Shortening passengers' travel time: A dynamic metro train scheduling approach using deep reinforcement learning

Zhaoyuan Wang, Zheyi Pan, Shun Chen, Shenggong Ji, Xiuwen Yi, Junbo Zhang, *Member, IEEE,* Jingyuan Wang, Zhiguo Gong, *Senior Member, IEEE,* Tianrui Li, *Senior Member, IEEE,* and Yu Zheng, *Fellow, IEEE*

Abstract—Urban metros have become the foremost public transit to modern cities, carrying millions of daily rides. As travel efficiency matters to the work productivity of the city, shortening passengers' travel time for metros is therefore a pressing need, which can bring substantial economic benefits. In this paper, we study a fine-grained, safe, and energy-efficient strategy to improve the efficiency of metro systems by dynamically scheduling dwell time for trains. However, developing such a strategy is very challenging because of three aspects: 1) The objective of optimizing the average travel time of passengers is complex, as it needs to properly balance passengers' waiting time at platforms and journey time on trains, as well as considering long-term impacts on the whole metro system; 2) Capturing dynamic spatio-temporal (ST) correlations of incoming passengers for metro stations is difficult; and 3) For each train, the dwell time scheduling is affected by other trains on the same metro line, which is not easy to measure. To tackle these challenges, we propose a novel deep neural network, entitled AutoDwell. Specifically, AutoDwell optimizes the long-term rewards of dwell time settings in terms of passengers' waiting time at platforms and journey time on trains by a reinforcement learning framework. Next, AutoDwell employs gated recurrent units and graph attention networks to extract the ST correlations of the passenger flows among metro stations. In addition, attention mechanisms are leveraged in AutoDwell for capturing the interactions between the trains on the same metro line. Extensive experiments on two real-world datasets collected from Beijing and Hangzhou, China, demonstrate the superior performance of AutoDwell over several baselines, capable of saving passengers' overall travel time. In particular, the model can shorten the waiting time by at least 9%, which can boost passengers' experience significantly.

Index Terms—Metro Systems, Spatio-temporal Data, Neural Network, Deep Reinforcement Learning, Urban Computing.

1 Introduction

URBAN metros have become the most important public transit because of the convenience, safety, punctuality, and high efficiency. For example in Beijing, a city with about 30 million people, the metro delivers an average of 10.5 million trips per day, as exhibited in Figure 1(a) [1]. According to [1], [2], travel time significantly matters people's work productivity [2]. That is, shortening metro passengers' travel time can improve the work efficiency of the city, bringing substantial economic benefits. To this end, many approaches have been taken by metro operators, such as increasing the frequency of trains and accelerating their speed. However, these ways introduce large energy consumption with huge costs and, more importantly, carry risks [3]. Thus, a fine-grained, safe, and energy-efficient operation is urgently needed for metro systems.

In the real world, the dwell time of trains for each station is usually fixed and decided by expert experience. For example, the dwell time of Line 14, Beijing is 62s and 74s during peak hours and off-peak hours, respectively. However, such a fixed schedule takes very limited information about the dynamic and diverse distributions of incoming passengers into consideration, and consequently there is still a large space to improve the system efficiency. As the two cases presented in Figure 1(b), by knowing how many people will come, we can extend the dwell time such that upcoming passengers do not miss the train (*i.e.,* Case 1), or reduce the dwell time to make the train do not cost unnecessary waiting time at stations (*i.e.,* Case 2). Recently, large amounts of passengers' check-in and -out records of metro stations are collected, inspiring us to study the dynamic strategies that can assign the dwell time of trains based on passengers' demands. However, it meets several challenges.

1) The objective of optimizing the average travel time of passengers is extremely complex. First of all, changing the dwell time has interacted impacts on the overall travel time from two aspects: I) the waiting time for passengers at platforms, and II) the journey time of those people who are already on trains. For instance, a long dwell time can potentially reduce people's waiting time at the platform, whereas it could increase the journey time of those people

- *Zhaoyuan Wang, Shun Chen, Shenggong Ji, and Tianrui Li are with School of Information Science and Technology, Southwest Jiaotong University, China.*
- *Zheyi Pan is with Department of Computer Science and Engineering, Shanghai Jiaotong University, China.*
- *Xiuwen Yi, Junbo Zhang, and Yu Zheng are with JD Intelligent Cities Research and JD Intelligent Cities Business Unit, China.*
- *Jingyuan Wang is with School of Computer Science and Engineering, Beihang Unversity, China.*
- *Zhiguo Gong is with Faculty of Science and Technology, University of Macau, China.*

E-mail: wang_zhaoyuan@foxmail.com
Manuscript received November 1, 2020.

1. http://www.beijing.gov.cn/gongkai/shuju/

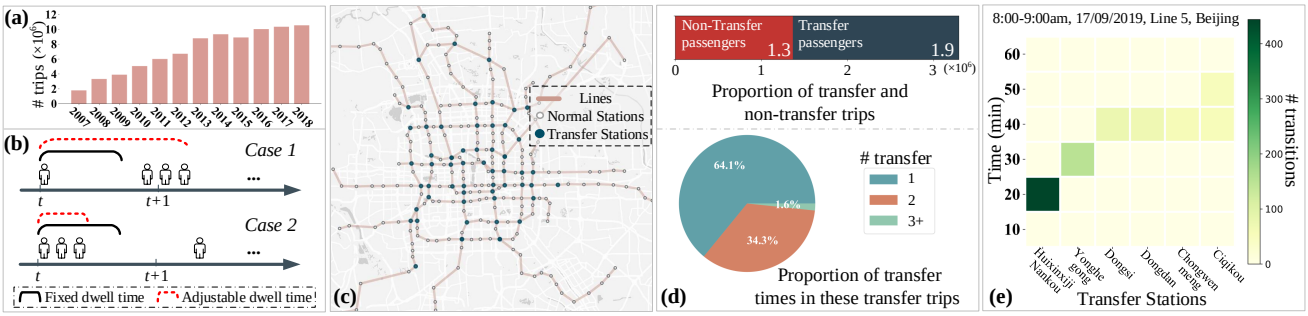


Fig. 1: (a) Average # of daily trips in Beijing metro over years; (b) Comparison between two types of dwell process; (c) Multi-line metro system of Beijing; (d) Passenger distributions of a single day's trips in terms of the transfer behavior (17/09/2019 in downtown of Beijing); (e) Spatial-temporal correlations of passenger flows between different stations.

who are already on the train. Therefore, the optimization process needs to consider both factors simultaneously. Second, changing a train's dwell time at a station will entirely modify the schedule of arrival time at the subsequent stations, introducing very long-term impacts on the overall travel time. Therefore, it is non-trivial to schedule the right dwell timing by considering the long-term impacts.

2) To accurately capture passenger states of metro stations is difficult, owing to complicated and intertwined spatio-temporal (ST) correlations. As demonstrated in Figure 1(c) and (d), in current metro systems, there are many metro lines intersected and many passengers have to change many metro lines to reach their destinations. That is, the passenger flow of a metro station is affected by other stations that are geographically related to the station. In addition, the number of incoming passengers is highly dynamic in temporal domains. As the example shown in Figure 1(e), large amounts of passengers got on trains at Tiantongyuan station and will appear on subsequent transfer stations after different periods, indicating the short-term temporal impacts. There are also long-term impacts, *e.g.*, the passenger flow pattern during morning rush hours is similar on consecutive workdays, repeating every 24 hours.

3) The interactions between trains on the same metro line are dynamic. When a train arrives at a station, it would pick up waiting passengers on the platform, changing the state of the passenger distribution in this station and impacting the upcoming trains' decisions for dwell time. For instance, if we know there is not enough space for the preceding train of the current train, we should consider reducing the dwell time of the train such that passengers who cannot get on the preceding train can save their waiting time.

Recently, many works have been studied to determine the dwell time for trains based on passengers' demand [4]–[7]. Given the exact number of waiting passengers at a station, these existing studies learn a function that estimates the dwell time required for these passengers. However, their performances are limited due to two reasons. First, they only consider to satisfy the demand of the current station, and consequently can not capture these long-term impacts on the overall travel time. Second, these prior works overlook the interactions between trains, leading to the improvement of the overall metro system is limited.

To tackle all aforementioned challenges, we propose a novel deep neural network, entitled AutoDwell, which can reduce the average passengers' travel time by properly scheduling the dwell time to each train for its next station. In summary, the contributions of this paper are four-fold:

- We propose using reinforcement learning framework to learn AutoDwell. Specifically, we design the reward of dwell time by weighing the passengers' waiting time at platforms and the journey time on trains, and take advantage of the reinforcement learning framework to optimize the long-term impacts of the reward.
- For each train, AutoDwell makes decisions by taking into consideration the dynamic passenger states. Particularly, a network component, named passenger feature extractor, consisting of gated recurrent units and graph attention networks, is adopted to embed the passengers' information by capturing the complex ST correlations among stations.
- For each train, we propose to leverage the states of its both front and rear trains on its metro line as the additional input of AutoDwell, and use attention mechanisms as the train feature extractor of AutoDwell to capture the interactions between them.
- We evaluate our model using real-world datasets collected from two big cities in China, *i.e.*, Beijing and Hangzhou. Experimental results demonstrate the advantages of our model over several baselines, capable of saving passengers' overall travel time. In particular, the model can shorten the waiting time by at least 9%, which can boost passengers' experience significantly.

2 Overview

In this section, we first formulate the dynamic dwell time scheduling problem. Then, the framework of our solution for addressing the problem is introduced.

2.1 Problem Formulation

2.1.1 Preliminary

We introduce two important concepts of this paper as follows.

1) **Multi-line metro system**. Namely, the formulation of a real world metro system.
   1) *Line*. As we known, a metro line always has two directions in the real world. Since very few passengers interchange between two directions of the same line, we regard the two different directions as different lines

in the formulated system That is, there are n^{line} lines in the formulated metro system, which can correspond to $\frac{n^{\text{line}}}{2}$ lines in the real world).

2) *Station*. We represent a metro station by a 2-tuple, *e.g.,* $u_i = (\#line, \#realstation)$ where $\#line$ and $\#realstation$ denotes the line ID and the station ID, respectively. Through the 2-tuple, we can uniformly model transfer stations and transfer stations. That is, unlike normal stations that only belong to one line, in transfer stations, passengers can pass multiple lines. Specifically, we decompose a transfer station as multiple stations where each station belongs to only one line and these decomposed stations share the same station ID. For example, given u_i and u_j ($i \neq i$), $u_i.\#line = u_j.\#line$ means the two stations belong to the same line while $u_i.\#realstation = u_j.\#realstation$ represents they are two decomposed stations from the same real-world transfer station. In sum, there are n^{stat} stations in the formulated system.

3) *Train*. In the system, we assume all trains' type are the same. Concretely, the passenger capacity and the speed upper bound (determines the energy consumption of trains [3]) of all trains are the same. The capacity is expressed as n^{capa} and the train speed are set according to the real-world (see Section 4.1.2 for details). In addition, in the formulated system, the number of trains are fixed per unit time as the real-world system.

4) *Dwell process*. Accodring to the real-world setting, the dwell time in the formulated system is within a reasonable range $[\delta^{\min}, \delta^{\max}]$, during which passengers can get off and get on the train. We use δ_{m,u_j} to denote the dwelling time of train m at station u_j where $\delta^{\min} \leq \delta_{m,u_j} \leq \delta^{\max}$. Specifically, $\delta_{m,u_j} = \delta^{\text{fixed}} + \delta^{\text{alig}}_{m,u_j} + \delta^{\text{boar}}_{m,u_j}$, where δ^{fixed} is a constant for all trains and stations, including the time for a train to enter the platform, the time to open and close the door, and the time for the train to leave the platform. The rest of two items, *i.e.,* $\delta^{\text{alig}}_{m,u_j}$ and $\delta^{\text{boar}}_{m,u_j}$, are door utilization time for alighting and boarding passengers. In this paper, linear models are used to simulate the alighting and boarding procedures [8]. Concretely, we use $\rho^{\text{alig}}_{m,u_j}$ represents the number of alighting passengers of train m in station u_j. Namely, given a fixed alighting velocity for trains v^{alig}, $\delta^{\text{alig}}_{m,u_j}$ is calculated by $\delta^{\text{alig}}_{m,u_j} = \rho^{\text{alig}}_{m,u_j}/v^{\text{alig}}$. If $\delta^{\text{alig}}_{m,u_j} > \delta_{m,u_j} - \delta^{\text{fixed}} - \delta^{\text{mixed}}$, we set $\delta^{\text{alig}}_{m,u_j} = \delta_{m,u_j} - \delta^{\text{fixed}} - \delta^{\text{mixed}}$ where δ^{mixed} is a very small value, simulating the real-world situation that all alighting passengers will get off and remains a very short time. After the end of the alighting process, the boarding process starts. The boarding time $\delta^{\text{boar}}_{m,u_j} = \delta_{m,u_j} - \delta^{\text{fixed}} - \delta^{\text{alig}}_{m,u_j}$. The ideal number of boarding passengers is $\bar{\rho}^{\text{boar}}_{m,u_j} = v^{\text{boar}} \times \delta^{\text{boar}}_{m,u_j}$, where v^{boar} is a fixed boarding velocity for trains. However, the actual number of passengers on board is subjected to the number of passengers on the train and waiting at the platform. Assuming the total number of passengers already on the train m when it leaves from the station u_i (the last station of the station u_j) is $\rho^{\text{total}}_{m,u_i}$, the remaining carrying capacity of the train $\rho^{\text{remain}}_{m,u_j} = n^{\text{capa}} - \rho^{\text{total}}_{m,u_i} + \rho^{\text{alig}}_{m,u_j}$. In addition, the number of waiting passengers at the platform of the station u_j can be represented as $\rho^{\text{plat}}_{m,u_j}$. As a result, the number of boarding passengers is given:

$$\rho^{\text{boar}}_{m,u_j} = \begin{cases} \bar{\rho}^{\text{boar}}_{m,u_j}, & \text{if } \bar{\rho}^{\text{boar}}_{m,u_j} \leq \rho^{\text{plat}}_{m,u_j} \text{ and } \bar{\rho}^{\text{boar}}_{m,u_j} \leq \rho^{\text{remain}}_{m,u_j}; \\ \rho^{\text{remain}}_{m,u_j}, & \text{if } \rho^{\text{remain}}_{m,u_j} \leq \rho^{\text{plat}}_{m,u_j} \text{ and } \rho^{\text{remain}}_{m,u_j} \leq \bar{\rho}^{\text{boar}}_{m,u_j}; \\ \rho^{\text{plat}}_{m,u_j}, & \text{if } \rho^{\text{plat}}_{m,u_j} \leq \rho^{\text{remain}}_{m,u_j} \text{ and } \rho^{\text{plat}}_{m,u_j} \leq \bar{\rho}^{\text{boar}}_{m,u_j}. \end{cases}$$

2) **Historical passenger check-in readings**. The historical passenger check-in readings is denoted as a tensor $\mathbf{X}_{\text{data}} = [X_1, ..., X_{n^{\text{hist}}}] \in \mathbb{R}^{n^{\text{hist}} \times n^{\text{stat}}}$, where n^{hist} is the number of timestamps and n^{stat} is the number of regions. Given an index (i, j), where $1 \leq i \leq n^{\text{hist}}$ and $1 \leq j \leq n^{\text{stat}}$, the corresponding value of tensor $\mathbf{X}_{\text{data}}$ at this index denotes the check-in value of the station j at timestamp i.

2.1.2 Problem statement

Given the above multi-line metro system and historical passenger check-in readings, we try to learn an effective policy π, which determines trains' dwell time by capturing the states of dynamic passengers and their context trains, aiming to shorten the overall passengers' average travel time of the system for a long time. Specifically, since the metro system is a dynamic system, one dwell decision may affect the whole system. That is, we can not optimize each station dependently, *e.g.,* to wait for passengers at the current station, it may cause over waiting for passengers at the next stations, resulting in an increase of the overall travel time. Hence, we formulate the long-term optimization problem by Q-learning-based reinforcement learning, which can be characterized with an agent and five major components $\{\mathcal{A}, \mathcal{S}, r, \pi, Q\}$:

1) *Agent* & *Action set* $\mathcal{A}$. In a multi-line system, there will be many trains running simultaneously. Thus, optimizing all trains together leads to a considerable solution space that is difficult to be optimized. Hence, we propose to let the agent be a single train and all trains share the same policy. When an arbitrary train is going to leave from its current station, our policy selects the action, *i.e.,* the dwell time for the train. It can therefore largely reduce the action space from $(n^{\text{action}})^{n^{\text{train}}}$ to n^{action} where n^{action} and n^{train} is the number of actions and running trains, respectively. Note that, even if the agent is a single train, by using the policy, trains also can cooperate to improve the system efficiency. This happens because, for each train, the policy can consider statuses of its related trains when performing the dwell action (see Section 3.1).

$\mathcal{A}$ consists of all the dwell time selections for trains. As the dwell time ranges from a continuous range $[\delta^{\min}, \delta^{\max}]$, to further reduce the action space, we uniformly discretize the range to n^{action} values as our actions.

2) *State set* $\mathcal{S}$. For a train that is going to depart, it observes two kinds of information impacting the dwell time action, including the passenger flow states of its following stations and the states of other trains on the same metro line. Notably, the passenger flow states are extracted from the historical passenger check-in readings. Mathematically, we denote the passenger state and the train state as $\mathcal{X}$ and $\mathcal{C}$, respectively, which will be detailed in the next section.

3) *Immediate reward* r. After taking an action under a state and transiting to the next one, the agent receives an immediate reward. In this work, to optimize the average travel time and involve the long-term impacts by the reinforcement learning framework, we devise the immediate reward by considering the waiting time of passengers at platforms and the journey time of passengers on trains.

Concretely, when train m has dwelled for a given action δ_{m,u_j} in station u_j and is going to depart, we can calculate the immediate reward for the action by Eq. 1.

$$r_{m,u_j} = \omega \times \psi_{m,u_j} - (1-\omega) \times \epsilon_{m,u_j} \quad (1)$$

where ψ_{m,u_j} and ϵ_{m,u_j} are two metrics obtained based on the waiting time and journey time respectively, and ω is a parameter to weight the importance of the two parts. Suppose the number of passengers boarded, alighted, and when leaving the station are $\rho^{\text{boar}}_{m,u_j}$, $\rho^{\text{alig}}_{m,u_j}$, and $\rho^{\text{total}}_{m,u_j}$ respectively, the two metrics are defined as below.

Definition 1. Waiting time cost ψ. For passenger p_k who just get on the train, the waiting time, *i.e.,* the time from her arrival at platform to the train starts to run, is denoted as $\delta^{\text{waiting}}_{p_k}$. Then, we define the metric as $\psi_{m,u_j} = \sum_{k}^{\rho^{\text{boar}}_{m,u_j}} \delta^{\text{waiting}}_{p_k}$. The larger the ψ_{m,u_j}, the more number of or the more over-waited passengers the train serves.

Definition 2. Journey time cost ϵ. The metric can be represented as $\epsilon_{m,u_j} = \kappa \times \delta_{m,u_j}$ where $\kappa = \rho^{\text{total}}_{m,u_j} - \rho^{\text{boar}}_{m,u_j}$ denotes the number of passengers passing this station. The smaller the ϵ_{m,u_j}, the shorter time spent on the journey for passing passengers.

4) *Policy* π & *long-term value function* Q. An agent interacts with its environment in discrete time steps. At each time step t, an agent under state s_t takes an action a_t according to a policy π (*i.e.,* a mapping function: $\mathcal{S} \times \mathcal{A} \rightarrow \pi$). Then, it transits to the next state s_{t+1}, receiving an immediate reward r_t. In an episode, *i.e.,* a specific time interval, the cumulated reward R_t of the action from time step t can be calculated as Eq. 2.

$$R_t = r_t + \gamma \times r_{t+1} + \gamma^2 \times r_{t+2} + ... + \gamma^{T-t} \times r_T \quad (2)$$

where γ is a discount parameter and T is the last period in the episode. Then, the optimal value function Q returns the maximum expected long-term reward of each action a_t as shown in Eq. 3.

$$Q(s_t, a_t) = \max_\pi \mathbb{E}[R_t | s_t, a_t; \pi] \quad (3)$$

After obtaining this function, the corresponding optimal policy of the reinforcement learning can be easily inferred. Namely, always taking the action with the maximum optimal Q value under the current state: $a^*_t = \text{argmax}_a Q(s_t, a_t)$

Bellman equation shown in Eq. 4 is usually adopted to estimate the optimal long-term value function via an iterative approach.

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \times \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) | s_t, a_t] \quad (4)$$

In this paper, due to the difficulty of specifying the sophisticated long-term value function Q, we design a novel deep network, *i.e.,* AutoDwell, to store our policy π. That is, we try to learn the policy network to optimize the passenger travel efficiency for a long time.

2.2 System Framework

To solve the formuated problem, we propose a deep neural network, *i.e.,* AutoDwell, as the scheduling policy. Figure 2 overviews the system that trains and deploys the AutoDwell, consisting of two phases: an offline learning procedure and an online deployment procedure.

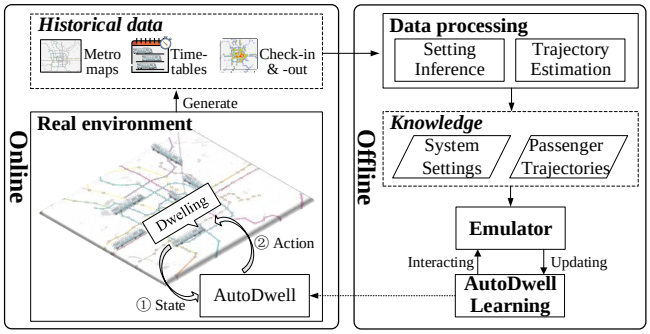


Fig. 2: System framework

2.2.1 Offline learning

In this procedure, we train the policy model AutoDwell, containing three major components: a data processing module, an emulator module, and an AutoDwell learning module.

Data processing. We extract two kinds of knowledge in this module, which will be used subsequently. First, we extract settings of the real-world metro system, such as metro structures, departure timetables, and speed upper bounds. Second, we estimate passengers' trajectories based on check-in and -out records (a kind of origin-destination (OD) data with timestamp). Specifically, a trip of passengers from check-in to check-out can be divided into the following parts, including entry time (walking time from station entry to the train platform), waiting time (waiting time from arriving at the platform to the running of the train), journey time (the time on the train), and exit time (walking time from the platform to the exit). In particular, line transfer time is considered when a trip's origin and destination belong to different lines. As the trajectory estimation task has been investigated [9]–[11], we adopt the authoritative method of Beijing Metro Network Control Center that has been reported in [9].

Emulator. Because emulation is an effective way and has been widely used in a large number of previous works on training deep reinforcement learning models [12]–[16], in this work, we construct an emulator to provide the formulated metro system above. Despite using emulator, we can still train robust models because we build the emulator with real data (*i.e.,* adopting the extracted real-world settings and using the real passenger trajectories as the environment). Specifically, in the emulator, we model the metro system based on the real metro structures. Given the real departure timetables, trains are sent from the initial stations of the

lines accordingly. Moreover, between every two successive stations, we use the real maximum speed to control trains. That is, trains will run at a speed less than or equal to the specified maximum speed between the current two stations, on the premise of ensuring safety. In particular, passengers enter the emulated metro system according to their real check-in time, and their original station, destination station, and trajectories are all consistent with their real records. Details of the settings of the emulator can be found in Section 4.1.2.

AutoDwell Learning. In this component, we follow the reinforcement learning paradigm to learn the deep policy network by interacting with an emulated environment. The deep network will be described in Section 3.

2.2.2 Online deployment

After the deep network is converged, we can deploy our model in the real world. That is, given the observed input state, we directly use the learned network, *i.e.,* AutoDwell, to guide the dwell time scheduling according to the maximum Q-value. In practice, the model can be embedded into the metro control system to direct the dwell time, or can be independent with the control system and provides dwell time suggestions for metro operators.

3 AUTODWELL

Figure 3(a) shows the architecture of the proposed deep network, *i.e.,* AutoDwell, which unlocks long-term rewards of actions according to the observed state by the guidance of the immediate reward. Concretely, it consists of three components: 1) a train feature extractor to capture interactions between the current train and other trains on the same line based on the train state $\mathcal{C}$; 2) a passenger feature extractor for embedding the upcoming passenger' information among the passenger state $\mathcal{X}$ by considering and weighing the ST correlations among all these subsequent stations of the train, and 3) a fusion network to fuse the two parts of knowledge and accordingly provide Q-values for actions.

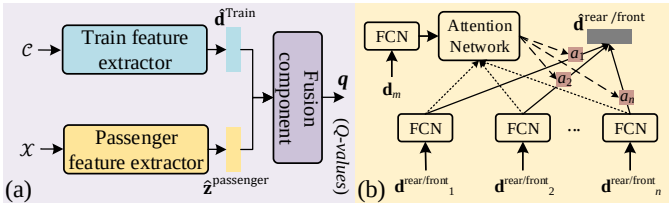


Fig. 3: (a) Overview of AutoDwell; (b) Structure of the train feature extractor.

3.1 Train Feature Extractor

We use the train state $\mathcal{C}$ to record statuses of the current train and its context trains such that we can model their interactions. Specifically, for the current train m that is heading to station u_i, the train state can be represented as $\mathcal{C}_{m,u_i} = \{\mathbf{d}_m, \mathbf{D}_m^{\text{front}}, \mathbf{D}_m^{\text{rear}}\}$. Vector $\mathbf{d}_m \in \mathbb{R}^{n^{\text{fea}}}$ denotes features of the current train, including its running status, location, passenger quantity, etc. In addition, for the train, as two types of context trains (trains running on the same line can impact the current train) exist, *i.e.,* front-sequence trains and rear-sequence trains, we use matrices $\mathbf{D}_m^{\text{front}} \in \mathbb{R}^{n_m^{\text{front}} \times n^{\text{fea}}}$ and $\mathbf{D}_m^{\text{rear}} \in \mathbb{R}^{n_m^{\text{rear}} \times n^{\text{fea}}}$ to represent features of the two types of context trains respectively, in which n_m^{front} and n_m^{rear} are the number of front trains and rear trains, respectively.

Thereafter, as depicted in Figure 3(b), we propose to use attention mechanisms [17] to capture the interactions based on the state for two reasons. First, the attention mechanisms are studied to infer the importance of different parts of the training data and let the learning algorithms focus on the most informative parts. That is we can focus on these important trains that have the greatest impact on the current train. Second, with the running of the current train, the number of these context trains is changing. Namely, later context trains will start from the initial station to join the system, and earlier context trains will arrive at the terminal station and exit the system. The attention mechanisms can handle such a situation with the length-variable data.

We respectively capture the impacts of the two types of context trains by using the same attention network component. In the following, we take the rear trains as a concrete example to describe this component. Formally, the inputs of the component are $\mathbf{d}_m$ and $\mathbf{D}_m^{\text{rear}} = [\mathbf{d}_1^{\text{rear}}, ..., \mathbf{d}_{n_m^{\text{rear}}}^{\text{rear}}]$. First, we use a shared linear transformation, parametrized by weight matrix $\mathbf{W}^{\text{Tr}}$, to apply to these trains' feature vector for obtaining sufficient expressive power. Then, we attend the interaction between m and each rear train and get a normalized importance weight by a softmax function. For instance, the importance weight of the i-th rear train, denoted as a_i, can be obtained as follows:

$$e_i = \tanh\left(\mathbf{v}^T\left(\mathbf{W}^{\text{Tr}}\mathbf{d}_m, \mathbf{W}^{\text{Tr}}\mathbf{d}_i^{\text{rear}}\right)\right);\ a_i = \frac{\exp(e_i)}{\sum_{z \in n_m^{\text{rear}}} \exp\left(e_z\right)} \tag{5}$$

where $\mathbf{v}$ is a single-layer feed-forward neural network. Next, the output hidden representation vector, capturing the interactions of all rear trains, is derived as:

$$\hat{\mathbf{d}}_{m,u_i}^{\text{rear}} = \sum_{z \in n_m^{\text{rear}}} a_z \mathbf{W}^{\text{Tr}} \mathbf{d}_z^{\text{rear}} \tag{6}$$

Similarly, we can obtain $\hat{\mathbf{d}}_{m,u_i}^{\text{front}}$ by feeding the $\mathbf{d}_m$ and $\mathbf{D}_m^{\text{front}}$ into this component. To sum up, the final output of this component, capturing both the current train's status and impacts from the two types of context trains, is: $\hat{\mathbf{d}}_{m,u_i}^{\text{train}} = \hat{\mathbf{d}}_{m,u_i}^{\text{rear}} || \hat{\mathbf{d}}_{m,u_i}^{\text{front}} || (\mathbf{W}^{\text{Tr}}\mathbf{d}_m)$, where $||$ is the concatenation operator. Notably, by considering other trains' state when a train performs actions, AutoDwell follows a multi-agent learning paradigm that can make trains cooperate mutually [13].

3.2 Passenger Feature Extractor

When a train is performing an action, we aim to perform long-term benefited actions by referring to passenger states of all its subsequent stations. To this end, we store statuses of the current train's subsequent stations in the passenger state. Moreover, to reveal the short- and long- term ST correlations of passengers, for each station, its passenger state is with two kinds of information, including recent passenger flows and external features such as hours of a day, holiday or

not, weather, and POIs. For instance, suppose train m at station u_i is going to leave and the subsequent station of u_i are u_j, u_k, ..., and u_y, the passenger state set $\mathcal{X}$ therefore includes the two kinds of state information of these subsequent stations. Notably, first, the subsequent station amount is variable with the running of a train. That is, the $\mathcal{X}$ is length-variable. Second, there are two types of stations for a metro system, *i.e.,* normal stations and transfer stations. For a normal station, passengers are all entered from the station itself. Whereas passengers in a transfer station are not only entered from the station but also transferred from other lines. Thus, for a normal station, the state just includes flow data of itself, but for a transfer station, we record recent flows of the transfer station itself and all its related stations.

To handle such complex data and reveal information of upcoming passengers, as shown in Figure 4(b), we propose the passenger feature extractor, consisting of transfer station learners, normal station learners, and a conclusive recurrent neural network (RNN). Specifically, we present the transfer station learners and the normal station learners to extract ST correlations and unlock the future flows (that affect the dwell time decision) for transfer stations and normal stations, respectively. Thereafter, we employ the conclusive RNN component to integrate the impacts of these length-variable subsequent stations. Notably, the structure of the passenger feature extractor in Figure 4(b) is constructed according to the train at Station 2, Line 1 of the toy metro system exhibited in Figure 4(a). In this section, we will detail the extractor subsequently.

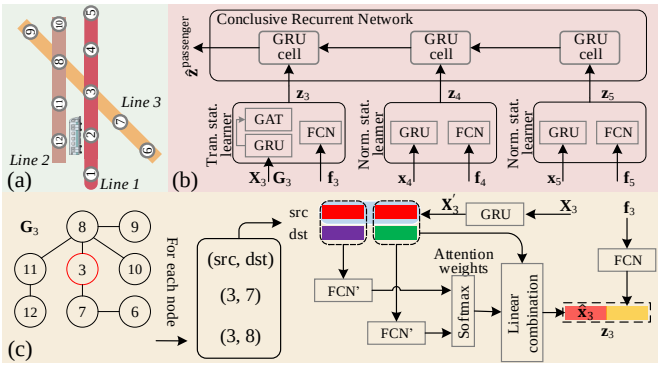


Fig. 4: (a) An example of metro system. (b) Overview of the passenger feature extractor; (c) Structure of the transfer station learner.

3.2.1 Transfer station learner

For a transfer station, in addition to the passengers who swipe the card to enter this station, some passengers transfer from other stations. These transfer behaviors cannot be easily obtained from the check-in data and therefore are complicated to derive the passenger transfer probabilities. Indeed, for a transfer station, the passenger flow at other stations impacts it on two aspects, *i.e.,* spatial and temporal. 1) Spatial impact: Given a transfer station, other stations can affect this station as long as they are connected by the metro lines. 2) Temporal impact: Different stations have different trends of passenger flow at different periods. Also, for a given transfer station, the impacts from these related stations are various in the time dimension. In the transfer station learner, we learn a graph attention network to capture the spatio-temporal impacts, for the following reasons. First, the spatial impact can be modeled by a graph. Concretely, the metro structure can be seen as an undirected graph where stations are connected with. That is the impact graph of the transfer station is the subgraph of the metro structure graph that contains the station. Note that, for a complex system, a given transfer station will connect with too many stations, which will lead to a large graph (high computational complexity). To this end, we use historical data to find the stations with the most interactions with the given transfer station, and prune those infrequent stations from the graph. Second, the temporal impacts can be handled by the attention mechanism. That is the attention mechanism can calculate different weights according to different stations and time, *i.e.,* different degrees of impacts.

Specifically, suppose the first subsequent station u_j is a transfer station, the passenger states of u_j is $(\mathcal{X}_{m,u_i})_1 = (\mathbf{X}_{u_j}, \mathbf{G}_{u_j}, \mathbf{f}_{u_j})$ where $\mathbf{X}_{u_j}$, $\mathbf{G}_{u_j}$, and $\mathbf{f}_{u_j}$ denote the passenger flow matrix, adjacency matrix, and external feature vector of station u_j, respectively. As for the transfer station, other stations may affect its state, we include all these related stations' passenger flows using $\mathbf{X}_{u_j}$. Namely, $\mathbf{X}_{u_j} = [\mathbf{x}_{u_j}, \mathbf{x}_1^{u_j} ..., \mathbf{x}_{n_{u_j}^{\text{neig}}}^{u_j}]$ where $\mathbf{x}_{u_j}$ denotes the historical flows of the transfer station u_j and the rest of vectors represent historical flows w.r.t. $n_{u_j}^{\text{neig}}$ related stations of u_j. Specifically, for an arbitrary station $\mathbf{x}_* = \mathbf{x}_*^{\text{in}} + \mathbf{x}_*^{\text{transfer}}$; $\mathbf{x}_*$, $\mathbf{x}_*^{\text{in}}$, and $\mathbf{x}_*^{\text{transfer}} \in \mathbb{R}^{n^{\text{hist}}}$. Variable n^{hist} is the number of previous timestamps of passenger flow readings. Vector $\mathbf{x}_*^{\text{in}}$ and $\mathbf{x}_*^{\text{transfer}}$ represent passengers who entered from this station and transferred from other lines to the station in each timestamp, respectively. Note that for normal stations, each entry of $\mathbf{x}_*^{\text{transfer}}$ equals 0. Moreover, the spatial relationship of these stations are reserved by a graph, *i.e.,* $\mathbf{G}_{u_j}$. These related stations can be pre-selected according to the passenger trajectory data. Namely, we choose those stations, each of which has many passengers who entered from the station and transferred to the transfer station in history, and construct a graph based on their geographic relationships.

In this learner, we propose to adopt RNNs and graph attention networks (GATs) [18] to capture the complex short-term ST correlations from the passenger flows, as depicted in Figure 4(c) (we use the graph of Station 3 of the metro system in Figure 4(a) to demonstrate the learner). First, we employ an RNN to model the short-term intra-station temporal correlations from the given historical flows. As the gated recurrent unit (GRU) [19] is a simple but effective structure of RNN, we introduce GRU as the specific implementation.

GRU can be defined as $\mathbf{h}_t = \text{GRU}(\mathbf{c}_t, \mathbf{h}_{t-1} | \mathbf{W}_*, \mathbf{U}_*, \mathbf{b}_*)$ formally, in which $\mathbf{c}_t \in \mathbb{R}^D$ and $\mathbf{h}_t \in \mathbb{R}^{D'}$ are the input vector and the encoding state at timestamp t, respectively. $\mathbf{W}_* \in \mathbb{R}^{D' \times D}$ and $\mathbf{U}_* \in \mathbb{R}^{D' \times D}$ are weight matrices; $\mathbf{b}_* \in \mathbb{R}^{D'}$ are bias vectors ($* \in \{u, r, h\}$). Concretely, GRU derives the representation of the hidden state as:

$$\mathbf{u} = \sigma(\mathbf{W}_u \mathbf{c}_t + \mathbf{U}_u \mathbf{h}_{t-1} + \mathbf{b}_u);\ \mathbf{r} = \sigma(\mathbf{W}_r \mathbf{c}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r);$$
$$\mathbf{h}_t = \mathbf{u} \circ \mathbf{h}_{t-1} + (1 - \mathbf{u}) \circ \tanh(\mathbf{W}_h \mathbf{d}_t + \mathbf{U}_h(\mathbf{r} \circ \mathbf{h}_{t-1} + \mathbf{b}_h)) \quad (7)$$

where $\circ$ is the element-wise multiplication and $\sigma(\cdot)$ denotes the sigmoid function.

In the learner, we use a GRU, denoted as GRU^{Ind}, to encode the passenger flow of each station. Given a input vector $\mathbf{x} = [x_1, ..., x_{n^{\text{hist}}}]$, we obtain the hidden state stored in the last states of the GRU cells, *i.e.*, $\mathbf{h}^{\text{Ind}}_{n^{\text{hist}}} = \text{GRU}^{\text{Ind}}(x_{n^{\text{hist}}}, \mathbf{h}^{\text{Ind}}_{n^{\text{hist}}-1} | \mathbf{W}^{\text{Ind}}_*, \mathbf{U}^{\text{Ind}}_*, \mathbf{b}^{\text{Ind}}_*)$, and use the hidden state as the embeddings of historical flows. For simplicity, for station u_j, we denote $(\mathbf{h}^{\text{Ind}}_{n^{\text{hist}}})_{u_j}$ as $\mathbf{x}'_{u_j}$.

Then, to capture the short-term inter-station spatial correlations between u_j and its related stations, we propose to employ GAT in this learner. Concretely, following GAT, we employ a fully-connected network (FCN), denoted as $\mathbf{W}^{\text{St}}$, to calculate a function that provides attention scores between passenger flow embeddings of stations. Then, we use the softmax function to determine the impacts. For instance, the impact weight of z-th neighbor station of u_j, denoted as $a_{j,z}$, can be obtained as follows:

$$e_{j,z} = \phi\left(\mathbf{g}^T\left(\mathbf{W}^{\text{St}}\mathbf{x}'_{u_j}, \mathbf{W}^{\text{St}}\mathbf{x}'_z\right)\right);\ a_{j,z} = \frac{\exp(e_{j,z})}{\sum_{u_c \in \mathcal{N}_{u_j}} \exp(e_{j,c})} \tag{8}$$

where $\mathcal{N}_{u_j}$ is a set of neighbors of u_j in the given graph $\mathbf{G}_{m,u_j}$ and $\mathbf{g}$ is a single-layer feed-forward neural network. The mechanism therefore injects the graph structure into the mechanism by only attending with neighbors in the graph.

Next, for the transfer station, we calculate and output the overall impact of neighbors by linearly combining the hidden states according to the normalized weights and applying a nonlinearity function σ.

$$\hat{\mathbf{x}}_{u_j} = \sigma\left(\sum_{u_c \in \mathcal{N}_{u_j}} a_{j,c}\mathbf{W}^{\text{St}}\mathbf{x}'_{u_c}\right) \tag{9}$$

In addition to the short-term ST correlations, the future flows also can be affected by the long-term factors, such as hours of a day and regional functionalities. We capture such correlations from the external features $\mathbf{f}_{u_j}$ by an FCN, denoted as $\mathbf{W}^{\text{Lo}}$. Finally, we output the new embedding $\mathbf{z}_{u_j}$ of the station with short- and long- term ST correlations: $\mathbf{z}_{u_j} = \hat{\mathbf{x}}_{u_j} || \left(\mathbf{W}^{\text{Lo}}\mathbf{f}_{u_j}\right)$.

3.2.2 Normal station learner

Suppose the second subsequent station u_k is a normal station, $(\mathcal{X}_{m,u_i})_2 = (\mathbf{x}_{u_k}, \mathbf{f}_{u_k})$ is the input of the learner. Since a normal station is not affected by other stations, the state only has its flow vector $\mathbf{x}_{u_k}$. Also, $\mathbf{f}_{u_k}$ is the external feature vector of u_k.

In this learner, we use the same GRU of the transfer station learner, *i.e.*, GRU^{Ind}, to capture the short-term intra-station temporal correlations of normal stations. Given the input $\mathbf{x}_{u_k}$ to GRU^{Ind}, we also obtain and output the hidden state stored in the last states of the GRU cells, denoted as $\hat{\mathbf{x}}_{u_k}$. As well, we adopt the same FCN, *i.e.*, $\mathbf{W}^{\text{Lo}}$, to embed the external features $\mathbf{f}_{u_k}$. In sum, the output embedding of the normal station u_k is $\mathbf{z}_{u_k} = \hat{\mathbf{x}}_{u_k} || \left(\mathbf{W}^{\text{Lo}}\mathbf{f}_{u_k}\right)$.

3.2.3 Conclusive recurrent network

Because the number of subsequent stations is variable, we propose to use a new RNN to collectively weight and integrate the impacts of these stations based on these obtained embeddings of these stations. Specifically, after learned by the above two learners, the input of the RNN is obtained as $[\mathbf{z}_{u_y}, ..., \mathbf{z}_{u_k}, \mathbf{z}_{u_j}]$. The order of the input is based on the reverse of distance from station u_i, considering the impacts of the distance. Also, we use GRU as the implementation of RNN, denoted as GRU^{Con}, and output the hidden vector of the last states of the GRU cells, *i.e.*, $\hat{\mathbf{z}}^{\text{passenger}}_{m,u_i} = \text{GRU}^{\text{Con}}(\mathbf{z}_{u_j}, \mathbf{h}^{\text{Con}}_{u_k} | \mathbf{W}^{\text{Con}}_*, \mathbf{U}^{\text{Con}}_*, \mathbf{b}^{\text{Con}}_*)$, as the passenger information embedding with complex ST correlations of these subsequent stations.

3.3 Fusion Component

To fuse the two influential aspects of knowledge and reveal the Q-value for each action, we propose the fusion component. Concretely, we feed the outputs of the above two components to FCNs to model their latent correlations and estimate the Q-value for each action. Specifically, for train m that is heading to station u_i, we denote the estimated Q-values as $\mathbf{q}_{m,u_i}$, which can be obtained as follows.

$$\mathbf{q}_{m,u_i} = \mathbf{D}_c(\sigma(...\mathbf{D}_1(\sigma(\hat{\mathbf{d}}^{\text{train}}_{m,u_i} || \hat{\mathbf{z}}^{\text{passenger}}_{m,u_i}))...)) \tag{10}$$

where $\{\mathbf{D}_1, ..., \mathbf{D}_c\}$ represents c dense layers.

3.4 Algorithm of Optimization

Following the basic deep Q-learning framework [13], [16], [20], the network can be optimized end-to-end. Algorithm 1 shows the learning algorithm of AutoDwell. Specifically, we first initialize the deep network. Then, for each episode, AutoDwell assigns the dwell time for a train, and stores the states, actions, and immediate rewards in the replay memory (Lines 3-9). Next, AutoDwell randomly selects a mini-batch of samples from the replay memory to train the policy (Lines 10-13). Finally, after the network is converged, we can obtain the scheduling policy.

4 Experiments

The code of AutoDwell has been released[2]. In this section, we conduct experiments to evaluate AutoDwell.

4.1 Experimental Settings

4.1.1 Datasets

In the experiment, we build emulators to train and evaluate our model based on the real-world datasets collected from Beijing and Hangzhou, China. The two real-world datasets are described as follows. More statistics of the two datasets are summarized in Table 1.

1) **Beijing**. This dataset is obtained from Beijing Metro Network Control Center[3], containing a large number of check-in and -out records for ten downtown lines in Beijing (11 days, 9/17/2018 - 9/21/2018 and 9/25/2018 - 9/30/2018). The locations of these ten lines in the Beijing Metro Map are depicted in Figure 5(a). The spatial and temporal distributions of the check-in records are visualized in Figure 5(b) and (c). Moreover, there are a lot of transfer

2. https://github.com/AutoDwell/AutoDwell.git
3. https://www.bjsubway.com

Algorithm 1: Training algorithm of AutoDwell

Input : A small probability ϵ and a discount parameter γ
Initialize : A replay memory $\mathcal{D}$ and parameters θ of AutoDwell.

for *episode* $= 1, n^{episode}$ **do**
  for $t = 1, n^{dwell\ process}$ **do**
    generate a random value $\hat{\epsilon} \in [0, 1]$
    if $\hat{\epsilon} < \epsilon$ **then**
      select a random action a_t
    else
      select $a_t = \text{argmax}_a Q(s_t, a_t; \theta)$; $s_t = \{\mathcal{C}_t, \mathcal{X}_t\}$
    execute a_t in the emulator and observe immediate reward r_t and new state $s_{t+1} = \{\mathcal{C}_{t+1}, \mathcal{X}_{t+1}\}$;
    store transaction (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$;
    sample random minibatch $\mathcal{M}$ of transaction from $\mathcal{D}$
    for $i \in \{1, ..., |\mathcal{M}|\}$ **do**
      set $y_t^i =$
      $\begin{cases} r_t^i, & \text{if } s_{t+1}^i \text{ is a terminal state;} \\ r_t^i + \gamma \text{argmax}_a Q(s_{t+1}^i, a'^i; \theta), & \text{if } s_{t+1}^i \text{ is a non-terminal state.} \end{cases}$
      perform a gradient descent step on $(y_t^i - Q(s_t^i, a_t^i; \theta))^2$

Output : learned AutoDwell

TABLE 1: Descriptions for the two real-world datasets

Statistical levels & indicators			Beijing	Hangzhou
System	# lines		10	3
	# stations		182	66
	# transfer stations		36	5
	Transfer ratio		0.59	0.31
	Total length		274.34km	89.99km
	# daily records	aver.	4,215,786.91	1,154,317
		SD	163,251.11	88,848.90
	# stations for a trip	aver.	9.84	7.54
		SD	5.47	4.80
	Trip time	aver.	1,923.71s	1,515.50s
		SD	905.76	840.72
	# lines for a transfer trip	aver.	1.38	1.06
		SD	0.52	0.24
Line	# stations of one line	max	45	32
		min	13	18
		aver.	21.80	23.67
		SD	8.42	6.02
	# of trips per day & line	max	757,302	594,262
		min	170,982	155,484
		aver.	351,336.70	350,789.30
		SD	167,671.38	182,339.6
Station	# of check-in records per day & station	max	84,561	93,372
		min	2,051	3,418
		aver.	19,304.21	16,677.12
		SD	12,176.76	13,279.19
	Distance between two neighbors	max	3.00km	3.32km
		min	0.42km	0.60km
		aver.	1.31km	1.32km
		SD	0.43	0.48

trips between these ten lines, and the transfer probability is shown in Figure 5(d). In our experiment, we use the first 8 days' data as the training environment and the last 2 days' data as the testing environment.

2) **Hangzhou**. The dataset can be accessed from an open website[4]. The check-in and -out records are from 1/1/2019 to 25/1/2019 for three lines of Hangzhou. We also show the metro map, the spatial/temporal distribution of daily records, and the transfer probability in Figure 5(e) - (h), respectively. In this experiment, we adopt the first 20 days' data to train our model and use the rest of the 5 days' data to be the testing environment.

4.1.2 Emulator Settings

We use real-world settings and trajectories to build the emulators Specifically, the structures of the two real-world systems are used to construct emulators, respectively. Trains in the emulators have six carriages that can totally hold 1500 passengers (n^{capa}). The fixed part of the dwell process, *i.e.,* δ^{fixed}, is set to $46s$ that includes $25s$ for entering the platform to stopping, $15s$ for leaving the platform, and $6s$ to open and close doors. The minimum and maximum dwell time are set as $\delta^{\min} = 58s$ and $\delta^{\max} = 74s$, respectively. For simplicity, the dwell time range, *i.e.,* $[\delta^{\min}, \delta^{\max}]$, are uniformly discretized into $n^{\text{action}} = 5$ intervals, each of which represents an action. The operation time for both the two cities is limited from 7:00 to 21:00. A real-world timetable is used where the departure interval ranges from 2-6 minutes for Beijing and 2.5-9 minutes for Hangzhou [5].

4. https://tianchi.aliyun.com/competition/entrance/231708/information

5. http://hz.bendibao.com/traffic/20181022/73829.shtm

We adopt two settings for the maximum speed of trains. 1) We calculate the maximum speed for each adjacent station pair in the data and let it be the maximum speed. The setting way is denoted as $\mathcal{P}_1$. 2) to better simulate the reality, the second way, denoted as $\mathcal{P}_2$, uses the hourly maximum speed for every two adjacent stations in the data. In addition, according to [21], [22], $\delta^{\text{mixed}}$is set to $2s$, and the boarding and alighting velocities of a train (v^{boar} and v^{alig}) are both equal to 12 passengers/s.

4.1.3 Evaluation Metrics

We use the following two metrics to validate the effectiveness of our model on shortening the travel time of passengers, *i.e.,* average waiting time $\overline{\delta^{\text{waiting}}}$ and average journey time $\overline{\delta^{\text{journey}}}$:

$$\overline{\delta^{\text{waiting}}} = \frac{1}{n^{\text{trip}}} \sum_{i}^{n^{\text{trip}}} \delta_{p_i}^{\text{waiting}}; \qquad \overline{\delta^{\text{journey}}} = \frac{1}{n^{\text{trip}}} \sum_{i}^{n^{\text{trip}}} \delta_{p_i}^{\text{journey}}$$

where n^{trip} is the number of trips in the testing environment. For a trip p_i, we denote the waiting time and journey time as $\delta_{p_i}^{\text{waiting}}$ and $\delta_{p_i}^{\text{journey}}$, respectively. Note that, $\delta_{p_i}^{\text{waiting}} + \delta_{p_i}^{\text{journey}}$ is the total travel time of trip p_i.

4.1.4 Models

Five groups of models are compared in the experiment.

1) **Fixed Dwell Time Model (FM)**. In the group, three models are provided to demonstrate the performances of the simplest scheduling strategies. That is the dwell

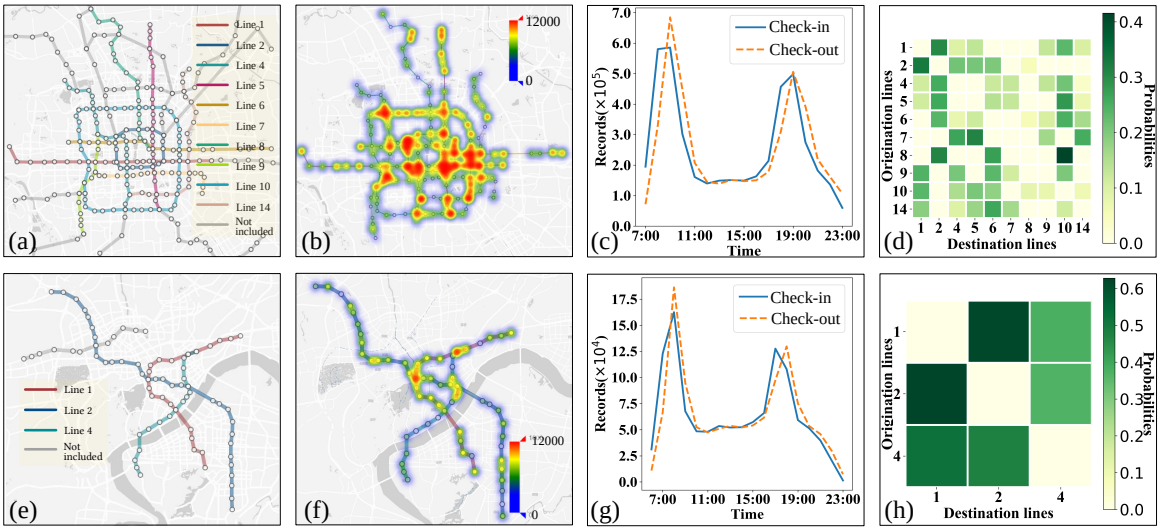


Fig. 5: Visualization of our datasets (best viewed in color). (a) Metro map of Beijing; (b) Daily spatial distribution of check-in records in Beijing; (c) Daily temporal distribution of check-in records in Beijing; (d) Transfer probabilities between lines of Beijing; (e) Metro map of Hangzhou; (f) Daily spatial distribution of check-in records in Hangzhou; (g) Daily temporal distribution of check-in records in Hangzhou; (h) Transfer probabilities between lines of Hangzhou.

time of any of these models in the group is fixed for all stations. Concretely, the three models set the dwell time as δ_{Min}, δ_{Max}, and $(\delta_{\text{Min}} + \delta_{\text{Max}})/2$, and are denoted as *Min*, *Max*, and *Aver*, respectively.

2) **Historical Data-Based Model (HM)**. In the group, the dwell time of a station is determined according to the historical passenger flows, revealing the performances of offline scheduling strategies. We use a toy example to describe the group's strategy. Suppose there are three stations on one line and they account for [20%, 30%, and 50%] of the whole line's flow respectively at a specific interval in the historical data. Then, we use the min-max approach to normalize the vector to [0.00, 0.66, 1.00] and select the dwell time as $\delta^{\min}$, $0.34\delta^{\max}$+$0.66\delta^{\min}$, and $\delta^{\min}$ for the three stations in the specific interval of our environment, respectively. There are two models in the group, denoted as *Day* and *Hour*, which set the interval for a day and an hour, respectively.
3) **Optimization-Based Model (OM)**. Models in the group are proposed for optimizing energy consumption and passenger waiting time simultaneously. To adapt to our scenario, we only retain the waiting time objective and use our dwell settings. More specifically, each line corresponds to a model, and every hour we use the check-in data at the corresponding time of the previous day to calculate the dwell time. We denote two models of the group as *BNP* [23] and *MLP* [24], respectively.
4) **Prediction-Based Model (PM)**. Models in this group can dynamically schedule the dwell time by a dwell time assessment method [7]. The assessment method can determine the dwell time based on the current passenger flows. That is we need a flow prediction method to provide such information to the assessment method in advance. Thus, with two different flow prediction methods, *i.e.*, ARIMA [25] and vanilla RNN [19], two models are in the group, denoted as *ARIMA* and *RNN*, respectively.
5) **Prediction-Based Models for Multiple Stations (PMM)**. Similar to the PMs, models in this group determine the dwell time dynamically based on the dwell time assessment method [7]. Different from PMs that determine the dwell time by only referring to the next station, models in the group aggregate the prediction results according to the inverse ratio of the distance between the train's current station and all these subsequent stations. We also use ARIMA and vanilla RNN as prediction models, and the two models are denoted as *ARIMA* and *RNN*, respectively.
6) **AutoDwell**. As there are two vital components in AutoDwell, in addition to the model, we also validate two variants of AutoDwell to evaluate their effectiveness. We denote three models in the group as *w/o-T*, *i.e.*, AutoDwell without the train feature extractor, *w/o-P*, *i.e.*, AutoDwell without the passenger feature extractor, and *w-T&P*, *i.e.*, the AutoDwell. The parameter settings of AutoDwell are as follows. 1) Train Feature Extractor: $\mathbf{W}^{\text{Tr}}$ is a two-layer FCN with units [32, 8]. 2) Passenger Feature Extractor: the length of the timestamp for historical readings l^{hist} is set to $60s$ and $n^{\text{hist}} = 60$. Besides, (a) Transfer station learner: $\mathbf{W}^{\text{St}}$ is a two-layer FCN with units [32, 8]; (b) Transfer/Normal station learner: the dimension of GRU^{Ind}'s hidden state $\mathbf{h}^{\text{Ind}}$ is 8 and $\mathbf{W}^{\text{Lo}}$ is a two-layer FCN with units [32, 8]; and (c) Conclusive recurrent network: the dimension of GRU^{Con}'s hidden state $\mathbf{h}^{\text{Con}}$ is 16. 3) Fusion Component: $\mathbf{D}_1$ is a two-layer FCN with units [64, 5]. That is the dimension of the last layer (*i.e.*, 5) corresponds to the number of actions. Note that, the sensitivities of the parameters will be discussed in Section 4.5.

4.2 Effectiveness of AutoDwell

The results of each model in terms of $\overline{\delta^{\text{waiting}}}$ and $\overline{\delta^{\text{journey}}}$ are reported in Table 2. Overall, by capturing complex ST corre-

TABLE 2: Overall performances

Methods		Beijing						Hangzhou					
		$\overline{\delta^{\text{waiting}}}$	$\frac{\mathcal{P}_1}{\delta^{\text{journey}}}$	Sum	$\overline{\delta^{\text{waiting}}}$	$\frac{\mathcal{P}_2}{\delta^{\text{journey}}}$	Sum	$\overline{\delta^{\text{waiting}}}$	$\frac{\mathcal{P}_1}{\delta^{\text{journey}}}$	Sum	$\overline{\delta^{\text{waiting}}}$	$\frac{\mathcal{P}_2}{\delta^{\text{journey}}}$	Sum
FM	*Min*	181.56	1714.84	1896.39	179.77	1713.88	1893.65	205.62	1297.74	1503.35	204.43	1295.02	1499.45
	Max	215.43	1723.48	1938.90	216.19	1720.15	1936.34	215.85	1303.44	1519.29	205.00	1298.36	1503.36
	Aver	189.16	1715.59	1904.76	186.86	1715.54	1902.40	212.65	1300.59	1513.24	204.79	1296.28	1501.07
HM	*Day*	166.06	1708.72	1874.79	167.27	1705.34	1872.60	196.81	1289.93	1486.74	199.02	1285.06	1484.08
	Hour	165.13	1703.13	1868.26	163.72	1701.55	1865.26	195.74	1290.25	1485.98	196.59	1286.22	1482.82
OM	BNP	158.23	1724.18	1882.41	157.47	1722.77	1880.24	184.54	1301.89	1486.43	182.79	1300.03	1482.82
	MLP	153.81	1719.69	1873.50	151.08	1718.12	1869.20	181.59	1299.07	1480.66	179.52	1297.92	1477.44
PM	*ARIMA*	173.24	1710.59	1883.83	173.94	1707.82	1881.76	203.03	1295.03	1498.06	198.41	1287.02	1485.43
	RNN	176.06	1709.13	1885.19	174.02	1707.25	1881.27	204.45	1297.25	1501.69	196.94	1286.48	1483.42
PMM	*ARIMA*	163.64	1701.88	1865.51	162.58	1700.85	1863.43	195.53	1286.75	1482.28	194.62	1285.56	1480.18
	RNN	162.56	1701.65	1864.21	161.14	1700.92	1862.06	193.98	1285.84	1479.82	191.30	1284.89	1476.19
AutoDwell	*w/o-P*	173.28	1695.55	1868.82	171.11	1693.24	1864.35	209.56	1283.66	1493.22	204.74	1282.40	1487.14
	w/o-T	150.24	1699.15	1849.39	146.84	1697.93	1844.77	176.19	1287.24	1463.43	173.44	1286.91	1460.34
	w-P&T	**148.07**	**1694.78**	**1842.86**	**145.70**	**1692.49**	**1838.19**	**174.40**	**1281.56**	**1455.97**	**171.14**	**1280.94**	**1452.08**

lations and interactions of trains, the proposed AutoDwell shows the best performance over the other four categories of baselines. In particular, compared with the best baseline, *i.e.*, PMM-*RNN*, the AutoDwell can save at least about 21s and 24s travel time for every passenger in the emulation of Beijing and Hangzhou, respectively. In other words, when applying to the real world, it can save millions of minutes a day for a tremendous number of passengers. Moreover, it can be found that AutoDwell can decrease at least about 9% waiting time of passengers, which is capable of boosting passengers' experience significantly as travel time plays an important role in passengers' satisfaction [26].

More specifically, we compare our model with each group. 1) FM. Since the dwell process is complex, directly assigning a fixed dwell time for all trains is too simple to obtain satisfying results. 2) HM. As historical knowledge can reveal the general trend of the passenger flows, models in this group can achieve better results. 3) OM. The performances of the two methods are acceptable in terms of the average waiting time. However, without considering the long-term effects, their dwell decision will increase the journal time, thus making their overall performance degrades significantly. 4) PM. Compared with our model, the performances of PM models are limited, because of two aspects. First, the results of the dwell assessment method are depended on the accuracy of prediction methods. Thus, without capturing the dynamic spatial correlations, these prediction methods deteriorate the performance of the PM models. Second, PM models overlook the interactions of these related trains. 5) PMM. By involving simple spatial knowledge, models in this group perform better than that of PM. However, our model can beat PMM models easily because we carefully design the spatial knowledge learner that can capture more complex spatial dependencies. 6) AutoDwell. The results of competing between *w-T&P* (*i.e.*, AutoDwell) with *w/o-T* and *w/o-P* show the effectiveness of the passenger feature extractor and the train feature extractor, respectively.

To further understand our policy, we demonstrate the performance results w.r.t. each period by comparing our model with the best baseline, *i.e.*, PMM-*RNN*, as reported in Table 3. The results indicate that AutoDwell outperforms PMM-*RNN* in each period, in particular these peak periods with very high passenger pressures such as 7:00 - 9:00am (see Figure 5(c) and (g)). That is having a good dwell timing is crucial in peak hours where the train resources and passenger demands are extremely imbalanced, and AutoDwell can effectively alleviate the imbalance, indicating the practicality of our model.

4.3 Effectiveness of Passenger Feature Learner

From Section 4.2, we learned that the passenger feature extractor is the most effective part of the model. Thus, in this section, we report an experiment to further understand the extractor.

Since the aim of the passenger feature extractor is to capture the dynamic passenger states, to a certain extent, we can verify its effectiveness by predicting passenger flows. In this experiment, we respectively select ten stations of Beijing and Hangzhou, and use a slide-window-based method to generate training and testing data. Specifically, we input the historical flow of passengers in the previous one hour to the below four networks (the passenger feature extractor and its three variants) to predict the flow in the next minute. Note that, to adapt the passenger feature extractor to this task, we use the extractor and its variants as prefix networks and add two dense layers as the predictor. The results in terms of mean absolute percentage error (MAPE; $\frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i|/y_i$, where n is the number of values, y_i is the ground truth, and $\hat{y}_i$ is the prediction value) are reported in Table 4.

1) P_1: The passenger feature extractor without the transfer station learner; We directly use the normal station learner to handle these transfers stations;
2) P_2: The passenger feature extractor without the conclusive recurrent network. To aggregate the variable length output of the normal and transfer station learners, we take an average of them;
3) P_3: We use Hadamard-product to replace the conclusive recurrent network;
4) P: The passenger feature extractor.

The results of comparing P_1 and P indicate that the transfer station learner can significantly improve perfor-

TABLE 3: Performance results w.r.t. each time period

	Metric	Method	7:00-9:00	9:00-11:00	11:00-13:00	13:00-15:00	15:00-17:00	17:00-19:00	19:00-21:00
Beijing	$\overline{\delta^{\text{waiting}}}$	PMM-*RNN*	233.428	142.205	127.137	129.810	112.541	106.158	159.175
		AutoDwell	210.378	133.208	126.286	129.297	112.398	103.763	153.209
	$\overline{\delta^{\text{journey}}}$	PMM-*RNN*	1689.196	1695.268	1715.886	1723.092	1707.954	1702.388	1711.811
		AutoDwell	1682.477	1709.562	1706.726	1711.088	1698.897	1687.217	1705.147
Hangzhou	$\overline{\delta^{\text{waiting}}}$	PMM-*RNN*	276.266	172.569	140.828	142.099	141.554	133.435	198.924
		AutoDwell	265.006	164.368	140.089	138.848	139.332	131.133	185.996
	$\overline{\delta^{\text{journey}}}$	PMM-*RNN*	1275.717	1287.449	1311.823	1308.448	1312.541	1307.333	1316.809
		AutoDwell	1270.679	1283.608	1295.232	1301.165	1306.574	1299.132	1304.635

mance. For a transfer station, in addition to the passengers who swipe the card to enter this station, some passengers transfer from other stations. These transfer behaviors cannot be obviously obtained from the check-in data and therefore cannot be handled by the normal station learner. Hence, by carefully introducing the RNN and graph attention network, the transfer station learner is capable of capturing the transfer probabilities and preserving complex the passenger dynamics. The comparison between P_2, P_3, and P reflects the effectiveness of the conclusive recurrent network, since the average and Hadamard-product fusion operators fail to capture the spatio-temporal information between stations and thus prevent them to achieve high accuracy. In sum, with these carefully designed network components, the passenger feature extractor therefore can capture the dynamic passenger information effectively.

TABLE 4: Performances on flow prediction

	MAPE	
	Beijing	Hangzhou
P_1	29.7%+0.8%	25.4%+0.6%
P_2	23.8%+0.4%	21.3%+0.2%
P_3	25.1%+1.2%	22.4%+0.9%
P	20.4%+0.6%	18.6%+0.3%

4.4 Case Study

In this section, we provide two case studies, exhibited in Figure 6, using data generated in our experiment. The case in Figure 6(a) shows our dynamic policy that considers both passenger flows and context trains with long-term impacts is superior to the dwell time fixed way in the real-world. 1) Though there are only a few passengers at Station 3 and many passengers at Station 5, our policy selects a long dwell time for Train I. The reason is that our policy considers there will be many upcoming passengers at Station 3 and the front train (*i.e.,* Train II) can serve passengers in Station 5. Thus, the dwell decision can benefit more passengers. 2) For Train III, Although AutoDwell can aware that Station 8 has some upcoming passengers, the policy decides a short dwell time at Station 8 because Station 9 has a large number of waiting and upcoming passengers. That is, the policy can serve these over-waited passengers as soon as possible. As shown in Figure 6(b), we select three different types of stations in Beijing to compare the number of check-in passengers with the average dwell time decided by the model among different periods of a day. The three stations are Anheqiao, Datunlu, and Dongdan, which locate in the residential area, working area, and working-residential mixed area, respectively. Because they are in different functional areas, these stations show various passenger distributions in time, as demonstrated by dot lines of Figure 6(b). The comparison between dot lines and their corresponding dash lines (average dwell time) reveals our model can well capture these different temporal patterns to decide the dwell time. Notably, as the model determines dwell time by considering not only the passenger flow distributions but also the status of context trains, the trend of dwell time is not strictly consistent with the trend of passenger flows. For example, even the passenger flow of Dongdan is more than that of Datunlu, the dwell time of Dongdan may be less than that of Datunlun.

4.5 Parameter Sensitivities

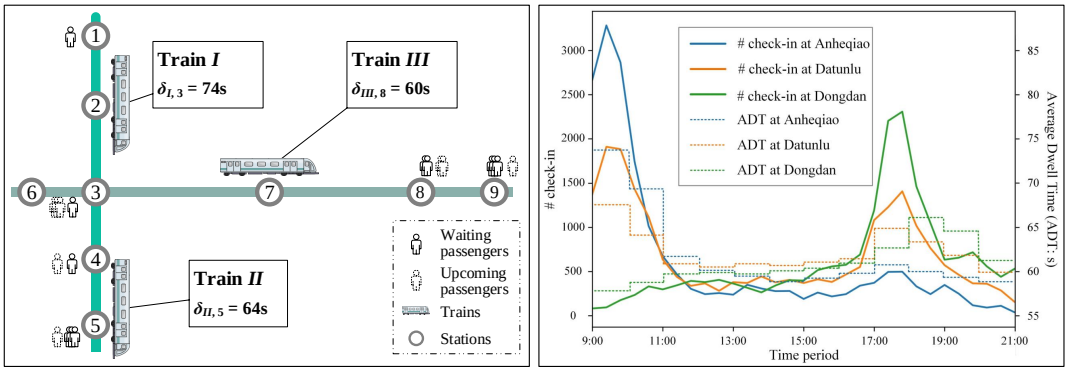
In this section, we conduct experiments to analyze the parameter sensitivities of the model. For each parameter, we adjust it from a reasonable range with other parameters fixed. For simplicity, we report results in terms of the summation of $\overline{\delta^{\text{waiting}}}$ and $\overline{\delta^{\text{journey}}}$ on the dataset of Beijing under the condition of the $\mathcal{P}_1$, as depicted in Figure 7.

It can be seen that our model is robust to the dimensions of the output state of $\mathbf{W}^{\text{Tr}}$, the hidden state of $\mathbf{W}^{\text{St}}$, the output state of $\mathbf{W}^{\text{St}}$, the hidden state of $\mathbf{W}^{\text{Lo}}$, the output state of $\mathbf{W}^{\text{Lo}}$, and the hidden state of $\mathbf{D}_1$, as shown in Figure 7(b), (f), (g), (h), (i), and (k), respectively. For the dimension of the hidden state of $\mathbf{W}^{\text{Tr}}$, n^{hist}, the hidden state of GRU^{Ind}, the hidden state of GRU^{Con}, and the hidden state of $\mathbf{D}_2$, performances of our model are not sensitive to the change of them after each of them reaches a specific value, as depicted in Figure 7(a), (d), (e), (j), and (l), respectively. Moreover, Figure 7(c) indicates that the performance is insensitive to the change of l^{hist} when it is small. When l^{hist} is large, more redundant information has been brought and the performance therefore decreases.

5 Related Work

To position our work in the research community, we study two categories of related works as follows.

Scheduling in Metro System. Conventionally, many works aimed to pre-determine the arrival and departure time for each train at each station [27]–[29] . Then, some works tried to reschedule the timetable after some emergency circumstance occurred (*e.g.,* an accident), to quickly resume the metro system [30]–[32]. However, these approaches take very limited information about the dynamic



(a) Spatial case study (b) Temporal case study

Fig. 6: Case study of our model

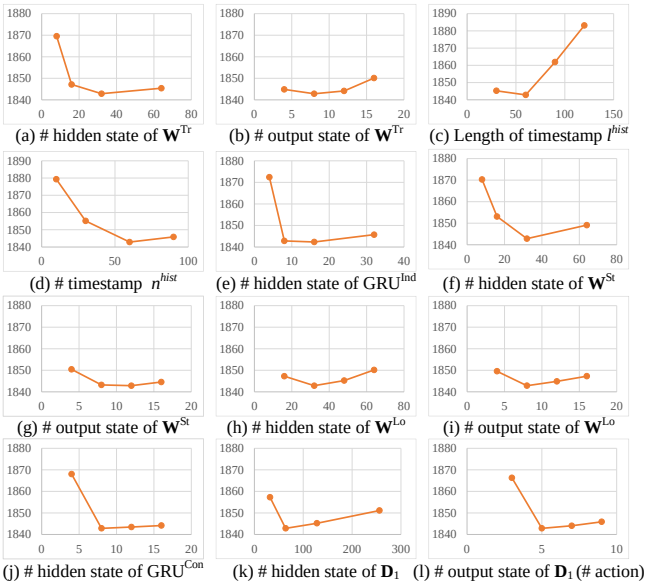


Fig. 7: Results of parameter sensitivities

and diverse distributions of incoming passengers into consideration and consequently they can not be applied to tackle our issue; Then, some researchers studied the impacts of different dwell time on passengers' travel time [4]–[7]. They learned a function that maps the quantity of waiting passengers to an appropriate dwell time such that trains can carry more passengers. Nevertheless, their performances for shortening the average travel time of passengers are limited as two aspects: 1) These methods only consider the impact of the dwell process on the current station, can not capture these long-term impacts on the overall travel time; and 2) These prior works overlook the interactions between trains. Recently, Yin et al. [23] and Yang et al. [24]'s work open a new branch of optimizing both energy consumption and passenger waiting time with consideration of real-world smart-card data, which is an important and realistic issue. However, due to the following three aspects, their work is fundamentally different from ours. First, unlike the bi-objective problem of the two papers, we only optimize passengers' travel time, including the waiting time at platforms and the journal time on trains. Specifically, the energy consumption of the train is mainly determined by the speed. As a common sense, we can easily shorten the travel time of passengers by speeding up the train, but at the same time it brings more energy consumption. Hence, these two articles model the two conflicting goals as a bi-objective optimization problem, while our research only optimize the travel time under a given speed profile (*i.e.*, energy consumption fixed). Second, real-world metro systems usually consists of multiple lines, and these lines are will affect each other. These two papers, however, only consider the optimization of a single line, but our work can work in such complex systems. Third, the dwell process model in the two papers are too simple. Particularly, their model assumes every waiting passengers can take the new arrival train. In fact, in the real world, due to the limitation of vehicle capacity, passengers are often unable to board the train, especially in the morning and evening rush hours. To this end, our work considers the vehicle capacity, such that we can better learn the model to optimize the real-world problem.

Being different from all the above works, our study employs a DRL based model to optimize the long-term impacts of dwell time settings. Furthermore, we model the spatio-temporal correlations of incoming passengers and the interactions between trains on the same line, which makes our model have the more necessary information to improve the efficiency of the metro system.

Deep Reinforcement Learning. Compared with vanilla reinforcement learning [33], the deep reinforcement learning [34] leverages the power of deep neural networks, thereby leading to the improvement on the performance of many challenging applications, such as Go [35] and Atari games [36]. Nowadays, the DRL has achieved success in tackling complicated issues in the field of urban computing, like bike reposition [13], order dispatching [14], [15], supply-demand balancing [16], ambulances redeployment [12], etc. Likewise, in this paper, we propose to employ DRL to address an urban transportation issue, *i.e.*, dwell time scheduling in metro systems.

6 Conclusion

In this paper, we explored a deep network to dynamically schedule trains' dwell time for the efficiency improvement

of a metro system. We optimized AutoDwell by a reinforcement learning framework by weighing the passengers' waiting time on platforms and journey time on trains and capturing the long-term impacts. Moreover, in AutoDwell, we devised a passenger feature extractor to capture complex spatio-temporal correlations of passenger flows, and a train feature extractor to model interactions between trains, respectively, providing necessary information to guide the action selection. Finally, we evaluated our model on two real-world datasets and the results showed that our model achieves better performance beyond several baselines, capable of saving passengers' overall travel time.

ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China (2019YFB2101801) and the National Natural Science Foundation of China (61773324, 72061127001, and 72171013).

REFERENCES

[1] J. Choi, J. F. Coughlin, and L. D'Ambrosio, "Travel time and subjective well-being," *Transportation Research Record*, vol. 2357, no. 1, pp. 100–108, 2013.
[2] C. H. DiMaria, C. Peroni, and F. Sarracino, "Happiness matters: Productivity gains from subjective well-being," *Journal of Happiness Studies*, pp. 1–22, 2017.
[3] X. Yang, X. Li, B. Ning, and T. Tang, "A survey on energy-efficient train operation for urban rail transit," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 1, pp. 2–13, 2015.
[4] P. Kecman and R. M. Goverde, "Predictive modelling of running and dwell times in railway traffic," *Public Transport*, vol. 7, no. 3, pp. 295–319, 2015.
[5] S. Seriani and R. Fernandez, "Pedestrian traffic management of boarding and alighting in metro stations," *Transportation Research Part C: Emerging Technologies*, vol. 53, pp. 76–92, 2015.
[6] L. D'Acierno, M. Botte, A. Placido, C. Caropreso, and B. Montella, "Methodology for determining dwell times consistent with passenger flows in the case of metro services," *Urban Rail Transit*, vol. 3, no. 2, pp. 73–89, 2017.
[7] S. Cornet, C. Buisson, F. Ramond, P. Bouvarel, and J. Rodriguez, "Methods for quantitative assessment of passenger flow influence on train dwell time in dense traffic areas," 2019.
[8] M. Pead, "The impact of boarding and alighting passengers on the dwell time at railway stations," *Aston Univ., Birmingham, UK*, 2007.
[9] B. Leng, J. Zeng, Z. Xiong, W. Lv, and Y. Wan, "Probability tree based passenger flow prediction and its application to the beijing subway system," *Frontiers of Computer Science*, vol. 7, no. 2, pp. 195–203, 2013.
[10] H. Lee, D. Zhang, T. He, and S. H. Son, "Metrotime: Travel time decomposition under stochastic time table for metro networks," in *IEEE International Conference on Smart Computing*, 2017, pp. 1–8.
[11] J. Zhao, F. Zhang, L. Tu, C. Xu, D. Shen, C. Tian, X. Li, and Z. Li, "Estimation of passenger route choice pattern using smart card data for complex metro systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 4, pp. 790–801, 2017.
[12] S. Ji, Y. Zheng, Z. Wang, and T. Li, "A deep reinforcement learning-enabled dynamic redeployment system for mobile ambulances." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 1, pp. 15:1–15:20, 2019.
[13] Y. Li, Y. Zheng, and Q. Yang, "Dynamic bike reposition: A spatio-temporal reinforcement learning approach," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1724–1733.
[14] Z. Wang, Z. Qin, X. Tang, J. Ye, and H. Zhu, "Deep reinforcement learning with knowledge transfer for online rides order dispatching," in *2018 IEEE International Conference on Data Mining*, 2018, pp. 617–626.
[15] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning," *WWW*, pp. 983–994, 2019.
[16] S. He and K. G. Shin, "Spatio-temporal capsule-based reinforcement learning for mobility-on-demand network coordination," *WWW*, pp. 2806–2813, 2019.
[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv:1409.0473*, 2014.
[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ArXiv Preprint ArXiv:1710.10903*, 2017.
[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *ArXiv Preprint ArXiv:1412.3555*, 2014.
[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI*, 2016.
[21] Q. Zhang, B. Han, and D. Li, "Modeling and simulation of passenger alighting and boarding movement in beijing metro stations," *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 5, pp. 635–649, 2008.
[22] F. Chen, W. U. Qibing, H. Zhang, L. I. Sanbing, and L. Zhao, "Relationship analysis on station capacity and passenger flow: A case of beijing subway line 1," *Journal of Transportation Systems Engineering and Information Technology*, vol. 9, no. 2, pp. 93 – 98, 2009.
[23] J. Yin, L. Yang, T. Tang, Z. Gao, and B. Ran, "Dynamic passenger demand oriented metro train scheduling with energy-efficiency and waiting time minimization: Mixed-integer linear programming approaches," *Transportation Research Part B: Methodological*, vol. 97, pp. 182–213, 2017.
[24] S. Yang, J. Wu, H. Sun, X. Yang, Z. Gao, and A. Chen, "Bi-objective nonlinear programming with minimum energy consumption and passenger waiting time for metro systems, based on the real-world smart-card data," *Transportmetrica B: Transport Dynamics*, vol. 6, no. 4, pp. 302–319, 2018.
[25] J. Durbin and S. J. Koopman, *Time series analysis by state space methods*. Oxford university press, 2012.
[26] F. Bielen and N. Demoulin, "Waiting time influence on the satisfaction-loyalty relationship in services," *Managing Service Quality: An International Journal*, vol. 17, no. 2, pp. 174–193, 2007.
[27] H. Niu and X. Zhou, "Optimizing urban rail timetable under time-dependent demand and oversaturated conditions," *Transportation Research Part C-emerging Technologies*, vol. 36, pp. 212–230, 2013.
[28] K. Li, H. Huang, and P. Schonfeld, "Metro timetabling for time-varying passenger demand and congestion at stations," *Journal of Advanced Transportation*, vol. 2018, pp. 1–26, 2018.
[29] L. Sun, J. G. Jin, D. Lee, K. W. Axhausen, and A. Erath, "Demand-driven timetable design for metro services," *Transportation Research Part C-emerging Technologies*, vol. 46, pp. 284–299, 2014.
[30] V. Cacchiani, D. Huisman, M. P. Kidd, L. Kroon, P. Toth, L. P. Veelenturf, and J. Wagenaar, "An overview of recovery models and algorithms for real-time railway rescheduling," *Transportation Research Part B - Methodological*, vol. 63, no. 63, pp. 15–37, 2014.
[31] J. Yin, T. Tang, L. Yang, Z. Gao, and B. Ran, "Energy-efficient metro train rescheduling with uncertain time-variant passenger demands: An approximate dynamic programming approach," *Transportation Research Part B - Methodological*, vol. 91, pp. 178–210, 2016.
[32] Y. Gao, L. Kroon, M. Schmidt, and L. Yang, "Rescheduling a metro line in an over-crowded situation after disruptions," *Transportation Research Part B - Methodological*, vol. 93, pp. 425–449, 2016.
[33] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237–285, 1996.
[34] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
[35] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.
[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

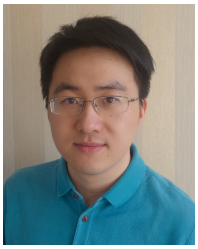
Zhaoyuan Wang is currently a researcher of Tencent Map. Before joining Tencent, he received BS, MS, and PHD degrees from the School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu, China in 2013, 2016, and 2020 respectively. His research interests include: Spatio-temporal Data Mining, Data Fusion, and Urban Computing

Zheyi Pan is a researcher at JD Intelligent Cities Research. His research interest lies in deep learning and data mining with a special focus on urban computing and spatio-temporal data. He has published several papers in top international conferences and well-recognized journals, including KDD, WWW, CIKM, and IEEE TKDE.

Shun Chen is a computer science Ph.D. candidate in Cloud Computer & Intelligent Technology Lab, Institute of Artificial Intelligence, Southwest Jiaotong University, supervised by Prof. Yu Zheng. His research interests include reinforcement learning and data mining on urban computing and spatio-temporal data.

Shenggong Ji Shenggong Ji received the Ph.D. degrees from the Southwest Jiaotong University, Chengdu, China in 2020. He is currently working at Tencent Inc., Shenzhen, China. His major research interests include data mining, reinforcement learning, and urban computing.

Xiuwen Yi received the Ph.D. degree from Southwest Jiaotong University in 2018. He was a postdoctoral researcher at Tsinghua University from 2019 to 2020, a visiting scholar at Penn State University during 2017-2018, and an research intern at MSRA from 2014 to 2017. He is currently a Data Scientist of JD Intelligent Cities Research. His research interests include: Urban Computing, Data Mining, and Deep Learning. He serves as associate editor of IET Smart Cities journal. He has published over 20+ research papers in refereed conferences and journals.



Junbo Zhang is a Senior Researcher of JD Intelligent Cities Research. He is leading the Urban AI Product Department of JD iCity at JD Technology, as well as AI Lab of JD Intelligent Cities Research. Prior to that, he was a researcher at MSRA. He has published over 50 papers in spatio-temporal data mining and AI, urban computing, deep learning, and federated learning. He serves as an Associate Editor of ACM TIST. He is a senior member of CCF, a member of IEEE and ACM.

Jingyuan Wang received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He is currently a Professor of School of Computer Science and Engineering, Beihang University, Beijing, China. He is also the head of Beihang Interest Group on SmartCity, and Vice Director of the Beijing City Lab. He published more than 30 papers on top journals and conferences, as well as named inventor on several granted US patents. His general area of research is data mining and machine learning, with special interests in smart cities.

Zhiguo Gong is currently a professor in the Department of Computer and Information Science, University of Macau, Macau, China. His research interests include databases, web information retrieval and data mining. He is a senior member of the IEEE.

Tianrui Li received the B.S., M.S., and Ph.D. degrees from Southwest Jiaotong University, Chengdu, China, in 1992, 1995, and 2002, respectively. He was a Post-Doctoral Researcher with Belgian Nuclear Research Centre, Mol, Belgium, from 2005 to 2006, and a Visiting Professor with Hasselt University, Hasselt, Belgium, in 2008; University of Technology, Sydney, Australia, in 2009; and University of Regina, Regina, Canada, in 2014. He is currently a Professor and the Director of the Key Laboratory of Cloud Computing and Intelligent Techniques, Southwest Jiaotong University. He has authored or co-authored over 300 research papers in refereed journals and conferences. His research interests include big data, machine learning, data mining, granular computing, and rough sets.

Yu Zheng is a Vice President of JD.com and Chief Data Scientist at JD Technology. He also leads the Intelligent Cities Business Unit as the president and serves as the managing director of JD Intelligent Cities Research. His research interests include big data analytics, spatio-temporal data mining, machine learning, and artificial intelligence. Zheng is an IEEE Fellow and an ACM Distinguished Scientist. Before joining JD Technology, he was a senior research manager at Microsoft Research. Zheng is also a Chair Professor at Shanghai Jiao Tong University, an Adjunct Professor at Hong Kong University of Science and Technology.