Spatial-Temporal Demand Forecasting and Competitive Supply via Graph Convolutional Networks

Bolon Zheng\textsuperscript{1}, Qi Hu\textsuperscript{1}, Lingfeng Ming\textsuperscript{1}, Jilin Hu\textsuperscript{2}, Lu Chen\textsuperscript{2}, Kai Zheng\textsuperscript{3}, Christian S. Jensen\textsuperscript{2}
\textsuperscript{1}Huazhong University of Science and Technology, Wuhan, China
\textsuperscript{2}Aalborg University, Aalborg, Denmark
\textsuperscript{3}University of Electronic Science and Technology of China, Chengdu, China
\{bolongzheng, huqi11, lingfengming\}@hust.edu.cn
\{hujilin, luchen, csj\}@cs.aau.dk
\{1,2,3\}

Abstract—We consider a setting with an evolving set of requests for transportation from an origin to a destination before a deadline and a set of agents capable of servicing the requests. In this setting, an assignment authority is to assign agents to requests such that the average idle time of the agents is minimized. An example is the scheduling of taxis (agents) to meet incoming requests for trips while ensuring that the taxis are empty as little as possible. In this paper, we study the problem of spatial-temporal demand forecasting and competitive supply (SOUP). We address the problem in two steps. First, we build a granular model that provides spatial-temporal predictions of requests. Specifically, we propose a Spatial-Temporal Graph Convolutional Sequential Learning (ST-GCSL) algorithm that predicts the service requests across locations and time slots. Second, we provide means of routing agents to request origins while avoiding competition among the agents. In particular, we develop a demand-aware route planning (DROP) algorithm that considers both the spatial-temporal predictions and the supply-demand state. We report on extensive experiments with real-world and synthetic data that offer insight into the performance of the solution and show that it is capable of outperforming the state-of-the-art proposals.

Index Terms—Spatial-temporal request forecasting, graph convolutional networks, route planning

I. INTRODUCTION

The near-ubiquitous deployment of smartphones has enabled transportation network companies such as Didi Chuxing \textsuperscript{[1]}, Uber \textsuperscript{[2]}, and Lyft \textsuperscript{[3]} to operate ride-hailing platforms that enable the servicing of transportation requests by means of fleets of drivers. In this setting, drivers accept requests and move to the origins of requests to complete the requests. Such platforms have reduced significantly the amounts of time drivers are idle and the amounts of time spent waiting for service by prospective passengers, thus improving the traffic efficiency of a city. In this setting, historical requests provide insight into the movement patterns of passengers and drivers, which is beneficial for many applications such as traffic demand prediction, supply and demand scheduling, and route planning.

We study the problem of spatial-temporal demand forecasting and competitive supply (SOUP), which consists of forecasting spatial-temporal service requests, as well as planning routes for agents to active requests in a manner that minimizes the average idle time of all agents. Besides drivers (agents) looking for passengers (requests), this kind of competitive assignment problem also occurs in other urban transportation settings, e.g., drivers looking for parking and drivers looking for electric charging stations.

Our focus is on a population of drivers servicing an evolving set of requests for transportation from an origin to a destination within a given time window. The drivers are often called taxis. Most existing proposals on crowdsourced taxis focus either on how to better match taxis with service requests to maximize global revenue \textsuperscript{[22], [44], [51]}, or how to learn taxi and passenger movement patterns from trajectory data to guide route planning \textsuperscript{[10], [28], [29], [47]}. Once a taxi drops off a passenger and completes a request, no further instructions are provided to the taxi to reduce the time it is idle before servicing the next request. Rather, taxis may either stay stationary or may move towards regions expected high demand, which may lead to competition. We aim to develop a data-driven solution that assigns a route to a taxi as soon as the taxi become idle such that the average time taxis are idle is minimized.

Overall, we address two sub-problems:

(i) **Dynamic request availability patterns.** In order to help agents service new requests quickly, we need to know the request availabilities across the road network of a city. We first choose carefully a spatial granularity for partitioning a road network and temporal granularity for partitioning time in order to achieve accurate predictions of requests. We then build a corresponding model that predicts the availability of future requests.

(ii) **Competition among agents.** If all agents tend to move towards hot regions to find new requests, they will compete if the supply-demand ratio is high, which causes the so-called “herding” effect. To eliminate this effect, we develop a route planning strategy that assigns agents to destinations with supply-demand balance.
The route recommendation framework we propose consists of an offline and an online components. The offline component comprises an end-to-end deep learning model, called spatial-temporal graph convolutional sequential learning (ST-GCSL), that is capable of predicting request availabilities at different locations and times. The online component comprises a demand-aware route planning (DROP) algorithm that exploits both the available spatial-temporal information on requests and the supply-demand state to guide idle agents.

The major contributions are summarized as follows:

1. We design a multi-level partitioning method that enables purposeful request availability prediction across space and time.
2. We propose ST-GCSL to accurately predict the availability of requests.
3. We develop DROP to assign routes that takes into account both the available spatial-temporal request availability information and the supply-demand state.
4. We report on experiments that suggest that the proposed ST-GCSL and DROP outperform their baseline methods.

The rest of the paper is organized as follows. We detail the problem addressed in Section II. In Section III, we present the multi-level partitioning method and the ST-GCSL algorithm. Section IV then presents the DROP algorithm. The papers experimental study is covered in Section V. Related work is the topic of Section VI. Finally, Section VII concludes the paper.

II. PRELIMINARIES

We proceed to introduce the background settings and to formalize the SOUP problem. Frequently used notation is summarized in Table I

| Notation | Definition |
|----------|------------|
| \( G = (V, E) \) | A road network |
| \( A = \{a_i\} \) | A set of mobile agents |
| \( \Omega = \{\omega_j\} \) | A set of requests |
| \( \mathcal{I}_i = \{I_{ik}\} \) | The set of idle times of agent \( a_i \) |
| \( R = \{r_j\} \) | A set of regions on road network |
| \( T = \{t_i\} \) | A set of time slots during a day |
| \( r^* \) | The search route |
| \( \mathcal{G} = (R, A) \) | A region correlation graph |
| \( A \in \mathbb{R}^{N \times |N|} \) | The adjacency matrix of graph |
| \( C_{in}, C_{out} \) | The number of channels for network input and output |
| \( \mathbf{D}_i \) | The predicted request availability vector at time slot \( t_i \) |
| \( \mathbf{E}_t \) | The context feature vector at time slot \( t_i \) |
| \( c_d \) | The final context features |
| \( X_{i,j}, X_{i,j+1} \) | The input and output of \( l \)-th layer |
| \( \mathbf{G}_l \) | The spectral kernel of graph convolution at \( l \)-th layer |
| \( \mathbf{F}_l \) | The temporal convolution kernel at \( l \)-th layer |

Table I: Summary of Notations

| Definition 1 (Road Network). A road network is defined as a weighted directed graph \( G = (V, E, W) \), where \( V \) is the set of nodes, \( E \) is the set of edges, and \( W \) is the set of edge weights. Each edge \( e(u, v) \in E \) that starts from node \( u \) to \( v \) has a positive weight \( w(u, v) \in W \), i.e., travel time on the edge.

Definition 2 (Mobile Agent). We assume a population of mobile agents \( A = \{a_i\} \), which is introduced into the system at once at the beginning. Each \( a_i \) has an original location \( l_o \) on the road network. The population of agents is fixed throughout the operation and has cardinality \(|A|\).

Agents are initially labeled empty and travel along a so-called search route \( r^* \) provided by our system.

Definition 3 (Request). We assume an evolving set of requests \( \Omega = \{\omega_j\} \) that are introduced into the system in a streaming fashion. Each request \( \omega_j = (o, d, t^0, t^\ast) \) has an origin \( o \), a destination \( d \), an introduction time \( t^0 \), and a maximum life time (MLT) \( t^\ast \). A request that is not serviced by \( t^\ast \) time units after \( t^0 \) is automatically removed from the system, an outcome that we call request expiration.

When a request enters the system, the agent that meets the following conditions is assigned to the request by the assignment authority:

(i) The agent is empty.
(ii) The agent is the agent that is closest to the request.
(iii) The shortest-travel-time from the agent to the request enables the agent to reach the request before it expires.

Once an agent is assigned to a request, the agent is labeled as occupied, and the request is removed from the system. Then the agent moves to the request (for pick-up) and then to the request destination (for drop-off), both along the shortest-travel-time path in the road network. Once the agent arrives at the destination, it is labeled as empty and travels along an assigned search route. If the agent finishes traversing its search route without having been assigned a new request, it is assigned a new search route.

If no agent meets the above conditions, the request remains in the system until an agent meets the conditions or the request expires.

It is worth noting that an agent knows neither when and where requests will appear nor has any information about other agents.

Definition 4 (Idle Time). The idle time of an agent is the amount of time from when the agent is labeled as empty to when it is assigned to a request. An agent may experience multiple idle times in a day, each corresponding to traveling on a search route. We denote \( I_{ik} = \{I_{ik}\} \) as the set of idle times of agent \( a_i \)’s search routes, where \( I_{ik} \) is the idle time of \( a_i \)’s \( k \)-th search route \( r^*_{ik} \).

In order to reduce the idle time when planning a search route for an idle agent, we build an accurate request data model that the assignment authority can use to make decisions. The data model is a software module that is shared with all agents.

A. SETTINGS

The problem setting encompasses four types of entities: a road network, mobile agents (taxis), requests (passengers), and an assignment authority.
and that is used to represent request availability patterns and predict future request availability.

B. Problem Statement

The SOUP problem consists of two tasks: (1) spatial-temporal request forecasting; and (2) competitive spatial-temporal request search.

Task 1 (Spatial-Temporal Request Forecasting). Given a road network and a historical set of requests $\Omega$, we aim to build a request data model to predict requests for different locations and times.

Task 2 (Competitive Spatial-Temporal Request Search). Given a road network, a request data model, a set of agents $A$ with original locations, and a stream of requests $\Omega$, we aim to plan a search route for each agent to travel when it becomes idle, such that the average idle time in Eq. 1 is minimized.

$$\frac{1}{\sum_{a_i \in A} |I_i|} \sum_{a_i \in A} \sum_{I_{ik} \in I_i} I_{ik}$$

C. Framework Overview

To solve the above problem, we design a search route recommendation framework. Instead of considering the two tasks as independent modules, we integrate them into a unified framework, shown in Fig. 1. The processing pipeline includes offline and online components. For the former, we propose the ST-GCSL algorithm to train a request data model based on historical request data, i.e., taxi order data. The data model builds a multi-level partitioning on road network and predicts future requests for the partitions. For the latter component, we propose the DROP algorithm that computes a search route for an agent based on the location and time when it becomes idle and the supply-demand state in the near future.

Once an agent is assigned to a request, it travels to the request for pickup and travels to the request’s destination for dropoff. Afterwards, the assignment authority provides the agent with a search route. We use an open source framework called COMpetitive SEarching Testbed (COMSET) as a simulator to implement the whole process. We limit our discussion to the forecasting and search processes, and we omit the details on the assignment of agents to requests.

III. SPATIAL-TEMPORAL REQUEST FORECASTING

In order to solve Task 1, we design an end-to-end deep learning model, called ST-GCSL, to build the request data model for predicting where and when the requests are likely to appear. First, we develop a multi-level partitioning method that partitions the space into a hierarchical structure. Then, we present the details of proposed ST-GCSL.

A. Multi-level Partitioning

We partition the space into a three-level structure. For the first level, we partition the road network into regions based on the administrative boundaries. Fig. 2(a) shows the first-level partition of Manhattan, where the boundary information can be found in [41]. The intuition is that the functionalities differ in these administrative regions, and the travel patterns of people in the same region are usually similar. For instance, the downtown districts are financial and business centers, and the upper district is a traditional wealthy district. With the first-level partition, our method can quickly filter out the most unpopular regions, thus reducing the search space.

For the second level, we adopt the partitioning method in STP [21] that obtains $K$ subregions of a first-level region by applying KMeans algorithm [31]. We find $K$ cluster centers of all the request locations in historical data, and classify the vertices into $K$ groups according to their closest centers. Therefore, we form the regions in the second-level partition, denoted as $R = \{r_1, r_2, \ldots, r_K\}$. Fig. 2(b) shows an example of a second-level partition of Manhattan. Through the second-level partition, we can further refine the popularity of regions in the first-level partition and find relatively more popular regions.

For the third level, we partition the edges (roads) in the second-level regions to different groups based on their starting vertices. Therefore, each third-level partition contains a vertex and several edges, and the vertex can be used as the destination of a search route after determining the third-level partition.

1https://github.com/Chessnl/COMSET-GISCUP
B. Region Correlation Graph

Before we introduce the details of ST-GCSL, we first explain how to build a region correlation graph based on the historical request data.

Let \( T = \{ t_1, t_2, \ldots, t_{|T|} \} \) be the set of time slots, where the length of each time slot is 10 mins. Let \( D_{i,j} \) denote the request availability, i.e., the number of requests (taxi demands) that appear in region \( r_i \) at time slot \( t_j \), we have

\[
D_{i,j} = |\{ \omega \mid \omega.o \in r_i \land \omega.t^o \in t_j \}|. \tag{2}
\]

After processing the historical request data, we obtain a request availability sequence \( \{D_1, D_2, \ldots, D_{|T|}\} \), where \( D_j \in \mathbb{R}^{K} \) represents the request availabilities of all the \( K \) regions at time slot \( t_j \).

We follow previous studies [5], [39] to derive a region correlation graph \( \mathcal{G} = (R, A) \), where \( R \) is the aforementioned second-level region set, and \( A \) is the adjacency matrix of \( \mathcal{G} \). Specifically, we consider two type of neighbors between the regions, i.e., Geographical Neighbors and Semantic Neighbors, based on whether two regions are geographically close or have similar request availability patterns.

- The geographical neighbor is based on the first law of geography [37], “near things are more related than distant things”, that used to extract spatial correlations between a region and its adjacent regions. As shown in Fig. 3(a), \( r_5 \)'s geographical neighbors are \( r_1, r_4, r_6 \) and \( r_9 \).
- The semantic neighbor is used to extract semantic correlations between regions with similar request availability patterns. To quantify the request availability similarity between regions, we use the Pearson Correlation Coefficient [5]. Let \( D_t \) represent the request availability sequence of region \( r_i \) in the training data. The semantic similarity between \( r_i \) and \( r_j \) is defined as follows:

\[
\text{Sim}(r_i, r_j) = \text{Pearson}(D_i, D_j). \tag{3}
\]

We consider regions \( r_i \) and \( r_j \) are semantic neighbors if \( \text{Sim}(r_i, r_j) > \epsilon \), where \( \epsilon \) is a threshold to control the number of semantic neighbors. As shown in Fig. 3(b), the request availability patterns between region \( r_5 \) and \( r_{15} \) are similar, so \( r_5 \) and \( r_{15} \) are semantic neighbors.

Therefore, the adjacency matrix \( A \) is calculated as follows:

\[
A_{ij} = \begin{cases} 
1, & \text{if } r_i, r_j \text{ are neighbors.} \\
0, & \text{otherwise.} 
\end{cases} \tag{4}
\]

C. ST-GCSL Model

The architecture of ST-GCSL is shown in Fig. 4, which consists of two parallel chains to process the historical request availability sequences and contexts, respectively. The corresponding results are then concatenated and convoluted to return the forecasting result, \( \hat{\mathbf{D}}_{t+1} \in \mathbb{R}^{N} \). In the upper chain, the historical request availability sequences, \( \{D_{t-h+1}, D_{t-h+2}, \ldots, D_t\} \), are taken as inputs, which are then followed by two Spatial-Temporal Gate (ST-Gate) Blocks, each of which is composed of Multiple Spatial-Temporal Convolutional Module (MSTCM) and Spatial-Temporal Convolutional Module (STCM) sequentially. Meanwhile, the lower chain takes input as the context features, \( \{E_{t-h+1}, E_{t-h+2}, \ldots, E_t\} \), which is then processed by Toeplitz Inverse Covariance-Based Clustering (TICC) [16] and 2D convolution operation (Conv2d).

1) Spatial-Temporal Gate Block: ST-Gate Block is constructed to process graph-structured historical request availability sequences, which aims to capture the short and long-term spatial-temporal correlation simultaneously. The short-term spatial-temporal correlation is captured via MSTCM which is stacked by several STCMs, where each STCM takes input as short-term historical request availability sequences, i.e., \( m \) time steps, to extract spatial-temporal correlation. On the other hand, the long-term spatial-temporal correlation is maintained by a single STCM that takes input as the whole historical request availability sequences, e.g., \( h \) time steps.

Spatial-Temporal Convolutional Module. Fig. 5(a) depicts the construction of STCM, which consists of three operations: temporal convolution, Graph Convolution (GC), and Gated Linear Unit (GLU) [12]. Given an input \( X_t \in \mathbb{R}^{q \times N \times C_t} \), where \( q \) is the number of time steps and \( C_t \) is the number of input features, and the underlying region correlation graph \( \mathcal{G} \), it is first processed by temporal convolution that is defined as follows,

\[
\Gamma^t \ast X_t = B, \tag{5}
\]
the operation of each STCM can be formulated as follows,

\[ y_{i+1} = \sigma \left[ \sum_{j=1}^{m} \Theta_{i,j} \cdot x_{i+1} \right], \]

where \( \sigma \) denotes the sigmoid function, \( \otimes \) denotes the hadamard product. The right half controls what information can be output. On the left half, a residual connection is utilized to avoid the network degradation. Finally, we have the output \( X_{i+1} \in \mathbb{R}^{N} \).

**Multiple Spatial-Temporal Convolutional Module.** Fig. 5(b) shows the architecture of MSTCM, which is stacked by several STCMs along the time axis to capture the short-term spatial-temporal correlation. Each STCM takes input as consecutive \( m \) time steps, \( Y_i \in \mathbb{R}^{m \times N \times C^l} \), where \( C^l \) is the feature dimension, \( 1 \leq i \leq h - m + 1 \). According to Eq. 5, 6 and 7, the operation of each STCM can be formulated as follows,

\[ y_i = \sigma \left[ \sum_{j=1}^{m} \Theta_{i,j} \cdot x_{i+1} \right], \]

where \( \sigma \) denotes the sigmoid function, \( \otimes \) denotes the hadamard product. The right half controls what information can be output. On the left half, a residual connection is utilized to avoid the network degradation. Finally, we have the output \( X_{i+1} \in \mathbb{R}^{N} \).

To improve the processing efficiency, these STCMs can be executed parallelized, such that MSTCM can capture the short-term spatial-temporal correlation of the entire input at one time.

2) Clustering Context Feature Sequence: Due to the strong correlation between the request availability pattern and the data periodicity, e.g., working days, weekends, or rainy days, we cluster these context features. When predicting the request availability, we first determine which cluster the current context features belong to, and then add this cluster as an additional feature to improve the prediction, as is shown in Fig. 4.

The context feature \( E_{k+i} \) at time step \( i \) is a five-tuple, (time of day, day of week, weather, holiday, events), which is then clustered via TICC. Thus, the context features of all \( h \) time steps can be encoded into a cluster feature vector \( c \in \mathbb{R}^h \). Then, \( c \) is further processed by a convolution operation with a total of \( h - 4(k-1) \) filters, and the kernel size is \( h \times 1 \), such that the output is \( \hat{d}_{cd} \in \mathbb{R}^{[h-4(k-1)] \times 1} \). To enable to concatenate with the output from ST-Gate Block, \( \hat{d}_{cd} \) is duplicated into a tensor \( F \in \mathbb{R}^{[h-4(k-1)] \times N \times 1} \).

Finally, the outputs from historical request availability sequences and contexts are concatenated, and then convoluted to obtain the prediction of request availability in the next time step, \( D_{i+1} \in \mathbb{R}^N \).

**IV. DEMAND-AWARE ROUTE PLANNING**

We proceed to present the details of the demand-aware route planning (DROP) algorithm. In the offline component, the request availability patterns of different regions and time slots are predicted by ST-GCSL. In the online component, we assign the search routes to idle agents based on the forecasting results. Specifically, we first determine the current supply-demand state based on real-time request forecasting result, and compute the region weights according to the state. Then, we adopt a distance-aware random roulette wheel selection method to select a target region level by level. Finally, after determining the final destination, we take the shortest-travel-time path as the search route and send to the agent.
A. Supply-Demand Analysis

For better understanding, we add a visualization module into COMSET simulator to analyze the real-time state of requests and idle agents. The road network is from Manhattan, New York City, USA. The New York TLC Trip Record YELLOW Data\(^2\) on June 1st, 2016 is used as simulation data, and we select three moments of the day for visual analysis. The results are shown in Fig. 6 where the red dots represent idle agents and the blue dots represent active requests. Figs. 6(a), 6(b), and 6(c) show three distinct supply-demand states, i.e., oversupply, supply-demand balance and undersupply.

In Fig. 6(a) we observe that at 12:00, there are almost no active requests, while agents are mainly gathered in the midtown area. In this case, we need to spread agents across the network as much as possible to avoid herding effect such that the number of request expirations can be reduced. Fig. 6(b) shows that before the evening rush hour, the numbers of idle agents and active requests are small since most requests and agents are assigned to each other. The idle agents are gathered in uptown area since it is the destination of most people after work. The agent distribution at this time is reasonable since agents move to where they are requested, thus we aim to find a way to maintain this balance state. Fig. 6(c) shows that the number of requests is much greater than the number of idle agents at 21:00, and the request density in the midtown area is significantly higher than that in other areas. Therefore, to make an improvement, agents need to be guided to the midtown area, rather being scattered across the road network.

B. Supply-Demand State Determination

For the aforementioned three supply-demand states, we adopt different strategies based on real-time supply-demand states to reduce the search time. Therefore, we first need to determine the current supply-demand state. According to [25], we use $\Delta$ as the time horizon that represents the length of near future to be considered. Let $t$ denote the current time, $|A|$ be the number of agents and $T$ be the average time for an agent to finish an order. Through analyzing the data, we find that $T$ does not change too much everyday, which is around 20 minutes, so we take it as a constant. We use $S$ to represent the agent supply, which is estimated as follows:

$$S = \frac{|A|}{T}$$  \hspace{1cm} (9)

According to Eq. (9), $S$ can be regarded as the maximum number of requests that can be served per minute. With ST-GCSL, we can predict the request availability (demand) from $t$ to $t + \Delta$. Let $D$ be predicted average request availability per minutes during this time period, we can determine the supply-demand state by the supply-demand ratio as follows,

$$\begin{align*}
\frac{S}{D} > \alpha_1, & \quad \text{oversupply} \\
\frac{S}{D} < \alpha_2, & \quad \text{undersupply} \\
\text{otherwise}, & \quad \text{supply-demand balance}
\end{align*}$$  \hspace{1cm} (10)

where $\alpha_1 (\alpha_1 > 1)$, $\alpha_2 (\alpha_2 < 1)$ are the parameters to control the threshold of different supply-demand states. The values of $\alpha_1$ and $\alpha_2$ are tuned in the experiments, where $\alpha_1 = 1.2$ and $\alpha_2 = 0.8$ in New York dataset.

C. Region Weight Matrix

In order to choose a region from $R$ as destination, we need to define a weight for each region to compute its popularity. Therefore, we define a region weight matrix as follows.

**Definition 5 (Region Weight Matrix).** For region weight, following the definition in [27], we let $p_{ij}$ and $d_{ij}$ denote the number of pick-up and drop-off events predicted to occur in the region $r_i$ during the time slot $t_j$, respectively. Note that, we build prediction models for both pick-up and drop-off events. We define a region weight matrix $W$ shaped $|R| \times |T|$ and each $W_{ij}$ can be calculated as follows,

$$W_{ij} = p_{ij} - \lambda \times d_{ij} \quad (\lambda < 1).$$  \hspace{1cm} (11)

Obviously, the more pick-up events happen in a region, the more popular the region is. On the contrary, the more drop-off events happen in a region, the more agents competing in the region, thus making the region relatively unpopular. Therefore, we use a parameter $\lambda$ to balance the positive effect of pick-up events and negative effect of drop-off events.

Through above analysis, it is easy to see that the value of $\lambda$ is state-dependent. The intuition is that when the state is oversupply, the value of $\lambda$ should be large enough to prevent agents from going to regions with many drop-off events, which may cause herding effect. For the supply-demand balance state, the value of $\lambda$ is relatively smaller than that of oversupply. In the undersupply state, we use the smallest value of $\lambda$ since the number of idle agents in each region is small and the herding effect may not happen. The optimal value of $\lambda$ needs to be obtained through experiments.

Besides, for the undersupply state, we divide $W_{ij}$ by the total road travel time in the region. It means that we take the request “density” into consideration, and we lead the agents move to the regions with high request “density”, which can

---

\(^2\)https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
Algorithm 1: planSearchRoute Algorithm

Input: Agent’s current location \( l \) and timestamp \( t \)
Output: The search route \( r^* \)

1. Sample a region \( r^1 \) from the first-level regions using roulette wheel selection;
2. Initialize an empty set \( C \) for candidate regions;
3. while \(|C| < n\) do
   4. Sample a second-level region from \( r^1 \) and add it into \( C \);
5. end
6. for each \( r \in C \) do
   7. Calculate \( \text{Dist}(l, r)^\gamma \) as its weight;
8. end
9. Sample a region \( r^2 \) from \( C \) based on the weights;
10. if \( S-D \text{ state} \text{ is undersupply} \) then
    11. Get the intersection with the largest weight from \( r^2 \) as the destination;
12. else
    13. Use roulette wheel selection to select an intersection as the destination;
14. end
15. Get the shortest path from \( l \) to the destination as \( r^* \);

Further reduce the search time without worrying about the herding effect caused by competition. As for undersupply state, the demands far exceed the agent supply, which means that most agents can be assigned to requests, and the search time for finding requests in dense regions is much smaller than that of sparse regions.

D. Planning Search Route

The process of planning a search route for an agent is equivalent to selecting a target region level by level. It is worth noting that the search strategy is state-dependent, i.e., the search strategies of three supply-demand states are different. Algorithm 1 summarizes the planSearchRoute Algorithm.

1) Selecting First-level Region: In order to dispatch agents according to the weight distribution of regions, we adopt the same strategy as STP \cite{21} to select a first-level region. First, we obtain all the region weights at the current time slot \( t \) in the first-level region weight matrix \( W \). Then we use the random roulette wheel selection \cite{40} to obtain a target region. For a region \( r_m \), the probability of \( r_m \) being selected is

\[
Pr[r_m] = \frac{W_{mj}}{\sum_i W_{ij}} \quad (12)
\]

2) Selecting Second-level Region: We apply two rounds selections. The first round of selection is to find sever al popular regions in the first-level region, and the second selection tends to select a region close to the agent’s location to reduce travelling time.

(i) We select a set of \( n \) candidate regions from all the second-level regions of the selected first-level region by repeating \( n \) times of random roulette wheel selection, and denote it as \( C \).

(ii) For each \( r \in C \), we re-compute its weight by using \( \text{Dist}(a.l, r)^\gamma \), where \( \text{Dist}(a.l, r) \) is the road network distance between the agent’s current location \( a.l \) and a candidate region \( r \in C \), and \( \gamma \) is a preference parameter of agents for neighboring regions, i.e., the smaller the value of \( \gamma \) is, the more likely the agents tend to choose the nearby regions. Then we execute the roulette wheel selection to obtain the destination region.

3) Selecting Third-level Region: As a third-level region consists of a vertex and its edges, selecting a third-level region is actually selecting a destination. In order to allow agents move to hotspots, we consider the current supply-demand state.

- If the current state is undersupply, we calculate the weight of each intersection and simply choose the intersection with the largest weight as destination.
- Otherwise, in order to prevent competition, we still use roulette wheel selection to select the destination.

V. Experimental Study

The ST-GCSL model is implemented in Python3 with Tensorflow, and the DROP algorithm is implemented in Java. The experiments are run on a Windows machine with an Intel 2.8GHz CPU and 16GB memory.

A. Experimental Settings

Dataset Description. We use two real datasets: New York dataset and Haikou dataset\cite{4} and a synthetic dataset generated from the New York dataset, to evaluate our model. Table \ref{tab:dataset} summarizes the statistics of datasets, number of taxi requests, number of edges (roads), and number of nodes (intersections).

- New York: The New York dataset is a subset of the New York TLC Trip Record YELLOW Data with the records whose pick-ups and drop-offs are within the Manhattan area, which contains the taxi order records of yellow taxis. Each record includes the coordinates and times of pick-up and drop-off events. We extract the data from January, 2016 to June 2016 for evaluation. The related weather data is extracted from New York Central Park\cite{4}.
- Haikou: The Haikou dataset contains the taxi order data of Haikou city from May 1 to October 31, 2017, including the coordinates of origins and destinations, as well as the order type, the travel category, and the number of passengers. The related weather data\cite{3} is extracted as context features.

- Synthetic Dataset: We construct a synthetic dataset from New York dataset as follows. First, we take one day’s data and count the number of orders per hour in each first-level partition, and then we divide the number of orders by the number of intersections for each first-level partition.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Dataset} & \textbf{Number of Requests} & \textbf{Number of Edges} & \textbf{Number of Nodes} \\
\hline
New York & 200000 & 1000000 & 500000 \\
Haikou & 50000 & 200000 & 30000 \\
Sythetic & 10000 & 20000 & 5000 \\
\hline
\end{tabular}
\caption{Dataset Statistics}
\end{table}

\begin{thebibliography}{99}
\bibitem{3}https://outreach.didichuxing.com/app-vue/HaiKou?id=999
\bibitem{4}http://www.meteomanz.com/index?l=1&cou=4030&ind=72506
\bibitem{5}https://pan.baidu.com/share/init?surl=gj9rHC6Qe67IGEwx_DRyfw
\end{thebibliography}
partitions, thus we get the average number of orders per hour for each intersection. Finally, for each intersection, we take the intersection as origin, randomly choose one of the intersections 3 to 20 minutes away from the intersection as destination. We take the average number of orders at this intersection as the arrival rate \( \lambda \) of Poisson process to generate the pick-up time of records.

**Methods for Comparison.** For the request forecasting problem, we compare ST-GCSL with the following methods:

- **HA:** Historical average model treats the average number of requests of a region at a time slot as the predicted value.
- **VAR** [17]: Vector Auto-Regression model is useful to analyze multivariate time series data.
- **LSTM** [18]: Long Short-Term Memory Network is a typical time series model for the forecasting problem.
- **DCRNN** [26]: Diffusion Convolutional Recurrent Neural Network models the spatial-temporal dependency by integrating graph convolution into the gate recurrent unit.
- **STGCN** [46]: Spatio-Temporal Graph Convolutional Networks captures temporal dependency and spatial correlations by using 2D convolutional networks and graph convolutional network, respectively.
- **STG2Seq** [6]: Spatial-Temporal Graph to Sequence Model uses multiple gate graph convolution module with two attention mechanisms to capture spatial-temporal correlations.
- **Graph WaveNet** [42]: Graph WaveNet combines graph convolution with dilated casual convolution to capture spatial-temporal dependencies.

For a fair comparison, we use the same loss function in all models, which is defined as follows:

\[
\text{Loss}(\theta) = \left\| D_{t+1} - \hat{D}_{t+1} \right\|^2_2
\]

where \( D_{t+1} \) and \( \hat{D}_{t+1} \) denote the real and the predicted request availability, respectively. We normalized the context features data to \([0, 1]\) via using Max-Min normalization and then assign them into 10 clusters through TICC algorithm, and the request values are preprocessed by Mean-Std normalization.

For the route planning problem, we compare DROP with three baselines:

- **SmartAgent** [25]: SmartAgent uses non-negative matrix factorization (NMF) to model and predict the spatio-temporal distributions of requests, and then chooses destinations using a greedy heuristic.
- **TripBandAgent** [9]: TripBandAgent optimizes the taxis search strategy by using reinforcement learning (RL).
- **STP** [21]: Spatial-Temporal Partitioning divides the search space into regions and compute weights for planning routes.

**Parameter Settings.** We set the length of each time slot \( t \) to 10 minutes, so each day is evenly divided into 144 time slots. The length of the time horizon \( \Delta \) is 40 minutes, i.e., the region weight is determined by the prediction results within the next 4 time slots. The threshold \( c \) that controls the number of semantic neighbors is set to 0.5. The value of \( \alpha_1 \) and \( \alpha_2 \) are 1.2 and 0.8, respectively.

The batch size of random gradient descent is 32, the learning rate is 0.001, the number of filters of 2D convolution and graph convolution in the first ST-Gate block are 32 and 32, while the number of filters in the second ST-Gate block are 32 and 64, respectively. The dropout rate is set to 0.2. The parameter \( h \) is set to 10, while \( k \) and \( m \) are equal to 3.

For New York and synthetic dataset, the number of first-level regions is 4, and the number of second-level regions is 100. For Haikou dataset, the number of first-level regions is 4, and the number of second-level regions is 45. We set the distance attenuation parameter \( \gamma \) as -0.2, -0.5 and -0.6 for the three states of undersupply, supply-demand balance, and oversupply, respectively, in all datasets.

**Evaluation Metrics.** For the request forecasting problem, we use three well-adopted metrics:

- Mean Average Percentage Error (MAPE).
- Mean Absolute Error (MAE).
- Rooted Mean Square Error (RMSE).

For the route planning problem, we use three evaluation metrics:

- The average agent idle time.
- The average request waiting time, which is the period of time from its introduction until its pick-up or expiration.
- The expiration percentage, which is the percentage of expired requests.

For the route planning problem, as the search time differences among the algorithms are small (usually within a few seconds), in order to better compare the performance, we use the improvement percentage over the Random Destination (RD) algorithm [43] as metric. It randomly chooses an intersection in the road network as destination and then uses the shortest path between the current location and the destination as the search route.

**B. Performance Evaluation**

1) **Request Forecasting:** To evaluate the performance of ST-GCSL, we first compare ST-GCSL with baseline models. Then, we study the effect of different components of ST-GCSL. Furthermore, we compare our model with some popular models in multi-step prediction.

Note that, for the data of each month, we use the last 10 days for validation and testing (i.e., 5 days for validation and 5 days for testing) and the rest for training using the same method. The following results are outputted from the data in June 2016 on New York dataset and May 2017 on Haikou dataset.

| Dataset          | # of Taxi Requests | # Edges | # Nodes |
|------------------|--------------------|---------|---------|
| New York & Synthetic | 69,406,526        | 9,542   | 4,360   |
| Haikou           | 12,374,094         | 8,034   | 3,298   |

|                   |                  |         |         |
|-------------------|------------------|---------|---------|
|                  |                  |         |         |
| Method       | HA          | New York MAPE(%) | NEW YORK MAE | NEW YORK RMSE | Haikou MAPE(%) | HAIKOU MAE | HAIKOU RMSE |
|--------------|-------------|------------------|--------------|---------------|----------------|--------|--------|
| LSTM         | 37.60       | 4.757            | 7.03         | 41.40         | 2.689          | 4.143  |
| VAR          | 26.30       | 4.024            | 5.802        | 32.92         | 2.023          | 3.076  |
| DCRNN        | 23.77       | 5.838            | 5.519        | 31.70         | 2.002          | 3.041  |
| STGCN        | 20.94       | 5.756            | 5.557        | 31.87         | 2.019          | 3.084  |
| STG2Seq      | 19.42       | 5.708            | 5.519        | 31.54         | 1.985          | 3.007  |
| Graph WaveNet| 19.79       | 5.725            | 5.532        | 30.88         | 1.995          | 3.062  |
| ST-GCSL      | 20.53       | 5.890            | 5.838        | 33.56         | 2.143          | 3.288  |
| STG2Seq      | 18.667      | 5.648            | 5.458        | 29.87         | 1.958          | 2.986  |

**Comparison with Baseline Models.** Table III shows the test error comparison of different methods for request forecasting. We have the following observations:

(i) The classical methods including HA and VAR have poor performances. The reason is that they are not able to model the non-linear spatial-temporal dependencies.

(ii) In general, the deep learning methods perform better. LSTM only takes the temporal dependencies into consideration, while DCRNN, STGCN, STG2Seq, Graph WaveNet use two modules to model temporal dependencies and spatial correlations respectively. So they have better performance than LSTM. It is worth noting that Graph waveNet performs poorly on Haikou dataset, which may be caused by the data sparsity issue.

(iii) ST-GCSL achieves the best performance regarding all the metrics in both two real-world datasets. Our method takes localized spatial-temporal correlations and long-term temporal dependencies into account and can capture temporal dependencies, spatial correlations and spatial-temporal correlations simultaneously, while other approaches ignore either temporal dependencies or spatial-temporal correlations to some extent.

**Component Analysis.** To further evaluate the effect of different components of our model, we compare the following variants of ST-GCSL, including:

(i) Removing the context features;
(ii) Replacing GLU with Relu activation function;
(iii) Removing the STCM from ST-Gate block;
(iv) Removing the MSTCM from ST-Gate block.

The experimental results on both two datasets are shown in Table IV. We have three observations. First, without context features, the model have poor performance since the context features consist useful information for prediction. Second, the model with GLU has better performance than Relu activation function. This is because the module with GLU has twice the parameter size of Relu, so it can captures more complex spatial-temporal correlations. Besides, the gate in GLU can control the output more useful than Relu. Third, removing STCM or MSTCM from ST-Gate block may ignore some temporal information and spatial-temporal correlations to some extent. Specifically, removing STCM may ignore the long-term temporal dependencies while removing MSTCM may miss localized spatial-temporal correlations. Although removing MSTCM from ST-Gate block has slightly better performances in MAE and RMSE on Haikou dataset, it is normal to obtain a slight change of results due to data sparsity. Therefore, the results verify the superiority of our designed network.

**Multi-step Prediction Comparison.** ST-GCSL is able to conduct multi-step prediction as the same as DRCNN, STGCN and STG2Seq. So we compare ST-GCSL with these methods on the request forecasting in the following 3 time steps. Fig. 7 and Fig. 8 present the experimental results of all metrics on both two datasets. As we can see, ST-GCSL performs better than these methods.

2) **Route Planning:** To evaluate the performance of DROP, we randomly select the data of two days from each month to form the test data set. Then, we compare the performances of all algorithms. Finally, we compare the algorithms by varying the parameters, such as the number of agents, and the request’s maximum life time (MLT).

**Performance Overview.** To compare the performances of all the algorithms with default parameter settings on all the datasets, we demonstrate the idle time (s), waiting time (s), and expiration percentage (%) on all the datasets in Table VII. DROP is better than other algorithms in terms of idle time and expiration percentage on all the datasets. Even in terms of waiting time, it is only slightly worse than STP in the New York.
Table V
Performance Overview

|            | DROP     | SmartAgent | TripBandAgent | STP      | RD       |
|------------|----------|------------|---------------|----------|----------|
| New York   | Idle Time (s) | 452.5072   | 453.4331      | 453.8945 | 452.9509 | 481.0416 |
|            | Waiting Time (s) | 287.8552   | 288.5655      | 289.636  | **286.7099** | 324.1286 |
|            | Expiration (%) | 14.411     | 14.487        | 14.498   | 14.455   | 16.305   |
| New York   | Idle Time (s) | 449.3292   | 451.0788      | 452.608  | 451.3396 | 502.236  |
|            | Waiting Time (s) | 335.1126   | 335.7864      | 336.9288 | **332.1998** | 367.2696 |
|            | Expiration (%) | 16.035     | 16.142        | 16.250   | 16.155   | 19.422   |
| Synthetic  | Idle Time (s) | 554.0121   | 556.3383      | 555.1292 | 555.2257 | 560.2584 |
|            | Waiting Time (s) | 153.957    | 150.6131      | 157.526  | 163.4188 |          |
|            | Expiration (%) | 3.108      | 3.278         | 3.172    | 3.216    | 3.570    |

York dataset and synthetic dataset. Moreover, we find that the improvement of DROP compared to RD is related to the distribution of requests. The spatial and temporal distribution of requests on the New York dataset is quite uneven, so DROP improves more compared to RD, and the spatial and temporal distribution of requests on Haikou dataset is uniform, so the improvement of DROP on the Haikou dataset is smaller than that on the New York dataset. Therefore, the greater difference in the spatial and temporal distribution of the requests, the better performance that DROP has.

**Idle Time.** We evaluate the average idle time for agents on all the datasets by varying agent cardinality and MLT. The results are shown in Fig. 9 and Fig. 10. The results show DROP has a better performance than its competitors under different agent cardinality and MLT on all the datasets. Compared to RD, DROP has the largest percentage improvement in idle time, from about 6.3% to 2.9% on New York dataset, 11.8% to 2.4% on synthetic dataset, and 1.2% to 0.7% on Haikou dataset, as the agent cardinality increases. On all the datasets, DROP has the lowest average idle time no matter how long the MLT is.

**Waiting Time.** Fig. 11 and Fig. 12 show the performances of all algorithms on average request’s waiting time as the agent cardinality and MLT increase. Through the figures, we can see that when the agent cardinality is small, the waiting time of DROP is not the smallest. For example, on the New York dataset, when the number of taxis is 5000 and 6000, STP has the shortest waiting time. However, as the number of taxis increases, DROP is significantly better than other algorithms in waiting time. As we do not take the average waiting time our primary evaluation metric, the performance of DROP is still good.

**Expiration Percentage.** The performances of all algorithms on the expiration percentage are shown in Fig. 13 and Fig. 14. The results show that DROP has the lowest expiration percentage compared to the baseline algorithms in all cases. This is because DROP uses a multilevel partition-based method to dispatch agents such that the distribution of agents is consistent with the real-time distribution of requests, which reduces the expiration percentage.

![Fig. 9. Idle Time Improvement by Varying Agent Cardinality](image1)

![Fig. 10. Idle Time Improvement by Varying MLT](image2)

![Fig. 11. Waiting Time improvement by Varying Agent Cardinality](image3)

![Fig. 12. Waiting Time Improvement by Varying MLT](image4)
VI. RELATED WORK

A. Taxi Demand Prediction

Taxi demand prediction is a critical component to build an efficient transportation system. LinUOTD [38] applies a unified linear regression with high-dimensional features. One study [14] treats it as a classic time series problem by utilizing RNN network. ConvLSTM [32] combines CNN and RNN to model spatial and temporal correlation, which is an extension of fully-connected LSTM [18]. ST-ResNet [50] models the temporal closeness, period, and trend properties of crowd traffic based on [48], [49]. One study [27] conducts a systematic comparison of two recent deep neural networks [24], [48] for taxi demand prediction. DMVST-Net [45] employs graph embedding as an external features to improve forecast accuracy based on localized spatial and temporal views. However, all these CNN-based methods only model the Euclidean correlations among grid regions. While GCN can extract local features from non-Euclidean structures resulting in its increasing popularity. One study [20] predicts the travel cost, while GCWC [11] fills in missing stochastic weights of speed via GCN and their techniques can be applied on demand prediction. GEML [39] formulates origin-destination matrix prediction via GCN. DCRNN [26] models the spatial-temporal dependency by integrating graph convolution into the gate recurrent unit. STGCGN [46] captures temporal dependency and spatial correlations by using 2D convolutional networks and graph convolutional network. STG2Seq [6] uses multiple gate graph convolution modules with two attention mechanisms to capture spatio-temporal correlations, while Graph WaveNet [32] combines graph convolution with dilated casual convolution to capture spatial-temporal dependencies. STMGCN [52] employs multi-graph to extract spatial correlations and uses RNN to capture temporal dependencies. These GCN-based models are more flexible and progressive than CNN-based methods, and our method also uses GCN to extract spatial correlations.

B. Taxi Dispatch

The existing work on taxi dispatch can be divided into two categories, order matching [4], [7], [8], [11], [23], [34], [36], [44], [51], [52] and route recommendation [10], [13], [15], [19], [28], [30], [33], [35], [47]. Order matching aims to match idle taxis with passengers to maximize global revenue. One study [44] considers both instant passenger satisfaction and the expected future gain in a unified decision-making framework, and then optimizes long-term platform efficiency through reinforcement learning. One study [8] formulates the problem of online taxi routing and introduces a backbone algorithm that makes online vehicle routing problem tractable. Two studies [7], [11] introduce the idea of game theory into the matching problem, and model the matching as a process to reach a stable Nash equilibrium. OSM-KIID [52] not only maximizes the expected total profits, but also tries to satisfy the preferences among passengers and taxis.

For route recommendation, some studies [30], [47] learn taxi and passenger movement patterns from trajectory data to develop routing strategies. One study [10] divides the space into several regions, thus converting the infinite search space into a limited region search space, and then establish a fluid model associated with a closed queueing network composed of single and infinite server stations to find an optimal static empty-car routing policy. MDM [13] employs a continuous learning platform where the underlying model that predicts future customer requests is dynamically updated, and minimizes the distance of idle taxi drivers to anticipated customers through Monte Carlo Tree Search. RHC [28] combines highly spatio-temporal correlated demand supply models and real-time GPS location and occupancy information to dispatch taxis, the objectives include reducing taxi idle driving distance and matching spatio-temporal ratio between demand and supply for service quality.

VII. CONCLUSION AND FUTURE WORK

In this paper, we study the spatial-temporal demand forecasting and competitive supply (SOPU) problem. We propose an end-to-end deep learning model ST-GCSL for request forecasting, and develop a route planning algorithm DROP to guide agents to reduce the idle time. The experimental studies show that ST-GCSL has better performance compared with baseline models in both the single and multi-step predictions. The improvements on MAPE, MAE and RMSE are 3.9%, 1.6% and 1.2% over the baseline methods on New York dataset. In addition, DROP outperforms all the competitors on the average idle time with improvements of 6.3%, 11.8% and 1.2% in the three datasets, respectively.

For future work, we plan to further optimize our ST-GCSL for higher accuracy and better robustness. In addition, SOPU can be extended to more complex problems, such as dynamic vehicle routing, where customers can dial a ride in a dynamic time window. We plan to use deep reinforcement learning and
operation research methods to solve these problems in our future work.

REFERENCES

[1] Didi: https://www.xiaojueji.com, 2020.
[2] Lyft: https://www.lyft.com, 2020.
[3] Uber: https://www.uber.com, 2020.
[4] D. Ayala, O. Wolfson, B. DasGupta, J. Lin, and B. Xu. Spatio-temporal matching for urban transportation applications. *ACM Trans. Spatial Algorithms and Systems*, 3(4):11–1:139, 2018.
[5] L. Bai, L. Yao, S. S. Kanhere, X. Wang, W. Liu, and Z. Yang. Spatio-temporal graph convolutional and recurrent networks for citywide passenger demand prediction. In *CIKM*, pages 2293–2296, 2019.
[6] R. Bai, J. Li, J. A. D. Atkin, and G. Kendall. A novel approach to independent taxi scheduling problem based on stable matching. *JORS*, 65(10):1501–1510, 2014.
[7] D. Bertsimas, P. Jaillet, and S. Martin. Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1):143–162, 2019.
[8] F. Borutta, S. Schmoll, and S. Friedl. Optimizing the spatio-temporal resource search problem with reinforcement learning (Gis cup). In *SIGSPATIAL/GIS*, pages 628–631, 2019.
[9] A. Braverman, J. G. Dai, X. Liu, and L. Ying. Empty-car routing in ridesharing systems. *Operations Research*, 67(5):1437–1452, 2019.
[10] P. Cheng, L. Chen, and J. Ye. Cooperation-aware task assignment in on-demand taxi dispatching. *CoRR*, abs/1707.08110, 2017.
[11] Q. Guo and O. Wolfson. Probabilistic spatio-temporal resource search. *GeoInformatica*, 22(1):75–103, 2018.
[12] D. Hallac, S. Vare, S. P. Boyd, and J. Leskovec. Toegression inverse covariance-based clustering of multivariate time series data. In *IJCAI*, pages 5254–5258. ijcai.org, 2018.
[13] J. D. Hamilton. *Time series analysis*, volume 2. Princeton New Jersey, 1994.
[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[15] J. Holler, R. Vuorio, Z. Qin, X. Tang, Y. Jiao, T. Jin, S. Singh, C. Wang, and J. Ye. Deep reinforcement learning for multi-driver vehicle dispatching and repositioning problem. In *ICDM*, pages 1090–1095. IEEE, 2019.
[16] J. Hu, C. Guo, B. Yang, C. S. Jensen, and H. Xiong. Stochastic origin-destination matrix forecasting using dual-stage graph convolutional, recurrent neural networks. In *ICDE*, 2020.
[17] Q. Hu, L. Ming, C. Tong, and B. Zheng. An effective partitioning approach for competitive spatial-temporal searching (Gis cup). In *SIGSPATIALGIS*, pages 620–623, 2019.
[18] J. Jin, M. Zhou, W. Zhang, M. Li, Z. Guo, Z. Qin, Y. Jiao, X. Tang, C. Wang, J. Wang, G. Wu, and J. Ye. Coride: Joint order dispatching and fleet management for multi-scale ride-hailing platforms. In *CIKM*, pages 1983–1992, 2019.
[19] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *SIGSPATIALGIS*, pages 189–198, 2012.
[20] J. Ke, H. Zheng, H. Yang, and X. Chen. Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *CoRR*, abs/1706.06279, 2017.
[21] J. Kim, D. Pfoser, and A. Züfle. Distance-aware competitive spatiotemporal searching using spatiotemporal resource matrix factorization (Gis cup). In *SIGSPATIALGIS*, pages 624–627, 2019.
[22] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR* (Poster), OpenReview.net, 2018.
[23] S. Liao, L. Zhou, X. Di, B. Yuan, and X. Jiong. Large-scale short-term urban taxi demand forecasting using deep learning. In *ASP-DAC*, pages 428–433, 2018.
[24] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for taxi drivers. In *KDD*, pages 45–54, 2014.
[25] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi. Dnn-based prediction of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. In *KDD*, pages 1227–1235, 2019.
[26] Wikipedia contributors. *Fitness proportionate selection* — Wikipedia, the free encyclopedia, 2019. [Online; accessed 18-June-2020].
[27] Wikipedia contributors. List of Manhattan neighborhoods — Wikipedia, the free encyclopedia, 2020. [Online; accessed 11-June-2020].
[28] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *IJCAI*, pages 1907–1913. ijcai.org, 2019.
[29] B. Xu, Y. Pao, and U. Khurana. *Sigsparis gis cup*, 2019.
[30] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, C. Liu, W. Bian, and J. Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *KDD*, pages 905–913, 2018.
[31] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *AAAI*, pages 2588–2595, 2019.
[32] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640. ijcai.org, 2018.
[33] N. J. Yuan, Y. Zheng, L. Zhang, and X. Xie. T-finder: A recommender system for finding passengers and vacant taxis. *TKDE*, 25(10):2390–2403, 2013.
[34] J. Zhang, Y. Zheng, and D. Qi. Deep spatio-temporal residual networks for citywide crowd flows prediction. In *AAAI*, pages 1655–1661, 2017.
[35] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi. Dun-based prediction model for spatio-temporal data. In *SIGSPATIALGIS*, pages 92:1–92:4, 2016.
[36] Z. Zhang, Y. Zheng, D. Qi, R. Li, X. Yi, and T. Li. Predicting citywide crowd flows using deep spatio-temporal residual networks. *Artif. Intell.*, 259:147–166, 2018.
[37] L. Zhang, T. Hu, Y. Min, G. Wu, J. Zhang, P. Feng, P. Gong, and J. Ye. A taxi order dispatch model based on combinatorial optimization. In *KDD*, pages 2151–2159, 2017.