Dynamic Bike Reposition: A Spatio-Temporal Reinforcement Learning Approach

Yexin Li[1], Yu Zheng[2], Qiang Yang[1]
[1] The Hong Kong University of Science and Technology, Hong Kong
[2] Urban Computing Business Unit, JD Finance, Beijing, China
yliby@connect.ust.hk, msyuzheng@outlook.com, qyang@cse.ust.hk

ABSTRACT

Bike-sharing systems are widely deployed in many major cities, while the jammed and empty stations in them lead to severe customer loss. Currently, operators try to constantly reposition bikes among stations when the system is operating. However, how to efficiently reposition to minimize the customer loss in a long period remains unsolved. We propose a spatio-temporal reinforcement learning based bike reposition model to deal with this problem. Firstly, an inter-independent inner-balance clustering algorithm is proposed to cluster stations into groups. Clusters obtained have two properties, i.e. each cluster is inner-balanced and independent from the others. As there are many trikes repositioning in a very large system simultaneously, clustering is necessary to reduce the problem complexity. Secondly, we allocate multiple trikes to each cluster to conduct inner-cluster bike reposition. A spatio-temporal reinforcement learning model is designed for each cluster to learn a reposition policy in it, targeting at minimizing its customer loss in a long period. To learn each model, we design a deep neural network to estimate its optimal long-term value function, from which the optimal policy can be easily inferred. Besides formulating the model in a multi-agent way, we further reduce its training complexity by *two* spatio-temporal pruning rules. Thirdly, we design a system simulator based on two predictors to train and evaluate the reposition model. Experiments on real-world datasets from Citi Bike are conducted to confirm the effectiveness of our model.

CCS CONCEPTS

• **Applied computing** → **Transportation;** • **Information systems** → **Spatio-temporal systems;**

KEYWORDS

Bike-Sharing System, Dynamic Bike Reposition, Reinforcement Learning

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220110

ACM Reference format:

Yexin Li, Yu Zheng, Qiang Yang. 2018. Dynamic Bike Reposition: A Spatio-Temporal Reinforcement Learning Approach. In *Proceedings of KDD'18, August 19-23, 2018, London, United Kingdom*, 10 pages.
DOI: https://doi.org/10.1145/3219819.3220110

1 INTRODUCTION

Bike-sharing systems are widely deployed in many major cities, e.g. New York City, Paris and Beijing, providing a convenient transportation mode to citizens. A user can rent or return a bike at a random station via swiping her membership card, generating a bike usage record. However, as the bike usage in a city is very unbalanced, there are usually empty stations without bikes and congested ones lacking available docks in a system, causing severe customer loss. Currently, the system operators are conducting *dynamic bike reposition* to deal with this problem, i.e. adopt trikes to constantly reposition bikes among stations when the system is operating. However, how to reposition to minimize the customer loss in a long period remains an open problem. Real-time monitoring is not a good solution as it is too late to redistribute the bikes after an unbalance has been observed. Repositioning bikes solely based on the bike usage prediction for the coming period only results in a greedy and myopic policy, which may not be optimal for a long period. We summarize three challenges to solve this problem.

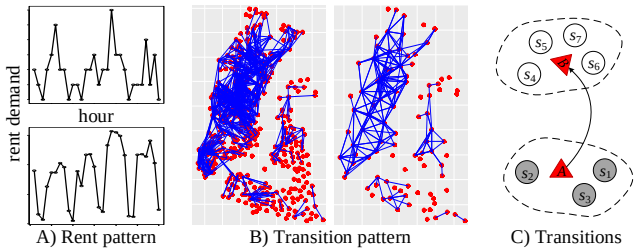
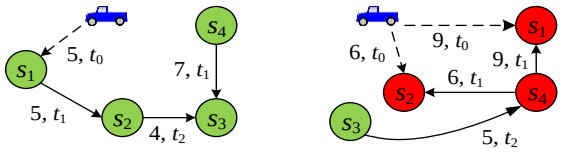


Fig 1. Rent demand and transition pattern in Citi Bike

A bike-sharing system is complex and dynamic. There are usually tens of trikes repositioning among hundreds of stations in a system simultaneously. Repositioning cooperatively in such a large system is complicated, not to say the system is very dynamic when operating. System dynamics is hard to predict for three reasons. 1) Fig 1. A) shows the rent demand at a station in each hour in one month. As we can see, the daily rent pattern fluctuates largely, being impacted by multiple complex factors, e.g. weather, events and correlation between stations. 2) Most transitions seem to be random trips. The first figure in Fig. 1 B) shows the historical commutes, i.e. the transitions averagely happened at least once in the morning on each weekday in Apr.

– Oct. 2016, which only account for 18 percentages. We explain this with an example shown in Fig. 1 C); there are 12 possible inter-station transitions from A to B, among which a customer usually chooses a random one to take based on which stations have available bikes and docks, making one possible frequent transition from A to B into 12 infrequent inter-station ones. 3) The external factors impacting bike usage are highly unbalanced observed, e.g. the sunny hours are much more than the rainy hours. Therefore, separately training a predictor under each condition cannot guarantee the accuracy under the minor ones.

A single bike reposition has long-term effect. Whether a single reposition is good or not cannot be told immediately. We elaborate this with two examples in Fig. 2, where a red circle denotes a station without available docks while a green circle denotes an empty one; a solid arrow labelled with a number and a time denotes how many bikes will be rented from the origin and returned to the destination in that period; a dashed arrow describes how a trike repositions; $t_0 < t_1 < t_2$. Firstly, a single reposition impacts the bike usage in a system for a long period. As shown in Fig. 2 A), if a trike goes to the empty station s_1 to unload 5 bikes there in t_0, the number of available bikes at s_1 becomes 5, thus the 5 coming renters can be served at s_1 in t_1; as these 5 renters ride to s_2 to return their bikes in $t_1 < t_2$, the 4 coming renters who want to rent at s_2 in t_2 can also be served by those returned bikes, and so on so forth. Therefore, how many extra customers can a single reposition serve is hard to estimate. Secondly, the current reposition impacts the following ones. As shown in Fig. 2 B), if a trike goes to station s_1 to pick up 9 bikes in t_0, the number of available docks there becomes 9, thus the 9 coming users can return their bikes there in t_1. However, as s_1 is too far from s_3, after completing picking up there, this trike cannot deliver 5 bikes to the empty station s_3 before t_2 for the 5 coming renters. On the contrary, if the trike goes to s_2 to pick up bikes in t_0, after picking up there, it still has enough time to deliver bikes to s_3 for the 5 coming renters in t_2.



A) Impact bike usage B) Impact next reposition

Fig 2. A reposition action has a long-term effect

Uncertainties in practical reposition. There are uncertain factors in practical reposition process. Although we can predict the system dynamics, we cannot guarantee they are totally the same with the actual observations because of model error and random noise. Besides, the time spent to complete a reposition fluctuates, e.g. delivering 5 bikes from s_1 to s_2 may take 10 minutes today while it took 15 minutes yesterday although they are conducted between the same pair of stations. This may be caused by variable external conditions, e.g. severe weather condition or traffic congestion, and random noise. As dynamic reposition is conducted when the system is operating, time matters, which can also be concluded from the two examples above. These uncertainties, as well as the long-term effect, make optimization models very complicated or even not work.

We propose a spatio-temporal reinforcement learning based dynamic reposition model to tackle these three challenges. Our contributions can be summarized into ***four-fold***.

- We propose a two-step clustering algorithm, named Inter-Independent Inner-Balance algorithm, i.e. IIIB. The algorithm first iteratively clusters individual stations to generate small function regions in a system, ensuring more stable rent demand and transition patterns at each region. Secondly, the algorithm clusters these regions into groups based on the inter-region transitions, guaranteeing that each cluster is inner-balanced and independent from the others. Dividing the entire system into clusters, we largely reduce the problem complexity.
- We generate a system simulator based on two predictors. One is an O-Model to predict the rent demand at each region by a similarity-based KNN method, considering the complex impacted factors and addressing the unbalanced observation issue. The other one is an I-Model to predict the return demand at each region by a transition-based inference method.
- We propose a Spatio-Temporal Reinforcement Learning model, i.e. a STRL, for each cluster to learn an optimal inner-cluster reposition policy. The state of a STRL is carefully designed to capture the system dynamics and real-time uncertainties. As the state and action spaces are very large, we design a deep neural network to estimate the optimal long-term value function for each STRL, from which its optimal reposition policy can be easily inferred. Besides formulating the model in a multi-agent way, we further reduce its training complexity by *two* spatio-temporal pruning rules.
- We conduct experiments on real-world datasets from Citi Bike in Apr. - Oct. 2016, to confirm the effectiveness of our model compared with baselines.

2 OVERVIEW

This section defines the notations and terminologies used thorough this paper and overviews the framework of our model.

Table 1. Notations

Notation	Description
s_i	A station or a region
E_i	An episode
v_i	A trike for reposition
$C_{i,j}$	The j-th cluster in episode E_i
$o_{i,t}$ / $r_{i,t}$	Rent / return demand at s_i in t

2.1 Preliminary

Definition 1 Transition. A transition $f_{ij} = (s_i, s_j, \tau_i, \tau_j)$ is a bike usage record describing that a bike is rented from location s_i at timestamp τ_i and returned to location s_j at timestamp τ_j.

Definition 2 Demand. Rent demand $o_{i,t}$ at a location s_i in period t is the number of customers who want to rent a bike at s_i in t, including the ones succeed or not. The return demand $r_{i,t}$ at location s_i in t has a similar definition.

Definition 3 Episode. An episode E is a long period in a day, in which the total customer loss we want to minimize. Episodes in our problem are carefully defined in 3.1.2 to guarantee some constraints, instead of randomly chosen.

2.2 Framework

As shown in Fig. 3, our model includes an offline learning process and an online reposition process. The learning process has three components, i.e. an IIIB clustering algorithm, system simulator generation and a STRL model for each cluster.

IIIB Clustering Algorithm. To deal with the first issue, i.e. a system is very large and complex, we propose a two-step IIIB clustering algorithm. We firstly cluster stations which are close to each other and have similar transitions to generate small function regions in a system. We then cluster these regions into groups based on their inter-region transition patterns. Multiple trikes are allocated to each cluster to conduct inner-cluster reposition among its regions without inter-cluster bike delivery.

Simulator Generation. To train and evaluate the reposition model, we generate a system simulator based on two predictors, i.e. an O-Model and an I-Model to respectively predict the rent and return demand at each region. To a specific period, e.g. 7:00 – 7:30am on Saturday, we firstly generate a possible weather condition according to the historical weather statistics, e.g. sunny, then O-Model predicts the rent demand at each region in 7:00 – 7:30am on Saturday when it is sunny. Based on these predictions, rent events at each region are simulated by Poisson process. Each time a bike has been rented, I-Model estimates its destination region and arrival time and keeps tracking it. The return events at each region are generated by continually checking whether some bikes arrive there.

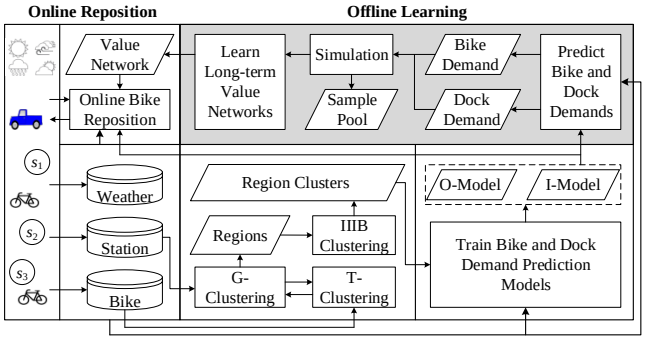


Fig 3. Dynamic bike reposition framework

STRL Model. A STRL model is proposed for each cluster to learn an optimal inner-cluster reposition policy. Our model based on reinforcement learning is formulated in a multi-agent way. Each time a trike completes its last reposition, it continues to conduct a new reposition generated by the policy immediately without waiting for the completion of others. The new reposition is generated based on its current state, which is carefully defined to capture the system dynamics and real-time uncertainties. A state includes multiple factors, e.g. the current bike and dock availability at each region; the real-time predicted rent and return demands; the status of trikes, including itself and the others; the current time, etc. We design a deep neural network to estimate the optimal long-term value function for each STRL, from which its optimal reposition policy can be easily inferred. The network is trained on the system simulator iteratively, which is highlighted with grey in Fig 3.

Online Reposition. After the learning process, we obtain a neural network for each cluster. In online process, when a trike requires for a new reposition, we firstly identify in which cluster it is and generate its current state by O-Model and I-Model. Then the corresponding network is adopted to estimate the optimal long-term value of each possible reposition under this state. The reposition with largest value is selected and returned.

3 METHODOLOGY

3.1 IIIB Clustering Algorithm

3.1.1 Region Generation

As shown by the example in Fig 1. C), the random inter-station transition scenario makes bike reposition among stations less meaningful, as we only need to guarantee that there are available bikes at s_1 or s_2 or s_3 and available docks at s_4 or s_5 or s_6 or s_7. Customers from A to B can choose where to rent and return themselves considering the bike and dock availability at each station. Motivated by this observation, we respectively cluster the several stations around the origin and those around the destination to generate two small regions, i.e. s_1, s_2 and s_3 make up one region while s_4, s_5, s_6 and s_7 make up the other one. Consequently, we only need to guarantee the bike and dock availability at each region. We claim that the rent demand at a region is more stable and regular than that at an individual station; besides, the transition between two regions is more frequent than that between a pair of stations.

To formally formulate this idea, we generate regions in a system based on two constraints. 1) Stations in one region should be close to each other, ensuring the convenience for customers in it. 2) Stations in one region should have similar origin and destination regions, making the inter-region transitions more concentrated and frequent. The methodology to generate these regions is an iterative approach, named bipartite clustering algorithm [2], which alternatively clusters stations based on their locations and transition patterns.

Based on the obtained regions, we analyze the historical bike usage data in Citi Bike to confirm the two advantages claimed above. As shown in the bottom in Fig 1. A), the rent demand at a region is much more stable and regular, thus easier to predict accurately. The random transition issue can also be addressed. The right figure in Fig 1. B) shows the inter-region commutes in the morning on weekday in Apr. – Oct. 2016, which take up 56 percentages. As we can see, the obtained inter-region transition pattern is much simpler, making the return demand prediction easier and more accurate. Regions obtained here can be considered as small function regions in a city, e.g. stations around a resident area are very possible to make up one region while those around an employment area make up another one. Instead of repositioning among regions in the entire system directly, we further cluster these regions into groups and only conduct inner-cluster reposition for two reasons. 1) Clustering can further reduce the problem complexity. 2) A driver usually gets familiar with an area instead of the whole city.

3.1.2 IIIB Clustering Insight

Clusters obtained should have two properties, i.e. inner-balance in each cluster and inter-independence between clusters.

Inner-Balance. The inner-balance property of a cluster C_i in a period t is defined as Eq. 1, meaning that the total bike rent and return demands in the cluster in t should be almost equal.

$$\sum_{s_j \in C_i}(o_{j,t} - r_{j,t}) \approx 0 \quad (1)$$

Therefore, an inner-balanced cluster in period t should include both jammed and starved regions[1]. Otherwise, imagine that a cluster has only starved regions requiring for more bikes to serve the coming bike renters, as there are not jammed regions in this cluster nor inter-cluster bike delivery, no available bikes can be delivered to those starved regions.

Inter-Independence. The inter-independence property of two clusters C_i and C_j in a period t is defined as Eq. 2, meaning that there are not frequent transitions between them in t.

$$|F_{ij,t}| \approx 0 \tag{2}$$

$$F_{ij,t} = \{f_{wv} | s_w \in C_i, s_v \in C_j, \tau_w \in t\} \tag{3}$$

Here, $f_{wv} = (s_w, s_v, \tau_w, \tau_v)$ is a transition defined in section 2.1; $|\cdot|$ denotes the set cardinality.

To reduce the problem complexity, inter-independence between clusters is necessary, thus to generate a reposition policy for each cluster, we only need to consider the regions and trikes in it without the ones outside. Otherwise, the bike and dock availability in a cluster may be largely impacted by the repositions in other clusters, thus impact its own repositions.

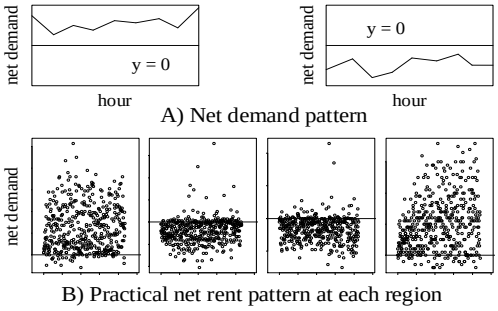


Fig 4. Net demand in the morning rush hours

Episode Selection. As we can see, the inner-balance and inter-independence properties of clusters vary from time to time, leading to different clustering results. We divide the time in the day into *five* episodes as shown in Table 2, in each of which we assume the inner-balance and inter-independence properties of clusters do not change. Consequently, *five* clustering results corresponding to the *five* episodes are obtained.

Table 2. Episodes

Episode	Duration
Morning rush hours	7:00 am – 11:00 am
Day time	12:00 pm – 16:00 pm
Evening rush hours	17:00 pm – 22:00 pm
Travel hours	9:00 am – 17:00 pm
Evening hours	18:00 pm – 23:00 pm

The assumption that the two properties of clusters in each of the above episodes do not change much is reasonable according to prior knowledge and confirmed by the historical bike usage data. 1) The net bike demand $o_{j,t} - r_{j,t}$ at each region in each of the five episodes should have a pattern similar with either one in Fig 4. A), i.e. the net demand at each region should always be positive or negative in an episode, thus a cluster always includes both starved and jammed regions. This claim is reasonable, e.g. in the morning rush hours, the regions close to a resident area are always starved while they are always jammed in the evening rush hours. We analyze the bike usage data in Citi Bike and show the practical net demands at four regions in the morning with Fig 4. B) to confirm our claim. 2) The transition pattern of a region does not change much in an episode neither, e.g. in the morning rush hours, most rented bikes from a resident region head for the employment ones.

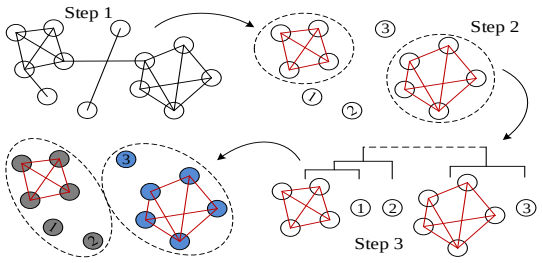


Fig 5. IIIB clusters the regions into 2 groups

3.1.3 IIIB Clustering Methodology

Our region clustering algorithm has *three* steps as shown in Fig 5, where each node denotes a region; the nodes with same color pertain to a same cluster.

1) Construct an inter-region transition graph. To a specific episode, when the transition between two regions in this episode has a frequency larger than a support ρ_1, we add an edge between these two regions. Consequently, an inter-region transition graph for commutes is obtained.

2) Detect community structures. We adopt a betweenness-based community detection algorithm [9] to detect the community structures in the obtained inter-region transition graph. A community, made up by multiple regions, should have dense inner edges while very few inter ones.

3) Cluster communities and regions. Obtaining some communities and the remaining regions that are not in any community, we adopt the agglomerative clustering algorithm [18] to cluster them into groups, based on the similarity defined by Eq. 4 and Eq. 5.

$$sim_{ij} = \frac{\lambda}{G_{ij}} + (|D_{i,t}| + |D_{j,t}| - |D_{i,t} + D_{j,t}|) \tag{4}$$

$$D_{*,t} = (o_{*,t} - r_{*,t}) \tag{5}$$

Here, λ is a tradeoff between the geographical distance and inner-balance increase; $o_{*,t}$ and $r_{*,t}$ stand for the rent and return demand at a region or in a community in t; G_{ij} is the geographical distance between two regions or communities or a region and a community, where the geographical location of a community is the center of the regions in it. Initially, we define each community and each region not in any community to be a cluster. Then, the pair of clusters which has the largest similarity calculated by Eq. 4 is chosen and combined to formulate a new one. Iterate until there are only m_i clusters left, where m_i is the number of clusters in this episode E_i.

Our IIIB clustering algorithm can guarantee the two required properties for three reasons. 1) Each community obtained in the

[1] To formally define, a jammed region has a negative net bike demand and a starved one has a positive net bike demand in a specific period.

second step is almost inner-balanced and inter-independent with others according to the community definition. 2) Agglomerative clustering algorithm in the third step has considered the inner-balance increase when defining the similarity metric, thus guarantees the final clusters to be nearly inner-balanced. 3) Those regions which are not in any community may add inter-cluster transitions, however, as these transitions are minor compared with those in each community, we ignore them.

3.2 System Simulator

In dynamic reposition process, there are interactions between reposition, rent and return. How many bikes are rented and returned at each region determines which regions are jammed or starved, thus impacts how to reposition; bike repositions impact how many available bikes and docks are at each region, thus impact the rent and return. Therefore, to train and evaluate a dynamic reposition model, a system simulator is required to simulate the system dynamics under repositions. Our simulator is based on an O-Model, an I-Model and two assumptions.

O-Model. The rent demand at each region in period t is predicted by a similarity-based KNN method including two steps [2]. Firstly, select the top-k most similar historical periods with t. The similarity is calculated based on the impacted factors, i.e. time and weather. Secondly, calculate the weighted average of the historical bike demands in those similar periods at each region as its predicted demand in t. Here the weights are the similarities. O-Model considers the external impacted factors and can address the observation unbalance issue.

I-Model. The return demand at each region in period t is inferred by tracking the rented bikes via two steps [2]. Firstly, each time a bike is rented, I-Model estimates its destination region and arrival time based on the learned inter-region transition probability and inter-region ride duration distributions; then keeps tracking it. Secondly, to period t, I-Model checks the tracked bikes and selects those that can arrive at their destinations in t to infer the return demand at each region.

Assumption 1. A bike renter arrived at an empty station leaves the system immediately without waiting.

Assumption 2. A customer who wants to return her bike but arrived at a region without available docks chooses the nearest neighborhood region to return.

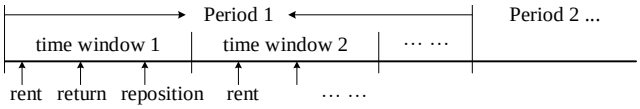


Fig 6. System simulation process in an episode

System simulation process in an episode is shown in Fig. 6. We firstly divide the episode into several periods, i.e. $(\tau_o, \tau_o + \delta_1]$, $(\tau_o + \delta_1, \tau_o + 2 \times \delta_1]$, ..., $(\tau_e - \delta_1, \tau_e]$, where δ_1 is a time length, e.g. 1 hour. A weather condition for each period is generated based on the historical weather statistics. O-Model is then adopted to predict the rent demand at each region in each period under the specific weather conditions, i.e. $O_{j,t}$. We then divide each period into tiny time windows, e.g. $(\tau_o, \tau_o + \delta_2]$, ..., $(\tau_o + \delta_1 - \delta_2, \tau_o + \delta_1]$, where δ_2 is a tiny time length, e.g. 1 minute. The rent demand in each tiny time window in period t is calculated by $O_{j,t} \times \delta_2 \times \frac{1}{\delta_1}$. We repeatedly simulate the rent, return and reposition process in each tiny time window $(\tau, \tau + \delta_2]$ until the episode ends.

1) Rent process. A sequence of rent events at each region is generated via a Poisson process whose parameter is its rent demand in this time window. According to *Assumption 1*, when a renter arrives at a region with available bikes, she succeeds to rent; otherwise, she leaves.

2) Return process. I-Model checks whether any tracked bike can arrive at their destinations in this time window to generate return events at those destinations. According to *Assumption 2*, when a destination region has available docks, I-Model generates a return event there and stops tracking the bike; otherwise, the bike heads for the nearest neighborhood and keeps being tracked.

3) Reposition Process. Check if any trike can arrive at their target regions in this time window. To each of these trikes, we firstly complete its loading there; then we generate a new reposition to it and estimate its completion time by Eq. 6, where G_x is the geographical distance from the current region to the target one; μ_r is a constant speed; t_r is a constant time for loading; ε_r is a reposition noise; we keep tracking it until it completes the new reposition, and so on so forth.

$$t_x \in (\tau + \frac{G_x}{\mu_r} + t_r + \varepsilon_r,\ \tau + \delta_2 + \frac{G_x}{\mu_r} + t_r + \varepsilon_r] \tag{6}$$

3.3 STRL Model

Obtaining a system simulator, whose regions are clustered into groups in each specific episode, a STRL model is proposed for each cluster to learn an inner-cluster bike reposition policy, minimizing its total customer loss in the episode.

3.3.1 Model Insight and Multi-Agent Formulation

As discussed previously, each single reposition has long-term influence and there are uncertain factors in practical reposition process. Therefore, we want a model which not only minimizes the total customer loss in a long period, but also generates repositions online based on the real-time observations instead of generating a sequence of decisions in advance and conducting them one by one. Traditional optimization methods cannot satisfy these requirements while our STRL model does.

Long-term Optimization. A Reinforcement Learning model, i.e. a RL model, which maximizes the long-term reward of a sequence of decisions, fits our problem formulation very well, when we set the reward as the negative customer loss.

Real-time Reposition Generation. Instead of generating a sequence of actions in advance for the trikes to conduct one by one, we generate and assign the next reposition to each trike until it has completed its last one. Therefore, a new reposition is determined based on the real-time observations, e.g. the time when the last reposition is completed, the real-time predicted bike and dock demands at each region, etc., better capturing the uncertainties in practical implementation.

Our STRL model is formulated in a multi-agent way. When a trike completes its last reposition, a new reposition is generated to it immediately based on its current state without waiting for the completion of others. This formulation has two advantages. Firstly, as reposition duration varies from trike to trike largely, always waiting for the others to complete before conducting a new reposition is very inefficient. Secondly, generating an action for each trike one by one can largely reduce the action space

compared with that to generate actions for all trikes at one time. The action space of the former is $O(n_i \times c)$ while that of the latter is $O((n_i \times c)^{k_i})$, where c is the trike capacity; k_i and n_i are respectively the number of trikes and regions in the cluster. Each time a trike requires for a new reposition, the status of the other trikes is contained in its current state. Therefore, trikes in one cluster can still reposition cooperatively instead of working without considering others in our multi-agent formulation.

3.3.2 STRL Model Methodology

To elaborate the methodology of a STRL, we firstly introduce the traditional RL model [15] briefly, then naturally extend to our STRL model. A RL model consists of six components, i.e. $(S, A, T, R, \pi, \gamma)$, where S denotes the state set; A is an action set; T describes the transition probability that an agent took action a_t under state S_t will transit to the next state S_{t+1}, i.e. $S \times A \times S \to T$; R stands for the immediate reward received after taking an action under a specific state and transiting to a next state, i.e. $S \times A \times S \to R$; π is a policy $S \times A \to \pi$, describing the probability to take an action under a specific state; γ is a time discount parameter. At each time t, an agent in state S_t takes an action a_t according to the policy π, then transits to the next state S_{t+1}, receiving an immediate reward r_t. An action has a long-term return defined as Eq. 7 where t_e is the episode end.

$$U_t = r_t + \gamma \times r_{t+1} + \gamma^2 \times r_{t+2} + \cdots + \gamma^{t_e - t} \times r_{t_e} \quad (7)$$

We define the *optimal long-term value function* as Eq. 8, describing the maximum expected return of an action a_t under a specific state S_t by following any policy after t.

$$Q^*(S_t, a_t) = max_\pi \mathbb{E}_\pi[U_t | S_t, a_t, \pi] \quad (8)$$

Obtaining the optimal long-term value of each action under each state, the optimal policy for a RL model can be easily inferred by Eq. 9, i.e. always take the action with maximum optimal long-term value under the current state.

$$a_t^* = argmax_{a_t} Q^*(S_t, a_t) \quad (9)$$

Usually, Bellman equation as Eq. 10 is adopted to estimate the optimal long-term value function via an iterative approach.

$$Q^*(S_t, a_t) = \mathbb{E}_{S_{t+1}}[r_t + \gamma \times max_{a_{t+1}} Q^*(S_{t+1}, a_{t+1}) | S_t, a_t] \quad (10)$$

Our model, formulated in a multi-agent way, is based on the traditionary reinforcement learning theory. The multiple agents, i.e. the trikes in a cluster, share one common reposition policy. Each time a trike completes its last reposition, the policy generates a new one to it immediately based on its current state. Completing this reposition, the trike transits to the next state and receives an immediate reward. The state in our model is carefully designed to capture the system dynamics and real-time uncertainties. We firstly introduce the real-time observation of a trike before stepping to its action, state and reward definitions.

An Observation. Each time a trike requires for a new reposition, it has a real-time observation of the current environment. We define the observation of a trike to include three factors, i.e. the system status, the status of other trikes and its own status. The system status includes the current bike and dock availability at each region and their real-time predicted rent and return demands in the next period. The status of other trikes describes how many bikes they are to pick up or unload at which regions. More information about the other trikes can be considered, e.g. the current number of bikes on each trike, their expected arrival times, etc., although we do not consider them here for simplicity. The status of the trike itself includes its current location and how many bikes it has.

An example of an observation to trike v_1 is shown in Fig 7. A). As we can see, the first large rectangle corresponds to the system status, which has four vectors $b_1, d_1, b_2, d_2 \in R^{n_i}$, where n_i is the number of regions in this cluster; b_1 and d_1 respectively stand for the current bike and dock availability at each region; b_2 and d_2 respectively denote the real-time predicted rent and return demand at each region. The second rectangle describes the status of other trikes, where p_i denotes how many bikes v_i will pick up or unload at which region, e.g. v_2 will pick up 4 bikes at s_1; v_3 will unload 8 bikes at s_{n_i}. We combine the system status and the status of other trikes by $b_1 - b_2 + d_2 - p_2 - p_3 - p_4 = (44, 18, \dots, 37, 35)^T \in R^{n_i}$, to predict the bike availability at each region in the coming period. As the capacity of each region is constant, considering either the bike or the dock availability is enough. The third rectangle corresponding to the status of the current trike, describes that v_1 is at region s_2 with 5 bikes on it. We transform q_1 to a one-hot vector which shows the current location of v_1, i.e. $(0, 1, 0, \dots, 0)$, and a scalar denoting the number of bikes on it. Concatenating the predicted availability vector, the one-hot vector and the scalar, we obtain the current observation for v_1, a $2 \times n_i + 1 -$ dimension vector.

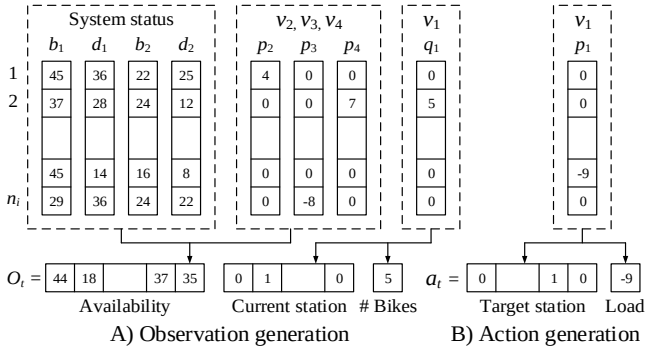


Fig 7. Generate an observation and an action

An Action. An action is defined as a vector describing where a trike should go to pick up or unload how many bikes. An example is shown in Fig 7. B), where the action for trike v_1 is $p_1 = (0, 0, \dots, -9, 0)$, i.e. v_1 unload 9 bikes at region $s_{n_i - 1}$. We transform p_1 to a one-hot vector and a scalar, respectively showing the target region and the number of bikes to load or unload there. Concatenating the one-hot vector and the scalar, we obtain a $n_i + 1 -$ dimension action for v_1.

A State. The state of a trike can be simply defined as its current observation vector extended by the current time. However, as the system dynamics is too complicated to be captured by a single observation, we define the state to be a sequence of interleaved observations and actions combined with the current time, i.e. $S_t = (O_{t-L_1}, a_{t-L_1}, \dots, O_{t-1}, a_{t-1}, O_t, t)$. Therefore, a state is a $(3 \times n_i + 2) \times L_1 + (2 \times n_i + 1) + 1 -$ dimension vector, where L_1 denotes the time lag. An example of a state is shown in Fig. 8, where we set $L_1 = 1$, i.e. $S_t = (O_{t-1}, a_{t-1}, O_t, t)$; A_t denotes the predicted bike availability at each region in t; s_t^c is a one hot vector denoting the current

region of the trike; b_t^c describes how many bikes are on the trike in t; s_t^r and b_t^r make up the action in t, respectively denoting the one-hot vector for the target reposition region and how many bikes to load or unload there.

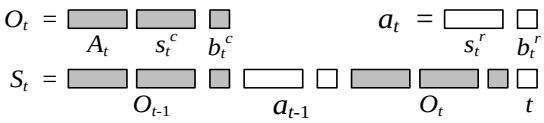


Fig 8. Generate a state S_t

An immediate reward. We set the immediate reward after taking a_t under S_t and transiting to S_{t+1} as the negative customer loss in $(t, t+1]$, thus to maximize the long-term reward can minimize the total customer loss in an episode.

3.3.3 Optimal Long-term Value Network

After formally defining the model, we want to estimate its optimal long-term value function by Eq. 10. As the state and action spaces are large, we design a deep neural network to estimate this function, i.e. $Q^*(S, A, \theta): S \times A \rightarrow Q^*$, where θ are the network parameters.

The input to our deep neural network is a state vector concatenated with an action vector, which contains multi-modal data, thus the network structure needs to be carefully designed instead of a simple fully-connected one. We design the network as Fig. 9, where the state and action shown in Fig. 8 are adopted for illustration, i.e. the grey rectangles make up a state while the orange rectangles make up the action. One-hot vectors in the input, corresponding to regions, are firstly connected to a shared embedding layer. The obtained embeddings are then concatenated with the remaining entries in the input and connected to fully-connected layers (FC). Lastly, the output from FC is combined with the time t to obtain the final long-term value, such that when $t < \rho_2$, the output from FC is returned, otherwise, return zero; here ρ_2 is a parameter which very close to the episode end t_e.

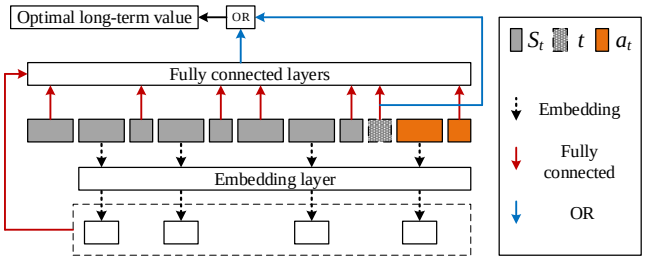


Fig 9. Optimal long-term value network

We train this network via an iterative approach based on Eq. 10 on the simulator. There are *seven* steps [13][14].

1) Random initialize the optimal long-term value network as $Q^*(S, A, \theta^-)$ and set the sample pool to be an empty set.

2) Begin a new episode.

3) When a trike requires for a new reposition, $Q^*(S, A, \theta^-)$ generates an action a_t under its current state S_t, such that $a_t = argmax_{a \in A} Q^*(S_t, a, \theta^-)$. With a probability ε, a_t may be replaced by a random chosen action from A; here $\varepsilon \in [0, 1]$ is a parameter for exploration.

4) The trike takes the reposition a_t and transits to the next state S_{t+1}, receiving an immediate reward r_t. Generate a new sample (S_t, a_t, S_{t+1}, r_t) and add it to the sample pool.

5) Update the network based on a mini batch from the sample pool by Eq. 11. Set $\theta^- = \theta^+$ for next reposition generation.

$$Q^*(S_t, a_t, \theta^+) = r_t + \gamma \times max_{a_{t+1}} Q^*(S_{t+1}, a_{t+1}, \theta^-) \quad (11)$$

6) Jump to step 3 until an episode ends.

7) When the number of episodes been simulated reaches a threshold, terminate the training process and output the obtained neural network; otherwise, jump to step 2.

3.3.4 Spatio-Temporal Pruning Rules

As the large state and action spaces lead to very slow training convergence, we further design *two* spatio-temporal pruning rules to prune some actions under each specific state, thus to improve the training efficiency. Before elaborating those rules, we formally define some region statuses by a parameter ρ_3.

Definition 4 (Predicted) Severely Deficient / Congested. A region is severely deficient when its current bike availability is less than ρ_3. A region is predicted severely deficient when its predicted bike availability is less than ρ_3; here the predicted bike availability can be obtained from A_t in Fig. 8, whose calculation is elaborated in 3.3.2. (Predicted) Severely congested is defined similarly.

Our pruning rules are *two* heuristic ones, to prune the most possible bad actions. 1) Always unload at a predicted severely deficient region and pick up at a predicted severely congested one. 2) The reposition target region should be chosen from the top$-k$ nearest neighborhoods of the current region; besides, how many bikes to load or unload is generated with a step size.

4 EVALUATION

We evaluate our model on real datasets from Citi Bike system and discuss the experiment results in the morning rush hours. Results in the other episodes are similar.

4.1 Data and Baselines

4.1.1 Real Datasets

We adopt two real-world datasets in our experiments, i.e. the Citi Bike data and the weather data in New York City. We only conduct experiments on the principle system, which only covers the urban center as shown in Fig 1. B), as the station status data, which are necessary to estimate the bike and dock demand at each location, is only available to these stations. Data details are summarized in Table 3.

As the potential bike demand is unknown, we estimate the rent demand at each station in each period from the bike usage data, as well as the station status data[2] by Eq. 12 [1], where $g_{i,t}$ is the number of bikes been rented at s_i in t; $L_{i,t}$ is the time length in period t when station s_i is not empty.

$$o_{i,t} = g_{i,t} \times \frac{|t|}{L_{i,t}} \quad (12)$$

[2] We thank Abe Stanway and Chris Heydt for sharing the station status data. https://github.com/astanway/citibike-data https://groups.google.com/forum/#!forum/citibike-hackers

Table 3. Citi Bike data and weather data

Bike Data		Weather Data	
# Stations	389	# Drizzle	220
# Regions	98	# Rainy	75
# Clusters	4	# Foggy	328
# Records	9,846,248	# Sunny	4,513

4.1.2 Baselines and Metric

No Reposition. We run the simulator without any bike reposition.

Greedy Reposition. Greedy algorithm generates a reposition based on *three* rules. 1) An empty trike always goes to the nearest severely congested region to pick up the most possible bikes, considering the trike capacity and how many bikes are at the target region. 2) A full trike always goes to the nearest severely deficient region to unload the most possible bikes, considering how many bikes it has and the dock availability there. 3) A trike which is neither full or empty goes to the nearest severely unbalanced region, which is either severely congested or deficient, to load or unload the most possible bikes depending on the status of the target region.

Prediction based Random Reposition. This reposition strategy means that a trike always randomly chooses a region, which is predicted severely deficient, to unload some bikes or a region which is predicted severely congested to pick up some bikes.

Prediction based Greedy Reposition. PGR is very similar with GR except that we consider the predicted status of each region when applying the three rules in GR.

Optimization Reposition. A related work [1] proposed an optimization model to conduct bike reposition. However, it solves the *static bike reposition* problem. Besides, the number of trikes in each cluster is constrained to one and the optimization objective is to minimize the reposition distance, therefore, this model does not fit our problem settings. However, we slightly refine our formulation by assuming that each cluster is allocated with one trike with a speed $\mu_r \times k$ and a capacity $c \times k$ to approximate our previous settings, i.e. k trikes with a speed μ_r and a capacity c. We can further refine the optimization objective to be the total customer loss while this leads to a too complex model to solve, where heuristic algorithms are required.

Evaluation Metric. The reposition results obtained by different models are compared by the total customer loss in an episode, including the ones failed to rent and the ones failed to return.

4.2 Evaluation Results

4.2.1 IIIB Clustering Results

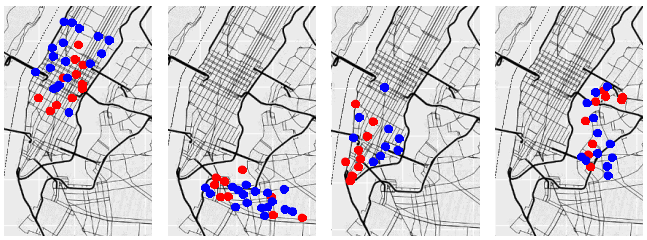


Fig 10. IIIB clustering algorithm results

We firstly generate the function regions in a system and show them in Fig. 10, where each region is denoted by the center of the stations in it. A blue point means a starved region while a red point denotes a jammed one in the morning rush hours. We further conduct IIIB to cluster these regions into *four* clusters. As we can see from Fig. 10, each cluster has both jammed and starved regions in this episode, which is necessary for each cluster to be inner-balanced.

Table 4. Inner-balance property

Cluster	# Regions	Unbalance	Capacity	Unbalance Ratio
C_1	30	-47	5285	-0.009
C_2	27	123	2123	0.058
C_3	18	38	3635	0.01
C_4	23	103	1906	0.054

To further confirm the inner-balance property of each cluster, we calculate their net bike demands in the morning rush hours and show the statistics in Table 4, where *#Regions* mean how many regions are in each cluster; *Unbalance* is the total net bike demand; *Capacity* denotes the total number of docks in each cluster and *Unbalance Ratio* is calculated by *Unbalance* / *Capacity*. As we can see, the unbalance of each cluster is very small, especially compared with the capacity.

Table 5. Inter-independence property

Cluster	C_1	C_2	C_3	C_4
C_1	315,137	0	31,882	0
C_2	0	21,463	0	0
C_3	46,174	0	226,723	0
C_4	670	0	0	41,094

Table 5 shows the statistics for the inner-cluster and inter-cluster commutes. As we can see, there are only a few inter-cluster commutes, i.e. about 10 percentages, which is minor that can be ignored. Therefore, we can consider that the 4 obtained clusters are almost independent from each other, confirming the effectiveness of our clustering algorithm.

4.2.2 STRL Reposition Results

Table 6. Customer loss in the morning rush hours

Customer Loss	NR	PR	GR	PGR	OR	STRL
C_1	267	178	200	157	190	113
C_3	286	229	256	238	237	178
$C_1 + C_3$	553	407	456	395	427	291

Evaluation results in C_1 and C_3 in the morning rush hours under different reposition models are shown in Table 6. We also show the corresponding loss reduction ratio by Fig. 11, which is defined as $\frac{L-L_X}{L}$; here L is the total customer loss without reposition and L_X denotes the customer loss under a specific reposition strategy. To C_2 and C_4, these two areas in Citi Bike system are not as busy as the other two. According to our simulation results, the customer loss in these two clusters is even not larger than fifty. Therefore, a simple heuristic algorithm, e.g. the baseline PGR introduced previously, is good enough for them

to conduct bike reposition. In the following discussions, we do not consider C_2 nor C_4 but only C_1 and C_3.

As we can see from Table 6 and Fig. 11, the customer loss is large when there is no bike reposition while a good reposition policy can reduce the customer loss effectively. GR performs the worst, even than PR reposition. This is reasonable as GR does not consider the predicted bike and dock demands while PR does. Besides, the PR algorithm here is not totally random but with some heuristic constraints, e.g. never pick up at a region which is not predicted severely congested. PGR can easily outperform GR because it considers the predicted bike and dock demands at each region while its other rules are totally the same with GR, making the reposition more hyperopic. However, as we can see, PGR and PR do not perform stably, i.e. PGR beats PR in C_1 while it performs worse than PR in C_3. As both PGR and PR are heuristic algorithms without any error bound, we think this is also reasonable. OR performs very poorly as it is designed for a static bike reposition problem. Besides, when we formulate the optimization objective as the total customer loss in a long period, only approximate solutions can be obtained by some heuristic algorithms, making its performance even worse.

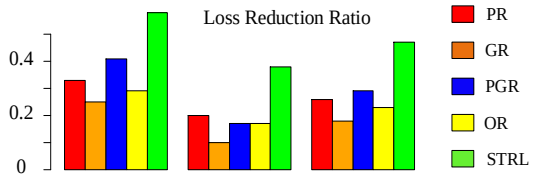


Fig 11. Customer loss reduction ratio

How the customer loss converges when training STRL for C_1 is shown in Fig. 12, where the x-axis denotes training epoch, each of which contains 500 episodes, i.e. simulating the system operation in the morning rush hours for 500 times; the y-axis stands for the average customer loss in each epoch; the exploration rate begins with 1 and keeps reducing to 0.1. As we can see, STRL outperforms the best baseline, i.e. PGR, after about 30 epochs. After iterating for 45 epochs, our model can beat PGR significantly. Our model begins with an initial customer loss around 180, which is not too bad. This is because we have designed two spatio-temporal pruning rules to prune the actions which are very possible to be bad. Therefore, at the very beginning of our training process, the reposition policy is a heuristic algorithm[3] defined by those pruning rules instead of a random one, making the initial customer loss not too large. By iteratively training, the initial heuristic algorithm keeps being continually improved.

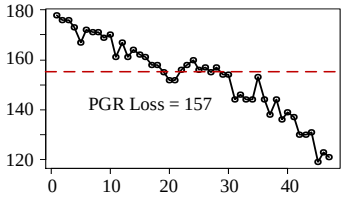


Fig 12. Customer loss in each epoch with pruning

[3] This initial heuristic algorithm is the same with the prediction based random reposition strategy.

To confirm that the pruning rules are very important in model training, we also try to train a model without the rules and obtain a result shown in Fig. 13. As we can see, there is almost no reduction to the customer loss with 45 epochs, not even to say convergence. This is because the number of actions under each specific state is large, thus the iteration times of 45 epochs is far from enough.

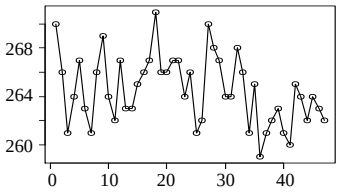
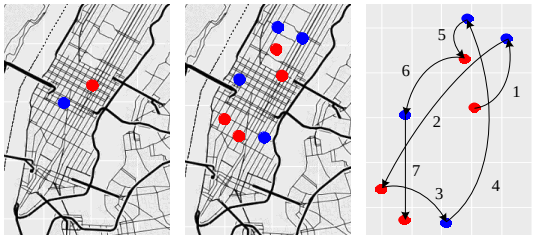


Fig 13. Customer loss in each epoch without pruning

***Case study* 1**. Fig. 14 A) shows a jammed and a starved region in C_1, which we denote as s_1 and s_2 respectively. During 8:30 – 9:00am on a day, the policy generates repositions for trikes to deliver bikes from s_1 to s_2 repeatedly. As s_2 is a starved region, delivering bikes to it is reasonable. However, as we can see from Fig. 10, there are other jammed regions in C_1, which are much closer to s_2 than s_1, but our policy does not choose to pick up bikes from them. According to data analysis, we think this is because the jammed condition at s_1 is very severe, i.e. its average net dock demand is even larger than 300 in the morning. Therefore, mitigating the congestion at s_1 is pressing. Actually, s_1 is frequently chosen as the target region to pick up bikes, not only for supplying s_2 in this 30 minutes, but also other starved regions in the 5 morning rush hours.



A) Case study 1 B) Case study 2

Fig 14. Case studies

***Case study* 2**. Fig. 14 B) shows a sequence of repositions by a trike in 9:00 – 11:00am on a day, where the left figure shows the locations to load and unload on a map; the right figure shows the reposition sequence. As we can see, the trike alternatively loads at jammed regions and unloads at starved regions. However, it usually does not choose the nearest region as the target. This confirms that reposition in a greedy way is usually myopic and may not be good in a long period.

5 RELATED WORK

Studies in bike-sharing systems can usually be categorized into *two* groups, i.e. prediction and system operation.

Jon Froehlich et al. [4] experimentally compared four simple demand prediction models. Kaltenbrunner et al. [5] worked on Bicing system to predict the bike availability at each station. The hourly rent in the entire system in Lyon is predicted by Borgnat et al. [6] via a combination model, i.e. a non-stationary

amplitude for a given day added with a fluctuation in a specific hour. Vogel et al. [7] predicted the rent demand at each station, adopting a time series analysis method similar with [6], but also considering the impact from weather. Yoon et al. [8] presented a personal journey advisor application, considering the spatial interaction between stations and some temporal impacted factors, to predict the bike and dock availability at each station. The authors evaluated their model on a system in Dublin. Li et al. [2] proposed a hierarchical prediction model, considering multiple external impacted factors, e.g. time, meteorology, correlation between stations, etc., to predict the bike and dock demands at each location more accurately. Their work is further extended by Liu et al. [1] and Yang et al. [3] later.

System operation works mainly focus on bike reposition, including static bike reposition, dynamic reposition, and user-based bike reposition, among which the first two are conducted by the system operators while the third one is conducted by the users. Static bike reposition means that the operators redistribute bikes in a system when it does not operate or in the night. Works to solve this problem are mainly based on optimization models [1], to optimize some objectives, e.g. minimizing the total travel distance. Optimization models are also adopted to deal with dynamic bike reposition problem [10][11][12], however, the models are usually too complicated to solve and cannot deal with the uncertainties in the practical operation process. User-based reposition means incentivizing the customers to rent or return a bike at specific stations with a reward [16][17][19]. However, how much reward to offer is a challenging problem.

Recently, a new bike-sharing mode appears in many Chinese cities, i.e. the station-less bike systems which can collect bike trajectory data. Bao et al. [20] adopted these data to conduct some interesting works, e.g. planning bike lanes in a city.

6 CONCLUSIONS

In summary, we propose a spatio-temporal reinforcement learning based dynamic bike reposition model. We firstly propose an IIIB clustering algorithm to divide the entire system into several clusters, largely reducing the problem complexity. Secondly, a STRL model is proposed to learn an optimal inner-cluster reposition policy for each cluster, targeting at minimizing its total customer loss in a long episode. Each STRL model is trained and evaluated on a system simulator, which is designed based on two predictors. We conduct experiment on real-world datasets from Citi Bike system to confirm the effectiveness of our model compared with baselines.

7 ACKNOWLEDGMENTS

We thank the support of National Grant Fundamental Research (973 Program) of China under Project 2014CB340304 and Hong Kong CERG projects 16211214, 16209715 and 16244616.

Yu Zheng was supported by the National Natural Science Foundation of China Grant Nos. 61672399, U1609217.

REFERENCES

[1] J. Liu, L. Sun, W. Chen, H. Xiong. Rebalancing Bike Sharing Systems: A Multi-source Data Smart Optimization. Proc. KDD, pp. 1005-1014, 2016.
[2] Y. Li, Y. Zheng, H. Zhang, L. Chen. Traffic Prediction in a Bike-Sharing System. Proc. SIGSPATIAL, pp. 33:1- 33:10, 2015.
[3] Z. Yang, J. Hu, Y. Shu, P. Cheng, J. Chen, T. Moscibroda. Mobility Modeling and Prediction in Bike-Sharing Systems. Proc. MobiSys, pp. 165- 178, 2016.
[4] J. Froehlich, J. Neumann and N. Oliver. Sensing and predicting the pulse of the city through shared bicycling. Proc. IJCAI, vol. 9, 2009.
[5] A. Kaltenbrunner, R. Meza, J. Grivolla, J. Condian and R. Banchs. Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system. Pervasive and Mobile Computing, pp. 455-466, 2010.
[6] P. Borgnat, P. Abry, P. Flandrin, C. Robardet, J.B. Rouquier and E. Fleury. Shared bicycles in a city: A signal processing and data analysis perspective. Advances in Complex Systems, pp. 415-438, 2011.
[7] P. Vogel and D. Mattfeld. Strategic and operational planning of bike-sharing systems by data mining–a case study. Computational Logistics, pp. 127-141, 2011.
[8] J.W. Yoon, F. Pinelli and F. Calabrese. Cityride: a predictive bike sharing journey advisor. Mobile Data Management, pp. 306-311, 2012.
[9] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E, 2004.
[10] S. Ghosh, Y. Adulyasak, P. Jaillet. Dynamic reposition to reduce lost demand in bike sharing systems. Journal of Artificial Intelligence Research, pp. 387- 430, 2017.
[11] M. Lowalekar, P. Varakantham, S. Ghosh, S. D. Jena, P. Jaillet. Online reposition in bike sharing sytems. Proc. ICAPS, 2017.
[12] S. Ghosh, M. Trick, P. Varakantham. Robust repositioning to counter unpredictable demand in bike sharing systems. Proc. IJCAI, 2016.
[13] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning. Nature, 2015.
[14] H. V. Hasselt, A. Guez, D. Silver. Deep reinforcement learning with double Q-learning. Proc. AAAI, 2016.
[15] R. S. Sutton and A. G. Barto. Reinforcement learning: an introduction. The MIT Press, 1998.
[16] A. Singla, M. Santoni, G. Bartok, P. Mulerji, M. Meenen, A. Krause. Incentivizing users for balance bike sharing system. Proc. AAAI, 2015.
[17] C. Fricker, N. Gast. Incentives and Redistribution in Homogeneous Bike-Sharing Systems with Stations of Finite Capacity. Euro Journal on Transportation and Logistics, 2016.
[18] S. C. Johnson. Hierarchical Clustering Schemes. Psychometrika, 1967.
[19] D. Chemla, F. Meunier, T. Pradeau, R. W. Calvo, H. Yahiaoui. Self-Service Bike Sharing Systems: Simulation, Repositioning, Pricing. 2013.
[20] J. Bao, T. He, S. Ruan, Y. Li, Y. Zheng. Planning Bike Lanes based on Sharing-Bike's Trajectories. Proc. KDD, 2017.

APPENDIX

Parameters used to train the model for the first cluster are summarized in Table 7, including those for both the simulator and the STRL model.

Table 6. Parameters in model training for C_1

Parameters	Values
Trike Capacity c	10
Trike Speed μ_r	200m / min
# Trikes	4
δ_1	1 hour
δ_2	1 min
ρ_2	20 min
ρ_3	24
Exploration rate ε	1 → 0.1
Embedding layer	30
1st FC layer	64
2nd FC layer	32
Loading time t_r	3 min
Reposition Noise ε_r	$N(0,3)$
Step size	5