 246, 322, 393) are added to the One-way Multi-OD pattern.

The second dataset is collected by the DTG installed in taxis. The taxis operating in Seoul city are installed with digital tachographs (DTG) and collect the driving records such as position (longitude and latitude), speed, and passenger occupancy. By linking the data points with the same taxi ID in chronological order, the raw data points are transformed into a taxi trajectory dataset. Among the Seoul taxi trajectories, the trajectories that passed Gangnam district are selected for this study. Gangnam district has major road links in a grid structure as shown in Fig. 6, so there are multiple choices in routes with similar travel distance for a given OD within the district. Since each taxi trajectory includes multiple trips associated different passengers, we extract a sub-trajectory covering the trip of each passenger and these passenger-level sub-trajectories are referred to as "**Gangnam**" dataset. The origin and destination of the vehicle trajectory in the Gangnam dataset represent the passenger OD demand in Gangnam district, and the routing patterns represent the route choice behaviors of the taxi drivers. As mentioned in Section 2.1, trajectory data need to be converted into link sequences. We use a map matching algorithm in Brakatsoulas et al. (2005) to map taxi trajectory data points to the underlying road links. After the data preparation, we obtained a total of 59,553 vehicle trajectories collected in Gangnam district on January 1st, 2018.

3.2. Baseline models

We tested TrajGAIL against three baseline models:

- Mobility Markov Chain

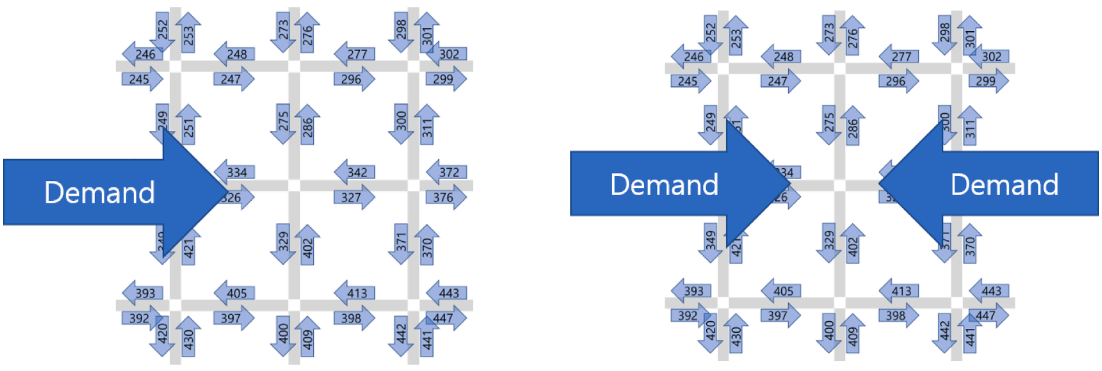
  Mobility Markov Chain (MMC) (Gambs et al., 2010) is one of the earliest models for the next location prediction problem based on the Markov model. MMC models each vehicle's trajectory as a discrete stochastic process, where the probability of moving to a next location depends on the immediate previous link observation.

$$P(o_{next} = i | o_{prev} = j) = \frac{N(o_{next} = i | o_{prev} = j)}{\sum_{\forall k \in O} N(o_{next} = k | o_{prev} = j)} \tag{15}$$

  where $N(o_{next} = i | o_{prev} = j)$ is the number of occurrences in the real vehicle trajectory dataset that a vehicle moves from link j to link i

- Recurrent Neural Network model for Next Location Prediction

  Several previous studies such as Liu et al. (2016, ?, ?) suggest that recurrent neural networks show good performance in predicting the next location by learning spatio-temporal features of trajectory data. When a vehicle trajectory is given, RNN cells repeatedly process and calculate the hidden state. The RNN cells decide which information to keep and which to forget. The RNN cell then calculates the probability of the next location. The cross-entropy loss is used to calculate the estimation error. When the RNN model is used as a generative model, an input vector, starting with the virtual token *Start*, is passed through RNN to compute the predictive probability over the possible next locations. One location is sampled with multinomial distribution, and the sampled next location is used as the next input vector. The procedure continues until the current location reaches the virtual token *End* representing the end of the trip. In this study, we use Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) for RNN cells.



(a) One-way Multi-OD (b) Two-way Multi-OD

Fig. 5. Multi-OD demand patterns. Blue arrows indicates the major demand flows. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

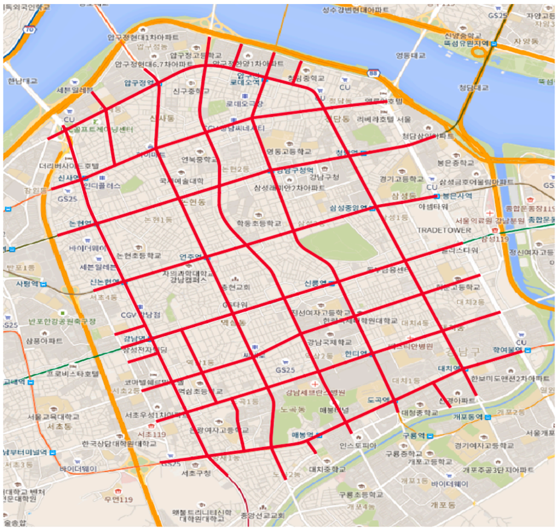


Fig. 6. Major road links in Gangnam district (Map data ©2020 SK Telecom).

- Maximum Entropy Inverse Reinforcement Learning

  Maximum Entropy IRL (MaxEnt) (Ziebart et al., 2008a) is one of the most widely used IRL models. MaxEnt uses a probabilistic approach based on the principle of maximum entropy to resolve the ambiguity in choosing distributions over decisions. MaxEnt uses a linear reward function for simplicity and uses a training strategy of matching feature expectations between the observed expert policy and the learner's behavior. In Ziebart et al. (2008a), the feature expectation is expressed in terms of expected *state visitation frequency*, meaning that the MaxEnt model calculates the expected number of visitation at each state (link in a road network in this study), and matches it with the actual number of visitation in the expert dataset. In this study, we extend the idea of matching state visitation to matching state-action visitations. We call the original MaxEnt model using state visitation frequency *MaxEnt(SVF)* and the new one using state-action visitation frequency *MaxEnt(SAVF)*.

3.3. Results

This section shows the training and testing results of the TrajGAIL and the baseline models. TrajGAIL and the baseline models are trained for AIMSUN and Gangnam datasets. And they are tested in various aspects with different performance measures. Two different evaluation levels are defined to measure trajectory-level similarity and dataset-level similarity. Section 3.3.1 shows the training result using convergence curves. Section 3.3.2 and Section 3.3.3 show the testing result based on the trajectory-level similarity and dataset-level similarity.

3.3.1. Training procedure

Hyperparameters. For the AIMSUN-based datasets, we first generate 20,000 vehicle trajectories for each demand scenario. Then, we split the total dataset into *training* and *testing* datasets in 0.7:0.3 ratio. As a result, we obtain 14,000 vehicle trajectories for training and 6,000 vehicle trajectories for testing. For the Gangnam dataset, we used the same ratio of 0.7:0.3, and this makes 41,687 vehicle trajectories for training and 9,866 vehicle trajectories for testing. The training dataset is only used to train each model, and all the results in Section 3.3.2 and Section 3.3.3 are based on the testing dataset. As mentioned in **Training Techniques** in Section 2.2.2, it is important to use sufficient number of sample trajectories at each training iterations. We tested different numbers of sample trajectories for each dataset and concluded that 2,000 sample trajectories are enough for the Single-OD dataset and 20,000 sample trajectories are

Table 1
Hyperparameters used for TrajGAIL.

Hyperparameter	Value
Number of iterations	20,000
Number of samples	20,000
Number of discriminator updates	2
Number of generator updates	6
Number of hidden neurons in each layer	64
Number of layers in RNN embedding	3
Learning rate	0.00005
Discount rate of reward (γ in Eq. (10))	0.95
Entropy coefficient (λ in Eq. (12))	0.01

enough for the Multi-OD datasets and Gangnam datasets. For a proper comparison of the model's performance, all models generate 20,000 sample trajectories for both trajectory-level evaluation and dataset-level evaluation. The more details on the hyperparameters used for training TrajGAIL is shown in Table 1.

Convergence Curve. Fig. 7 shows the convergence curve of the objective functions ($J_{Policy}, J_{Value}, J_{Discrim}$) and the causal entropy ($H(\pi_\theta)$) for the "One-way MultiOD Binomial" case based on the AIMSUN dataset. At the very beginning of the iterations (0 to 100 iterations), J_{Policy} increases and $J_{Discrim}$ decreases as the policy generator is designed to maximize J_{Policy} and the discriminator is designed to minimize $J_{Discrim}$. As the policy generator starts to produce more realistic trajectories, it gets difficult to discriminate from the perspective of the discriminator. As the discriminator starts to distinguish the real trajectories from the generated trajectories, it gets difficult for the generator to generate more realistic trajectories. As a result, J_{Policy} tends to decrease and $J_{Discrim}$ tends to increase in the middle of the iterations (100 to 1000 iterations). Afterwards, J_{Value} is almost converged to zero, and the generator and the discriminator makes small changes to win the minimax game. The entropy $H(\pi_\theta)$ is converged to maximize the causal entropy at this point. It is noticeable that $J_{Discrim}$ is converged to a value close to 1.38. According to Goodfellow et al. (2014), the discriminator objective ($J_{Discrim}$) converges to $log4 = 1.38629\cdots$ as the generator produces realistic outputs. This is because the discriminator cannot distinguish the real trajectories from the generated trajectories and, thus, the best strategy becomes a random guess, which gives 50:50 chance of getting it right.

Computation Time. Fig. 8 shows the computation time taken to generate 20,000 vehicle trajectories using five different models. This computation time is measured on a workstation with i9-10900KF CPU, 64 GB RAM, and Nvidia Geforce RTX 3080. MMC and MaxEnt models only used CPU when computation time is measured. The computation times of RNN and TrajGAIL are measured not only by using CPU, but also by using GPU. In Fig. 8, the shaded parts of the bar chart are the computation time gain by using parallel processing of GPU when using RNN and TrajGAIL. Overall, all five models have the capability to generate 20,000 vehicle trajectories in

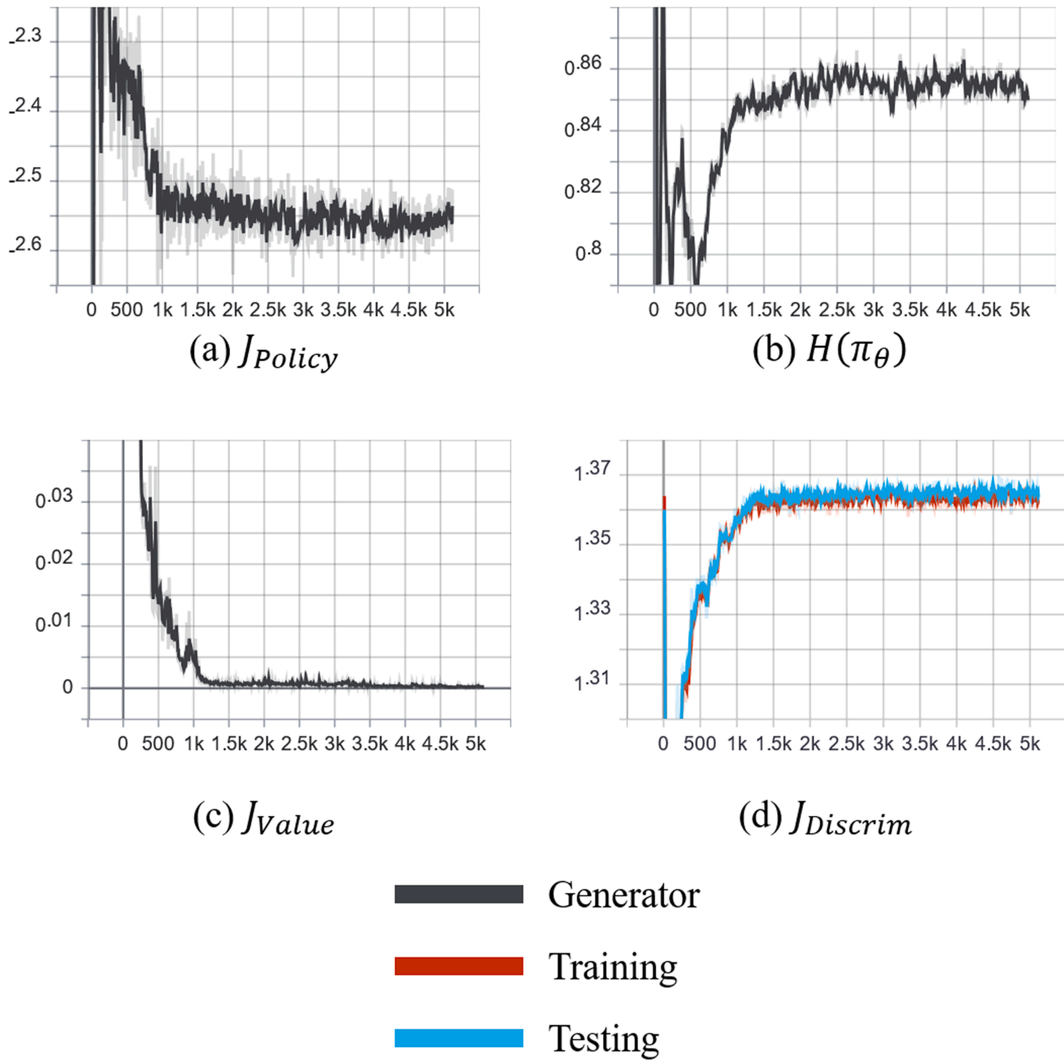


Fig. 7. Convergence curve of objective functions based on a sample case of "One-way Multi OD Binomial" using the AIMSUN dataset.

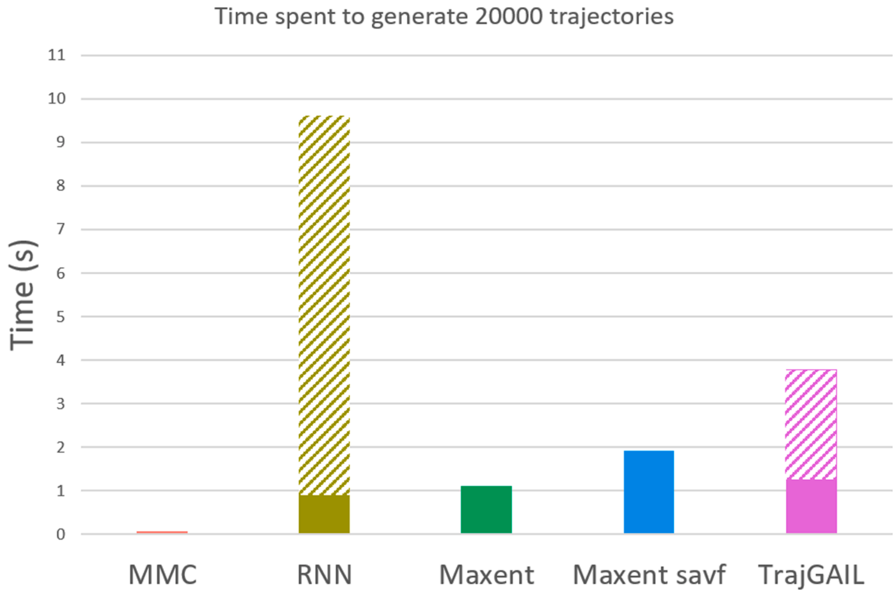


Fig. 8. Computation time taken to generate 20,000 vehicle trajectories.

less than 2 s.

Visualizing Estimated Values. To provide an intuition on how trajectories are generated in TrajGAIL, we visualize the estimated values of the state-action value function, $Q_\pi(s,a)$, in Fig. 9. Value function $Q_\pi(s,a)$ calculates the expected return of the actions at each state and these estimated values are used to update policy $\pi(a|s)$. During the training, the policy is shaped in such a way to select actions that have high estimated values. Fig. 9 shows the estimated values along one selected route in the **Single-OD Binomial** demand pattern in the AIMSUN dataset. All trajectories in the expert dataset have the same origin and same destination. There are six feasible routes that connect the given origin and destination, and Fig. 9 is based on one of the feasible routes. Along the route, there are five decision points (the intersections to choose the next link) and, at each intersection, there are three estimated values associated with the three possible actions: *Left* (*L*), *Straight* (*S*), and *Right* (*R*). Note that state *s* in $Q_\pi(s,a)$ represents link ID. Since $Q_\pi(s,a)$ is defined as the *expected discounted cumulative reward*, which is the sum of reward over the decision points between the current state and the end of a trajectory, the value of $Q_\pi(s,a)$ tends to decrease as an intersection is closer to the destination because there are fewer decision points over which the reward is summed. As such, the comparison of the estimated $Q_\pi(s,a)$ values is meaningful only within each intersection, not across intersections.

In Fig. 9, at the first intersection, the values of taking *Straight* and *Left* are higher than the value of taking *Right*. This is because taking *Right* here is unrealistic as there is no way that this next link is connected to the destination. On the other hand, taking *Straight* and taking *Left* have similarly high values because both choices are realistic (although *Straight* is the chosen action in this particular route, *Left* is also chosen in other feasible routes). At the second intersection, taking *Straight* and taking *Left* show higher values than taking *Right* and, at the third intersection, *Straight* and *Right* show higher values than *Left* (taking *Left* at the third intersection makes some kind of detour which is unrealistic). Similarly, the values are higher for *Right* at the fourth intersection and for *Straight* at the fifth intersection, which are the realistic action at each respective intersection. Overall, this demonstrates that TrajGAIL can capture realistic actions at each intersection and translate this information to the value function and policy to generate realistic trajectories.

3.3.2. Trajectory-level evaluation

In the trajectory-level evaluation, we measure how similar each generated vehicle trajectory is to a real trajectory. Two widely used evaluation metrics in sequence modeling are used to evaluate this trajectory-level similarity: BLEU score (Papineni et al., 2002) and METEOR score (Banerjee and Lavie, 2005).

In the next location prediction problem, it is common to use the probability of correctly predicting the next location to measure the model's performance. For example, in Choi et al. (2018), a complementary cumulative distribution function of the prediction probability is used to measure how accurately the model predicts the next one, two, or three consecutive locations. While this measure is intuitive and easy to interpret, it has a drawback that it only considers element-wise prediction accuracy and does not take the whole sequence into account. The element-wise performance measures can be sensitive to small local mispredictions and tend to underestimate the model's performance. As such, this study employs a BLEU score and METEOR score that consider the whole sequence. They are more robust and accurate as a performance measure for sequence modeling.

BLEU is one of the most widely used metrics in natural language processing and sequence-to-sequence modeling. When reference sequences are given, BLEU scans through the sequence and checks if the generated sequence contains identical chunks, or a contiguous sequence of *n* elements found in the reference sequences. Here, BLEU uses a modified form of precision to compare a reference sequence and a candidate generated sequence by clipping. For the generated sequence, the number of each chunk is clipped to a

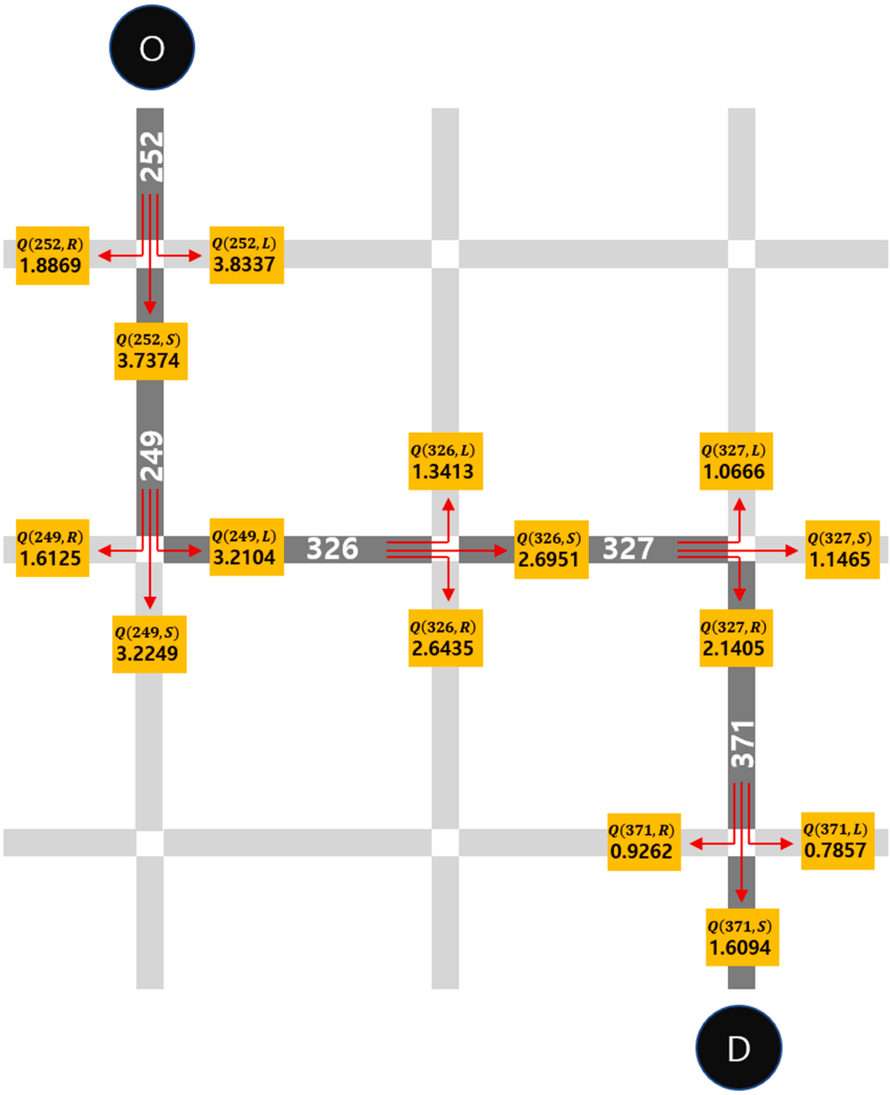


Fig. 9. Estimated values of $Q_\pi(s,a)$ along a selected route in Single-OD Binomial AIMSUN dataset.

maximum count (m_{max}) to avoid generating the same chunks to get a higher score.

$$P_n = \frac{\sum_{i \in C} \min(m_i, m_{i,max})}{w_t} \tag{16}$$

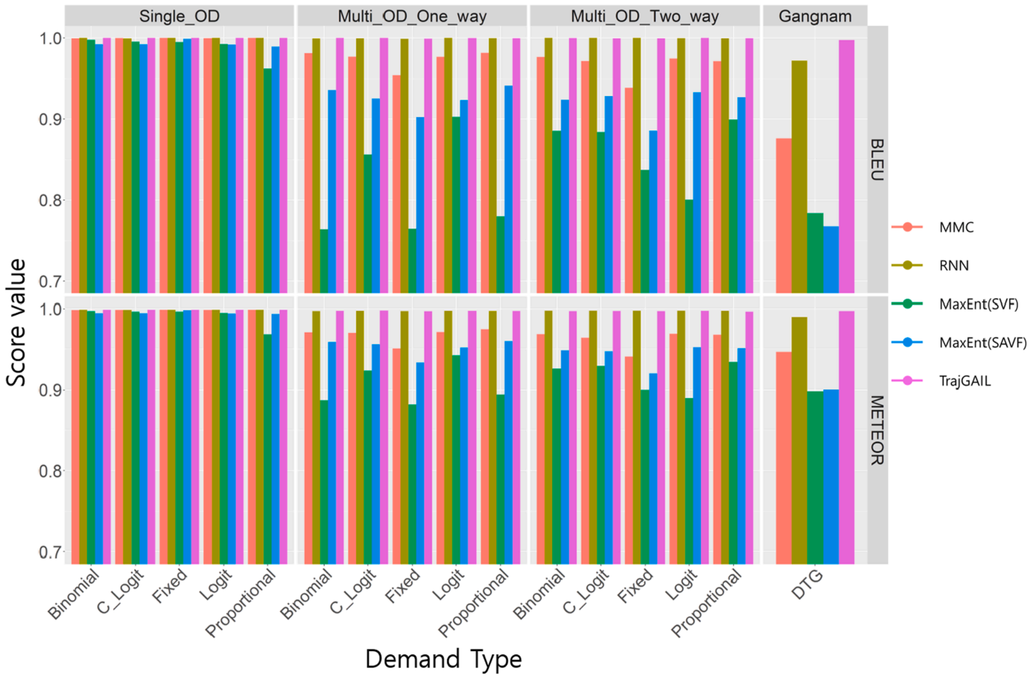
where n is the number of elements considered as a chunk; C is a set of unique chunks in the generated sequence; m_i is the number of occurrences of chunk i in the generated sequence; $m_{i,max}$ is the maximum number of occurrences of chunk i in one reference sequence; and w_t is the total number of chunks in the generated sequence.

$BLEU_n$ score is defined as a multiplication of the geometric mean of P_n and a brevity penalty. A brevity penalty is used to prevent very short candidates from receiving too high scores.

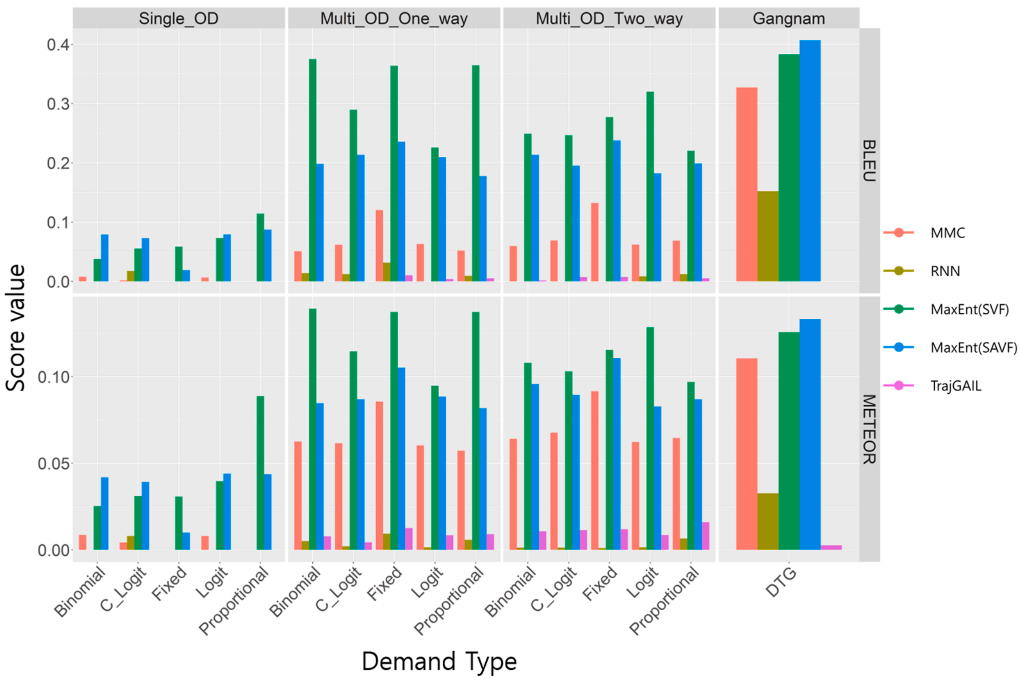
$$BLEU_n = min\left(1, \frac{L_{gen}}{L_{ref,close}}\right) \cdot \left(\prod_{i=1}^{n} P_i\right)^{\frac{1}{n}} \tag{17}$$

where L_{gen} represents the length of the generated sequence, and $L_{ref,close}$ represents the length of a reference sequence that has the closest length to the generated sequence.

METEOR (Banerjee and Lavie, 2005) is originally designed as an evaluation metric for machine translation. It can measure similarities in terms of both the occurrences of trajectory elements and the alignment of the elements in a trajectory. METEOR first creates an alignment matching between the generated sequence and the reference sequence. The alignment matching is a set of mappings between the most similar sequence elements. Since it is often used for natural language processing, the most similar sequence element refers to the exact match, synonyms, and the stems of words. In this study, it is difficult to define the "similar" observation and state, so we only use the exact match in the alignment matching. In alignment matching, every element in the candidate sequence should be



(a) Average



(b) Standard deviation

Fig. 10. Result of BLEU and METEOR score of the generated vehicle trajectories of each model.

mapped to zero or one element in the reference sequence. METEOR chooses an alignment with the most mappings and the fewest crosses (fewer intersections between mappings). Based on the chosen alignment, a penalty term is calculated as follows:

$$p = 0.5\left(\frac{c}{w_{map}}\right)^3 \tag{18}$$

where c is the number of chunks of elements with no crossings, and w_{map} is the number of elements that have been mapped. Then, we calculate the weighted harmonic mean between precision P and recall R with a ratio of the weights, 1:9.

$$F_{mean} = \frac{10}{\frac{1}{P} + \frac{9}{R}} = \frac{10PR}{R + 9P} \tag{19}$$

where $P = \frac{m}{w_{gen}}$ and $R = \frac{m}{w_{ref}}$; m is the number of elements in the generated sequence that is also found in the reference sequence; and w_{gen} and w_{ref} are the number of elements in the generated and reference sequence, respectively.

Finally, the METEOR score, M, is defined as follows:

$$M = F_{mean}(1 - p) \tag{20}$$

For both BLEU and METEOR, the higher the score, the better the model performance. For BLEU score, we use $n = 4$ for Eq. (17).

For each model, 20,000 synthetic trajectories are generated for score evaluation. Unlike a supervised learning model, a generative model as in our case does not have a single ground-truth trajectory that can be matched with each generated trajectory. As such, we consider any observed expert trajectory in the training dataset a possible reference trajectory and evaluate if the generated trajectory matches any of the available reference trajectories. More specifically, for each generated trajectory, scores are calculated against each reference trajectory in the training dataset. Then, we select the maximum value among the scores as a representative score for the given generated trajectory. As a result, a generated trajectory can be assigned a high score if it can be matched to any reference trajectory.

Fig. 10 shows the result of each model in different datasets and demand types. Fig. 10a shows the average score and Fig. 10b shows the standard deviation of the score result. When the models are tested with the Single-OD datasets, the result shows that all five models show good performance in most cases. MMC, RNN, MaxEnt(SAVF), and TrajGAIL scored more than 0.99 in both BLEU and METEOR with all five different demand types. MaxEnt(SVF) showed the lowest score (0.9627) in "Proportional" demand.

As the complexity of the dataset increases, the model's performance decreases. In Fig. 10, MMC, MaxEnt(SVF), and MaxEnt(SAVF) show decreases in both scores when tested with the One-way Multi-OD and Two-way Multi-OD datasets. However, the scores of RNN and TrajGAIL (in the second and fifth bars within each test group) only slightly decrease. In fact, both models received perfect scores (i.e., BLEU $= 1.0$, METEOR $= 1.0$) except for few cases. In addition, when the models are tested with the Gangnam dataset, the average score of RNN and TrajGAIL is 0.9726 and 0.9974 for BLEU, respectively, and 0.9899 and 0.9974 for METEOR, respectively. Overall, the standard deviations of RNN and TrajGAIL are significantly lower than that of the other models, suggesting a higher level of robustness and lower fluctuations in model performance across different trajectory cases. The main reason for the good performance of RNN and TrajGAIL is that both models are capable of capturing sequential information in trajectories considering the history of multiple previously visited locations, as opposed to determining the next locations based only on the current location as in the other three models. However, it is observed that RNN sometimes generates trajectories traversing unknown routes (link sequences not found in the given dataset) and the relatively lower scores of RNN compared to those of TrajGAIL are attributed to instances of such unrealistic trajectories.

3.3.3. Dataset-level evaluation

In the dataset-level evaluation, the statistical similarity between a generated trajectory dataset and a real trajectory dataset is assessed. There are many aspects of a dataset that can be considered for statistical similarity, such as the distributions of trajectory length, origin, destination, origin-destination pair, and route. Among these variables, route distribution is the most difficult to match since producing the similar route distribution requires matching all other variables, including the lengths, origins, and destinations of vehicle trajectories in a real dataset. As such, we use a measure of route distribution similarity to evaluate dataset-level model performance.

As with the trajectory-level evaluation, each model generates 20,000 vehicle trajectories to make a synthetic trajectory dataset. We first identified all the unique routes observed in the real dataset and counted their occurrences in the synthetic dataset. The synthetic trajectories that travel unknown routes (i.e., the routes that did not occur in the real dataset) were marked as "unknown" trajectories. The route frequencies are calculated by dividing the route counts by the total number of trajectories in a dataset. The route frequencies, or the routes' empirical probability distribution of the synthetic dataset is compared with that of the real dataset. In this study, we use Jensen-Shannon distance (d_{JS}) to measure the similarity of two route probability distributions.

Jensen-Shannon distance is a widely used distance metric for two probability distributions. Given two discrete probability distributions p and q, the Jensen-Shannon distance (d_{JS}) is defined as follows:

$$d_{JS}(p,q) = \sqrt{D_{JS}(p,q)} = \sqrt{\frac{D_{KL}\left(p \parallel \frac{p+q}{2}\right) + D_{KL}\left(q \parallel \frac{p+q}{2}\right)}{2}} \tag{21}$$

where D_{JS} is the Jensen-Shannon divergence, and D_{KL} is the Kullback-Leibler divergence. The Kullback-Leibler divergence from q to p, $D_{KL}(p \parallel q)$, is defined as:

$$D_{KL}\left(p \parallel q\right) = E[\log(p_i) - \log(q_i)] = \sum_i p_i \log \frac{p_i}{q_i} \tag{22}$$

$D_{KL}(p \parallel q)$ is also known as the relative entropy of p with respect to q. Since D_{KL} is an asymmetric similarity measure, it cannot be used as a distance metric. As a result, d_{JS}, a symmetrized version of D_{KL}, is often used to measure the distance between two probability distributions. The value of d_{JS} ranges from 0 to 1, where $d_{JS} = 0$ happens when the two probability distributions are identical and $d_{JS} = 1$ happens when the two distributions are completely different.

Table 2 shows the result of Jensen-Shannon distance (d_{JS}) of route distribution tested with different models. The lower the distance value, the better the model performance. The best model (with the lowest value) in each test case (row) is marked in bold. With the Single-OD datasets, all models except MaxEnt(SVF) show good results with d_{JS} less than 0.1. The d_{JS} of MaxEnt(SVF) is considerably larger than that of the other models. It is worth noting that MMC shows good performance, especially under the Proportional and Fixed demand patterns, which is surprising considering the simplicity of the MMC model. Although there are some differences in the distance measures, all models except MaxEnt(SVF) were able to reproduce a synthetic trajectory dataset with high degrees of statistical similarity to the real dataset, primarily because the travel patterns in the Single-OD dataset are very simple.

The Multi-OD datasets (One-way Multi-OD and Two-way Multi-OD) have more complex trajectory patterns than the Single-OD datasets. As a result, all models show an increase in d_{JS}. The increase rate is significantly large in the MaxEnt models. The main reason for this is because MaxEnt models use a simple linear function to describe reward functions and the linear reward function lacks the ability to model the complex non-linear patterns of real vehicle trajectories. Both RNN and TrajGAIL have recurrent neural networks in common to use sequential information in generating synthetic trajectory. Especially, the performance of TrajGAIL is noticeably better than the other models because it not only uses sequential embedding of visited locations but also uses the reward function from the discriminator, which offers an additional guidance for a model to produce trajectories that match the real observations.

The model performance overall decreases when tested on the Gangnam dataset, i.e., the d_{JS} values of all five models are above 0.4. TrajGAIL, however, still shows the best performance among the five models. The main reason for relatively large d_{JS} values is that the real-world trajectory patterns in the Gangnam dataset are more complex and sparse than the simulated trajectory patterns in the other datasets. For instance, there are many rare routes with counts less than 25 among 59,553 trajectories in the Gangnam dataset. Such rare routes are difficult for a model to learn due to the limited sample sizes and, consequently, the models end up generating "unknown" trajectories when attempting to reproduce these rare routes. Among 20,000 generated trajectories, the number of unknown trajectories generated by TrajGAIL is 181, while the other four models generate more than 2,000 unknown trajectories.

In Table 2, there is a tendency that d_{JS} increases as the complexity of the dataset increases. To further investigate how the dataset complexity affects a model performance, we use information entropy to quantify the complexity of a dataset. More specifically, we define the *link transition entropy* of a given vehicle trajectory dataset as follows:

$$H(D) = \frac{1}{|L|} \sum_{l_i \in L} \left(\sum_{l_j \in L} - P(l_j | l_i) \log P(l_j | l_i) \right) \tag{23}$$

where $H(D)$ is the link transition entropy of dataset D, L is a set of possible links in D, and $P(l_j|l_i)$ is an empirical transition probability that a vehicle moves from link l_i to link l_j. The intuition of this measure is to represent the dataset complexity in term of how complex or

Table 2
Jensen-Shannon distance (d_{JS}) of Route Distribution.

Dataset	Demand Type	MMC	RNN	MaxEnt(SVF)	MaxEnt(SAVF)	TrajGAIL
SingleOD	Binomial	0.0866	**0.0606**	0.0903	0.0748	0.0916
SingleOD	C-Logit	0.0381	0.0527	0.1145	0.0650	**0.0275**
SingleOD	Proportional	**0.0192**	0.0599	0.2364	0.0568	0.0274
SingleOD	Logit	0.0448	0.0526	0.1011	0.0683	**0.0284**
SingleOD	Fixed	**0.0038**	0.0153	0.0594	0.0490	0.0311
One-way MultiOD	Binomial	0.2822	0.2446	0.5234	0.3813	**0.2125**
One-way MultiOD	C-Logit	0.3032	0.2501	0.4666	0.3874	**0.1987**
One-way MultiOD	Proportional	0.2988	0.2604	0.5044	0.3825	**0.2059**
One-way MultiOD	Logit	0.3375	0.2799	0.4531	0.3893	**0.2163**
One-way MultiOD	Fixed	0.3763	**0.1747**	0.5529	0.4629	0.1791
Two-way MultiOD	Binomial	0.3018	0.3005	0.5011	0.4042	**0.2062**
Two-way MultiOD	C-Logit	0.3328	0.2587	0.4986	0.4409	**0.2072**
Two-way MultiOD	Proportional	0.3430	0.2739	0.4801	0.4388	**0.2090**
Two-way MultiOD	Logit	0.3375	0.2833	0.5815	0.4337	**0.2021**
Two-way MultiOD	Fixed	0.3763	0.1783	0.5694	0.4981	**0.1694**
Gangnam	DTG	0.4701	0.4823	0.7098	0.5558	**0.4230**

unpredictable a vehicle movement at each intersection is. For instance, a dataset would have a low $H(D)$ if all vehicles move in one direction at every intersection (e.g. all vehicles go straight at one intersection, all vehicles turn right at another intersection, and so forth), and a dataset would have a high $H(D)$ if all vehicles randomly choose the directions at all intersections.

Fig. 11 shows the relationship between the link transition entropy and the Jensen-Shannon distance of route distribution from each model. In the figure, the entropy-distance results of each model are fitted into a linear model. The slope of the linear model is defined as the *complexity sensitivity* of each model. A model with a high complexity sensitivity has a more significant drop in model performance as a dataset becomes more complex. On the other hand, a model with a low complexity sensitivity can learn the trajectory patterns regardless of the complexity of the dataset. The results show that TrajGAIL shows the lowest complexity sensitivity, suggesting its robustness and ability to scale to a larger and more complex datasets. RNN and MMC show the second and third lowest complexity sensitivity, while the two MaxEnt models have the highest complexity sensitivity.

3.3.4. Discussion

Overall, MaxEnt(SVF) showed poor performance in trajectory generation. One reason for this might be because it focuses on matching the state visitation frequency, which only cares about the element-wise similarity rather than the route-level similarity. When MaxEnt is trained to match the state-action visitation frequency, i.e., MaxEnt(SAVF), the performance improves to the level similar to the other models, which implies the importance of considering sequential information reflecting vehicles' link-to-link transition actions in generating realistic trajectories. The similar mechanism is also used in TrajGAIL, where the discriminator of TrajGAIL calculates the immediate reward based on the current state and the sampled action and this discriminator reward works in a similar way to matching the state-action visitation frequency.

From the evaluation results, TrajGAIL and RNN are found to be the most suitable models for synthetic trajectory generation. TrajGAIL, however, outperforms RNN in most of the test cases, especially in matching route distributions to the real vehicle trajectory datasets. It is, thus, worth discussing in more detail the difference between RNN and TrajGAIL and how TrajGAIL addresses the limitations of RNN. Overall, the generator of TrajGAIL is similar to the RNN model. However, TrajGAIL's generator has better performance than the RNN model. The difference comes from the modeling architecture, i.e., how the model is designed to give proper training signals to the generator. RNN is trained to minimize the cross-entropy loss between the probability of the predicted next location and the real next location as a label. This error is backpropagated through the input sequence to update model parameters to predict the next location based on the previously visited locations. While RNN can incorporate the previous locations into the next location prediction, it does not consider the rest of the trip. In contrast, TrajGAIL uses the reward function from the discriminator combined with the value estimator to consider the rest of the trip. The reward function from the discriminator captures how realistic the current state and action are. The value estimator then calculates the γ-discounted cumulative rewards, which capture how realistic the remaining states and actions will be. Using these two functions, the generator of TrajGAIL can learn a more comprehensive distribution of a given trajectory dataset and generate trajectories reflecting the sequential patterns along the whole trajectories in real data.

4. Conclusion

This study proposes TrajGAIL, a generative adversarial imitation learning framework for urban vehicle trajectory generation. In TrajGAIL, drivers' movement decisions in an urban road network are modeled as a partially observable Markov decision process

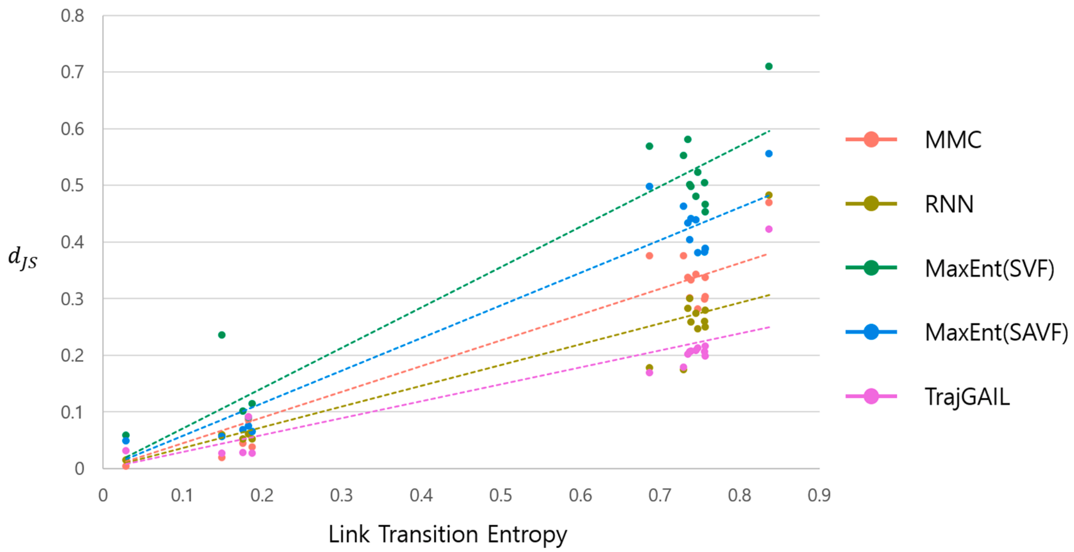


Fig. 11. Relationship between the link transition entropy and the Jensen-Shannon distance.

(POMDP). The generative adversarial imitation learning is then used to learn the underlying decision process demonstrated in the given trajectory dataset. This allows TrajGAIL to generate new synthetic trajectory data that are similar to real-world trajectory observations.

In terms of contributions, this study have made several theoretical and methodological advancements in the field of trajectory analysis and synthetic trajectory generation. TrajGAIL adopted the combination of POMDP and RNN embedding to encode the historical sequence in a current state without violating the Markov assumption required for the MDP-based imitation learning framework. This overcomes the disadvantage of standard GAIL and traditional IRL (e.g., MaxEnt) that they cannot take into account the previous locations in selecting the next location in a trajectory. TrajGAIL demonstrated the superiority of GAIL over GAN in generating location sequences from a generative adversarial framework. The imitation learning approach in GAIL allows the consideration of not only the previous locations but also future locations to make the whole trajectories more realistic, while GAN does not take into account future locations and is prone to generate unrealistic trajectories with excessive lengths and several loops. In terms of technical contributions, the use of deep learning approach (i.e., GAIL) over traditional IRL in the context of imitation learning overcomes the IRL's drawback of high computation cost when dealing with a large state-action space and, thus, offers better scalability for the applications using large-scale traffic networks.

The model's performance is evaluated on different datasets with various traffic demand patterns against three baseline models from previous studies. The evaluation is divided into two levels: trajectory-level evaluation and dataset-level evaluation. In the trajectory-level evaluation, the generated vehicle trajectories are evaluated in terms of BLEU and METEOR, the two most widely used scores in sequence modeling. In the dataset-level evaluation, the statistical similarity between the generated vehicle trajectory dataset and the real vehicle trajectory dataset is measured using the Jensen-Shannon distance of route distribution. The results show that TrajGAIL can successfully generate realistic trajectories that capture trajectory-level sequence patterns as well as match route distributions in the underlying datasets, evidenced by significantly higher performance measures of TrajGAIL compared to the other models. A model's performance sensitivity with respect to the complexity of a trajectory dataset was further investigated by measuring the link transition entropy of the dataset and analysing its relationship with model performance. The results show that TrajGAIL is least sensitive to dataset complexity among the tested models, suggesting its robustness in learning complex patterns of real vehicle movements and ability to scale to a larger and more complex dataset.

There are several directions in which the current study can be extended to further improve the trajectory generation performance. Currently, we only encode the sequence of links into the belief states in POMDP as a way to incorporate unobserved variables in modeling and predicting next locations. There are, however, other variables that can help sequence prediction in addition to the visited link sequence information. For instance, traffic conditions in a road network, vehicles' origin and destination information, and temporal information such as time-of-day and day-of-week can all provide additional information to further improve trajectory prediction and generation. While the current paper did not consider this as we focus on introducing the theoretical aspects of TrajGAIL and analyzing the effects of the structural differences between TrajGAIL and other models, we will consider incorporating other variables in our future study.

Additional information can be incorporated into TrajGAIL in various ways. Choi et al. (2019a) used an attention-based RNN model to incorporate network traffic states into the next location prediction problem. A similar attention mechanism can be employed to give the network traffic state information to the generator of TrajGAIL. As human drivers use traffic state information from navigation services to make better decision in route choices, this attention mechanism can guide the generator to pay attention to the traffic states of certain locations in the road network to select actions more accurately, in a way that human drivers do.

Another way is to use a conditional version of generative adversarial modeling framework such as conditional-GAN (cGAN) (Mirza and Osindero, 2014) and conditional-GAIL (cGAIL) (Zhang et al., 2019b), which allow models to generate synthetic data conditioned on some extra information. For example, we can feed trajectories' "origin" locations as additional input to TrajGAIL during training such that the generator and discriminator are trained to learn vehicle trajectory patterns given origin locations. This would enable the model to further capture and distinguish different route choice behaviors specific to different origin regions. Instead of feeding extra information to the model in a supervised way as in cGAN and cGAIL, it is also possible to achieve this in an unsupervised way as in infoGAN (Chen et al., 2016) and infoGAIL (Li et al., 2017). Instead of explicit condition inputs, InfoGAN and infoGAIL introduce latent variables, which are used by a model to automatically distinguish certain behaviors in data in a meaningful and interpretable way. For instance, it is possible to build a trajectory generation model that can automatically cluster and distinguish patterns in trajectory data by different latent features such as origins, destinations, and time-varying traffic demands and use this information to guide the generating process.

Lastly, it is worth mentioning the connection between our TrajGAIL model and traditional route choice models in transportation such as discrete choice model and dynamic traffic assignment (DTA) because there are similarities between them in that they both aim to model an individual driver's choice behaviors in selecting routes in traffic networks. A main difference lies in how "route" and "choice" are defined. In traditional route choice models, a route is "selected" from a pre-defined set of alternative routes for a given OD and, thus, a choice is made at the OD and route level. In our generative modeling approach, on the other hand, a route is "constructed" as a result of sequential decisions (link-to-link transitions) along the journey and, thus, a choice is made at the intersection and link level. Although a decision is made locally at each intersection, the *learned reward function* captures the network-wide route choice patterns and, thus, can give correct signals at each local decision point to enable the generated trajectories to exhibit realistic route- and network-level patterns globally. Another major difference lies in modeling approach. Traditional route choice models are "model-based" (or theory-based) in that they rely on behavioral assumptions and theories (e.g., utility maximization, user equilibrium) and aim to explain "why" a specific driver chooses a certain route. Our generative approach is "data-driven" and does not rely on any behavioral assumptions. It does not try to explain "why" but instead focuses "what" patterns exist in the actual realized data and "how" to reproduce them.

In sum, TrajGAIL offers a data-driven alternative to traditional route choice models for describing the underlying route distribution of a traffic network in synthetic data generation problems, without requiring the identification of ODs and the computationally expensive enumeration of route sets. There are limitations in TrajGAIL, however, that currently it does not take into account the effects of traffic conditions or interactions with other vehicles and, thus, is unable to serve as a route choice model for DTA or traffic simulation models. It is an important and interesting future research topic to consider how data-driven deep learning models could complement or replace existing theory-based modeling components to improve DTA and traffic simulation. The extensions to TrajGAIL discussed above (e.g., attention, cGAIL, and infoGAIL) can allow the consideration of additional information and, thus, could be potentially used in that direction.

CRediT authorship contribution statement

Seongjin Choi: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft. **Jiwon Kim:** Conceptualization, Methodology, Formal analysis, Supervision, Writing - review & editing. **Hwasoo Yeo:** Data curation, Resources, Supervision, Funding acquisition, Project administration, Writing - review & editing.

References

Abbeel, P., Ng, A.Y., 2004. Apprenticeship learning via inverse reinforcement learning, in. In: Proceedings of the twenty-first international conference on Machine learning, p. 1.

Banerjee, S., Lavie, A., 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, pp. 65–72.

Barceló, J., Casas, J., 2005. Dynamic network simulation with aimsun. In: Simulation approaches in transportation analysis. Springer, pp. 57–98.

Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C., 2005. On map-matching vehicle tracking data. In: Proceedings of the 31st international conference on Very large data bases, pp. 853–864.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P., 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Advances in neural information processing systems, pp. 2172–2180.

Chen, Z., Shen, H.T., Zhou, X., 2011. Discovering popular routes from trajectories. In: 2011 IEEE 27th International Conference on Data Engineering. IEEE, pp. 900–911.

Choi, S., Kim, J., Yeo, H., 2019a. Attention-based recurrent neural network for urban vehicle trajectory prediction. Procedia Comput. Sci. 151, 327–334.

Choi, S., Kim, J., Yu, H., Yeo, H., 2019b. Real-time prediction of arterial vehicle trajectories: An application to predictive route guidance for an emergency vehicle. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, pp. 4030–4036.

Choi, S., Yeo, H., Kim, J., 2018. Network-wide vehicle trajectory prediction in urban traffic networks using deep learning. Transp. Res. Rec. 2672, 173–184.

Chow, C.Y., Mokbel, M.F., 2011. Privacy of spatial trajectories. In: Computing with spatial trajectories. Springer, pp. 109–141.

De Cáceres, M., Coll, L., Legendre, P., Allen, R.B., Wiser, S.K., Fortin, M.J., Condit, R., Hubbell, S., 2019. Trajectory analysis in community ecology. Ecol. Monogr. 89, e01350.

De Cao, N., Kipf, T., 2018. Molgan: An implicit generative model for small molecular graphs. arXiv preprint arXiv:1805.11973.

Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., Courville, A., 2016. Adversarially learned inference. arXiv preprint arXiv:1606.00704.

Feng, J., Li, Y., Zhang, C., Sun, F., Meng, F., Guo, A., Jin, D., 2018. Deepmove: Predicting human mobility with attentional recurrent networks. In: Proceedings of the 2018 world wide web conference, pp. 1459–1468.

Feng, Y., Sun, J., Chen, P., 2015. Vehicle trajectory reconstruction using automatic vehicle identification and traffic count data. J. Adv. Transp. 49, 174–194.

Gambs, S., Killijian, M.O., del Prado Cortez, M.N., 2010. Show me how you move and i will tell you who you are, in. In: Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS, pp. 34–41.

Gambs, S., Killijian, M.O., del Prado Cortez, M.N., 2012. Next place prediction using mobility markov chains. In: Proceedings of the First Workshop on Measurement, Privacy, and Mobility. ACM, p. 3.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Advances in neural information processing systems, pp. 2672–2680.

Ho, J., Ermon, S., 2016. Generative adversarial imitation learning. In: Advances in neural information processing systems, pp. 4565–4573.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9, 1735–1780.

Howard, R.A., 1960. Dynamic programming and markov processes. John Wiley.

Hu, G., Shao, J., Ni, Z., Zhang, D., 2018. A graph based method for constructing popular routes with check-ins. World Wide Web 21, 1689–1703.

Jin, C., Lin, Z., Wu, M., 2019. Augmented intention model for next-location prediction from graphical trajectory context. Wireless Commun. Mob. Comput.

Kaelbling, L.P., Littman, M.L., Cassandra, A.R., 1998. Planning and acting in partially observable stochastic domains. Artif. Intell. 101, 99–134.

Kim, J., Mahmassani, H.S., 2015. Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories. Transp. Res. Part C: Emerg. Technol. 59, 375–390.

Laube, P., 2014. Computational movement analysis. Springer.

Lee, W.C., Krumm, J., 2011. Trajectory preprocessing. In: Computing with spatial trajectories. Springer, pp. 3–33.

Li, L., Zhu, J., Zhang, H., Tan, H., Du, B., Ran, B., 2020. Coupled application of generative adversarial networks and conventional neural networks for travel mode detection using gps data. Transp. Res. Part A: Policy Pract. 136, 282–292.

Li, Y., Song, J., Ermon, S., 2017. Infogail: Interpretable imitation learning from visual demonstrations. In: Advances in Neural Information Processing Systems, pp. 3812–3822.

Lin, Z., Khetan, A., Fanti, G., Oh, S., 2018. Pacgan: The power of two samples in generative adversarial networks. In: Advances in neural information processing systems, pp. 1498–1507.

Liu, Q., Wu, S., Wang, L., Tan, T., 2016. Predicting the next location: A recurrent model with spatial and temporal contexts. In: Thirtieth AAAI conference on artificial intelligence, pp. 194–200.

Liu, X., Chen, H., Andris, C., 2018. trajgans: Using generative adversarial networks for geo-privacy protection of trajectory data (vision paper). In: Location Privacy and Security Workshop, pp. 1–7.

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.

Monreale, A., Pinelli, F., Trasarti, R., Giannotti, F., 2009. Wherenext: a location predictor on trajectory pattern mining, in. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 637–646.

Ng, A.Y., Russell, S.J., et al., 2000. Algorithms for inverse reinforcement learning.. In: Icml, p. 2.

Oord, A.V.D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Ouyang, K., Shokri, R., Rosenblum, D.S., Yang, W., 2018. A non-parametric generative model for human trajectories. In: IJCAI, pp. 3812–3817.

Papineni, K., Roukos, S., Ward, T., Zhu, W.J., 2002. BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, pp. 311–318.
Popić, S., Pavković, B., Velikić, I., Teslić, N., 2019. Data generators: a short survey of techniques and use cases with focus on testing. In: 2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin). IEEE, pp. 189–194.
Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv: 1511.06434.
Rao, J., Gao, S., Kang, Y., Huang, Q., 2020. Lstm-trajgan: A deep learning approach to trajectory privacy protection. arXiv preprint arXiv:2006.10521.
Rao, R.P., 2010. Decision making under uncertainty: a neural model based on partially observable markov decision processes. Front. Comput. Neurosci. 4, 146.
Rao, W., Wu, Y.J., Xia, J., Ou, J., Kluger, R., 2018. Origin-destination pattern estimation based on trajectory reconstruction using automatic license plate recognition data. Transp. Res. Part C: Emerg. Technol. 95, 29–46.
Russell, S., 1998. Learning agents for uncertain environments. In: Proceedings of the eleventh annual conference on Computational learning theory, pp. 101–103.
Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y., 2000. Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, pp. 1057–1063.
Wu, R., Luo, G., Shao, J., Tian, L., Peng, C., 2018. Location prediction on trajectory data: A review. Big Data Min. Analyt. 1, 108–127.
Wulfmeier, M., Ondruska, P., Posner, I., 2015. Deep inverse reinforcement learning. CoRR, abs/1507.04888.
Xu, D., Wei, C., Peng, P., Xuan, Q., Guo, H., 2020. Ge-gan: A novel deep learning framework for road traffic state estimation. Transp. Res. Part C: Emerg. Technol. 117, 102635.
Zhang, K., Jia, N., Zheng, L., Liu, Z., 2019a. A novel generative adversarial network for estimation of trip travel time distribution with trajectory data. Transp. Res. Part C: Emerg. Technol. 108, 223–244.
Zhang, X., Li, Y., Zhou, X., Luo, J., 2019b. Unveiling taxi drivers' strategies via cgail: Conditional generative adversarial imitation learning. In: 2019 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 1480–1485.
Ziebart, B.D., Maas, A.L., Bagnell, J.A., Dey, A.K., 2008a. Maximum entropy inverse reinforcement learning. In: Aaai, Chicago, IL, USA. pp. 1433–1438.
Ziebart, B.D., Maas, A.L., Dey, A.K., Bagnell, J.A., 2008b. Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior, in. In: Proceedings of the 10th international conference on Ubiquitous computing, pp. 322–331.