2019

Decision-making for Vehicle Path Planning

Jun Xu
University of Central Florida

DECISION-MAKING FOR VEHICLE PATH PLANNING

by

JUN XU
B.S. Electrical Engineering, Tianjin Chengjian University, 2010
M.S. Electrical Engineering, Beijing University of Posts and Telecommunications, 2014

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2019

Major Professor: Damla Turgut

ABSTRACT

This dissertation presents novel algorithms for vehicle path planning in scenarios where the environment changes. In these dynamic scenarios the path of the vehicle needs to adapt to changes in the real world. In these scenarios, higher performance paths can be achieved if we are able to predict the future state of the world, by learning the way it evolves from historical data. We are relying on recent advances in the field of deep learning and reinforcement learning to learn appropriate world models and path planning behaviors.

There are many different practical applications that map to this model. In this dissertation we propose algorithms for two applications that are very different in domain but share important formal similarities: the scheduling of taxi services in a large city and tracking wild animals with an unmanned aerial vehicle.

The first application models a centralized taxi dispatch center in a big city. It is a multivariate optimization problem for taxi time scheduling and path planning. The first goal here is to balance the taxi service demand and supply ratio in the city. The second goal is to minimize passenger waiting time and taxi idle driving distance. We design different learning models that capture taxi demand and destination distribution patterns from historical taxi data. The predictions are evaluated with real-world taxi trip records. The predicted taxi demand and destination is used to build a taxi dispatch model. The taxi assignment and re-balance is optimized by solving a Mixed Integer Programming (MIP) problem.

The second application concerns animal monitoring using an unmanned aerial vehicle (UAV) to search and track wild animals in a large geographic area. We propose two different path planing approaches for the UAV. The first one is based on the UAV controller solving Markov decision process (MDP). The second algorithms relies on the past recorded animal

appearances. We designed a learning model that captures animal appearance patterns and predicts the distribution of future animal appearances. We compare the proposed path planning approaches with traditional methods and evaluated them in terms of collected value of information (VoI), message delay and percentage of events collected.

To my parents and whomever taught me something!

ACKNOWLEDGMENTS

I gratefully thank my advisor Dr. Turgut who made my study at UCF a memorable experience. I also would like to thank the members of my committee, Dr. Wei Zhang, Dr. Shaojie Zhang, and Dr. Samiul Hasan for their time and valuable comments.

I would also like to thank my labmates, collaborators, and friends for their support, friendship and the discussions about interesting ideas. Special thanks to my wife Ping Yu, for sharing laugh and tear with me during the past five years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

This dissertation presents contributions to the fields of vehicle path planning based on the knowledge acquired from historical data.

Vehicle path planning can be formulated as an optimization problem. The path planning goals vary greatly with different applications. In this dissertation, we present two path planning applications. In the first application we propose a centralized vehicle dispatch model for taxi scheduling and path planning in a large city. The goal is to balance the demand and supply ratio over the city as well as to minimize the required number of taxis, the passengers' waiting time and the drivers' idle driving distance. In the second application we present is about animal monitoring and tracking. Wireless sensor networks (WSNs) and unmanned aerial vehicles (UAVs) are considered for animal detection and data collection. The goal is to make the UAVs collect the sensed information as quickly as possible while not affecting the animal's life.

A key component in vehicle path planning is state prediction. For instance, if a system can predict the density of taxi demand, it would be helpful for the drivers to plan their schedules, even though they are not familiar with different areas of a city. Looking into the future, self-driving vehicles need to autonomously decide where to look for passengers and balance the supply-demand ratios over the city without human assistance [1]. To achieve these goals, robust and efficient prediction models are necessary. In this dissertation, we first propose different learning models to capture knowledge patterns from historical data, and then predict future network states for different vehicle path planning tasks. More specifically, in the taxi dispatch application, taxi scheduling and path planning are based on the taxi demand and destination prediction. In the animal monitoring application, we do path planning for

the UAV by predicting the animal appearance distribution over the whole area.

The remainder of this chapter is organized as follows. We first explain our motivation to propose prediction models for vehicle path planning in Section 1.1. We present a summary of our contributions in this research domain in Section 1.2. Finally, we include the outline of the dissertation in Section 1.3.

1.1 Motivation

Predicting the future state of systems often has a practical importance. For instance, with the prediction of the migration patterns of animals, we can identify the areas that are most critical for wildlife conservation. To make a prediction, one of the most relevant pieces of information is the historical data. Historical data can provide us rich insights about how things varied in the past.

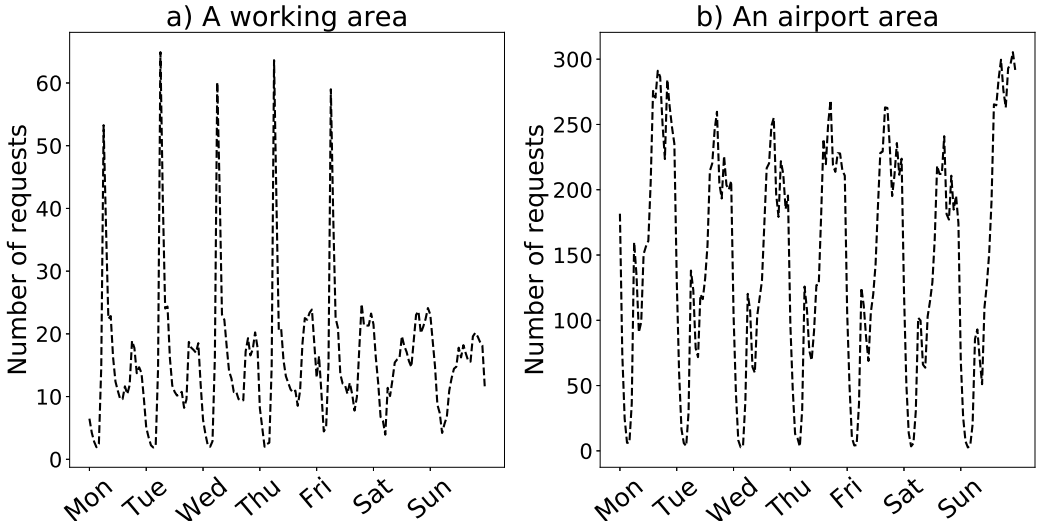

Figure 1.1: Taxi demand pattern in two different areas of New York City.

For instance, if we take a look at the number of taxi requests at two different areas in

Figure 1.1, we observe that in a specific area of the city, the historical taxi demand shows a similar sequential pattern every week. With this repeated demand pattern, we can improve the taxi service performance by pre-dispatching taxis to specific areas over the city.

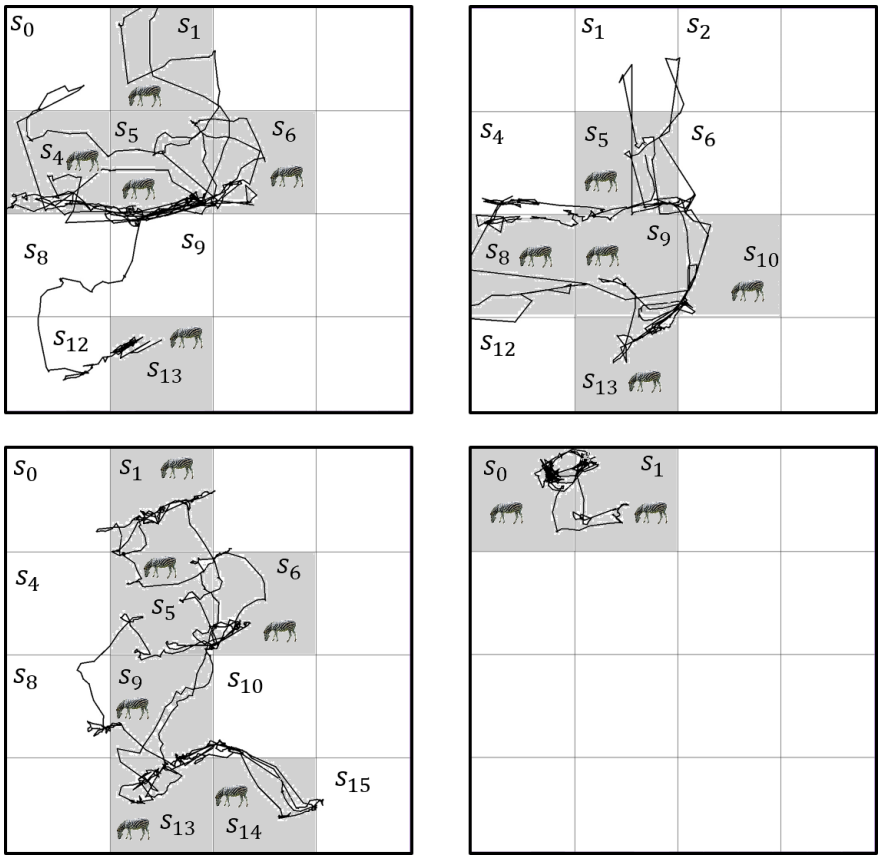

Figure 1.2: Movement trajectories of 4 zebras from the ZebraNet dataset.

We show another example on animal movement trajectories in Figure 1.2. Wild animals have their own movement patterns. They are more likely to stay in a certain location for rest or roam around in a small area. Animals also often re-visit an area multiple times. Therefore, observing animals in a region allow us to infer the possibility of future visits in the same region.

Motivated by these observations, we want to design different learning models to capture the

knowledge patterns from historical data. Then, with the learned patterns, we can predict future network states and do path planning with different goals accordingly.

1.2 Contributions

In this dissertation, we discuss different learning and prediction approaches based on historical data and then do path planning with the predicted results. Our major contributions are as follows:

Taxi demand and destination distribution learning and prediction. We propose a real-time method for predicting taxi demands and destinations in different areas of a city. We divide a large city into smaller areas and aggregate the number of taxi requests (from historical data) in each area during a time period (e.g. 30 minutes). Thus, past taxi data becomes a data sequence of the number of taxi requests in each area.

For the taxi demand prediction, we train a Long Short Term Memory (LSTM) [2] recurrent neural network (RNN) with the sequential data. The network input is the current taxi demand and other relevant information while the output is the demand in the next time step. The relevant information includes *Date & Time, Day of week, Weather and Drop-offs*.

For the destination prediction we found that the destination distribution is multi-modal. This motivates us to use Mixture Density Networks (MDNs) [3], developed by Christopher Bishop, that are designed to model real-valued multi-modal distributions. We design a prediction model with MDNs on top of a feed-ford neural network. The output of this model is the predicted parameters of a mixture of Gaussian kernels. Then we can sample from it to get the predicted destinations.

Taxi dispatch center with predicted demand and destination. Based on the predicted demand and destination distribution, we present a taxi dispatch system in which the predicted results are used for the taxi reallocation towards the future supply-demand balance in the city. The optimal taxi assignment and reallocation strategy is obtained by solving a mixed-integer program (MIP) whose objective is to minimize the total idle driving distances while serving all the coming requests as soon as possible. We validate our dispatch system with taxi trip data from 2016 in New York City.

Animal monitoring with Markov decision process (MDP). We present a path planning model for the UAV on animal monitoring in wildlife areas. In this application, static sensor networks are deployed for animal detection. The UAV visits the clusters of sensor nodes for data collection purpose. We propose different MDP-based path planning approaches for the UAV to maximize the collected value of information (VoI) [4]. The proposed model is evaluated using real-world mobility traces of zebras.

Animal monitoring with appearance prediction based on historical data. We present another path planning model for the UAV on the animal monitoring application. First, we train a neural network that learns the animal appearance patterns from the collected historical data. Second, we use the model to predict the future probability distribution of animal appearance. Finally, we propose a traveling salesman problem (TSP)-based path planning approach for the UAV using the predicted animal appearance distribution (TSP-D).

1.3 Outline

This dissertation is organized as follows.

Chapter 1 presents the problem descriptions and an overview of the proposed approaches.

Chapter 2 conducts a review of the literature related to taxi demand and destination prediction, taxi dispatch center modeling; path planning for mobile sinks in wireless sensor networks, and animal monitoring with sensor networks.

Chapter 3 shows details about the taxi demand and destination distribution prediction models. This chapter includes description of data processing, feature selection and the predicted results evaluation on two different prediction models.

Chapter 4 describes the taxi dispatch model based on the predicted taxi demand and destination from Chapter 3. We present and evaluate the taxi request assignment method and taxi dispatch strategy towards demand-supply balance.

In Chapters 5 and 6, we show two path planning approaches for the UAV on the animal monitoring application. In Chapter 5, we present a Markov decision process (MDP) based path planning approach for the UAV. The outcomes of the MDP approach is analyzed in details and compared with other path planning approaches.

Chapter 6 presents a path planning approach for the UAV based on the animal appearance prediction. The proposed approach is compared with the greedy algorithm and the traveling salesmen problem (TSP)-based path planning heuristics in terms of the collected value of information.

Chapter 7 concludes the dissertation and describes plans for future work.

CHAPTER 2: RELATED WORK

2.1 Taxi demand and destination prediction using historical taxi data

There are few previous research works conducted on taxi demand prediction. Zhang et al. [5] propose a passenger hot-spots recommendation system for taxi drivers. By analyzing the historical taxi data, they extract hot-spots in each time-step and assign a hotness score to each of them. This hotness score is predicted in each time-step and combined with the driver's location, the $top - k$ hot-spots would be recommended. Zhao et al. [6] define a maximum predictability for the taxi demand at street blocks level. They show the real entropy of past taxi demand sequence which proves that taxi demand is highly predictable. They also implement three prediction algorithms to validate their maximum predictability theory. Moreira-Matias et al. [7] propose a framework consisting of three different prediction models. In each time step, the predicted demand is a weighted ensemble of predictions from three models. The ensemble weights are updated with individual prediction performances of previous time steps in a sliding window. Their framework can make short term demand prediction for the 63 taxi stands in the city of Porto, Portugal. Davis et al. [8] use time-series modeling to forecast taxi travel demand in the city of Bengaluru, India. This information can be given to the drivers in a mobile application so that they know where the demand is higher.

Taxi destination prediction is more complex than the demand prediction because it contains more uncertainty. Some well-performing models are using a small window of GPS traces to predict the destination of each trip [9]. We consider a different scenario in which we predict possible destinations for future taxi trips without relying on their GPS traces. In some dispatch systems [10] the destination estimation is sampled from a normalized distribution

of destinations. However, the distribution is simply the historical average of the destinations.

More prediction applications using historical taxi information can be found on topics such as taxi demand, sharing, travel time and destination. Xu et al. propose different learning models to capture the patterns of historical taxi demand and destination distributions in each area over a city [11, 1, 12]. Yuan et al. [13] present a recommender system for taxi drivers and people expecting to take a taxi. They do this recommendation by learning from GPS traces of taxis and also mobility pattern of passengers. Ma et al. [14] propose a taxi ride-sharing system that efficiently serves real-time requests sent by taxi users. Rong et al. [15] model the passenger seeking taxis as a Markov Decision Process (MDP) and propose a method to increase the revenue efficiency of taxi drivers. Azevedo et al. [16] look further in the future and investigate the problem of improving the mobility intelligence of self-driving vehicles through a large-scale data analysis. Jing et al. [17] take a different approach and make recommendations to passengers where they can find taxis easier by analyzing GPS traces of taxis. For a more extensive survey on different approaches to analyze and learn from taxi GPS traces, the reader is referred to a survey [18] that focuses on this topic.

2.2 Taxi dispatch center modeling

Given the estimated future demand and destination, different intelligent transportation systems have been proposed. Alonso-Mora et al. [10] propose a taxi dispatch system considers dynamic vehicle routing to provide ride-sharing in a city. Future taxi demand probability distribution is also estimated in their system to further improve the dispatch performance. Zhang et al. [19] propose a real-time taxi dispatch application. Two types of passengers are defined to model real-time taxi demand: previously left-behind, and passengers arriving shortly. A demand inference model called Dmodel is designed with hidden Markov chain to

describe the state changes of passengers. Miao et al. [20] propose a dispatching framework for balancing taxi supply in a city. Their goal is to match taxi demand and supply and minimize taxi idle driving distance. In their work, the next time-step taxi demand is calculated by the mean value of repeated samples from historical data.

Transportation system with ride-sharing is also a recent popular topic with the hope of improving the utilization of taxis. Chen et al. [21] propose a system for vehicle dispatch and ride-sharing. The goal is to balance the taxi supply-demand ratio while minimizing the idle mileage. Ride-sharing is achieved by solving the taxis schedule with a Mixed Integer Programming (MIP). Lin et al. [22] present a dispatch system for transportation hubs with steady passenger streams. In their work, virtual demand pools, passengers walking time, and ride-sharing mechanism are considered. Trips pairing and taxi scheduling are done by a specific MatchMaking model in their system. Similar studies based on ride-sharing systems are conducted in [23, 24].

2.3 Path planning for mobile sinks in wireless sensor networks

Mobile sinks provide great advantages in wireless sensor networks (WSNs) such as distributing the energy consumption throughout the network and increasing network lifetime. The common goal of path planning for mobile sinks is to maximize the information collected while minimizing travel time.

The value of information (VoI) is such metric that should be maximized as a result of data collection for more accurate and timely decisions. VoI, which was originally formulated for intruder tracking sensor networks [25, 26], has also been applied within the underwater sensor networks domain for path planning [4, 27, 28, 29], scheduling [30, 31], and resurfacing [32]

of autonomous underwater vehicles (AUVs).

Mobile sinks can either visit each sensor node directly or just visit a subset of them. Cheng et al. [33] propose a path planning approach for a mobile sink based on the Traveling Salesman Problem (TSP). Instead of visiting each sensor, the mobile sink only visits a set of virtual points which are actually overlapping areas of communication ranges of sensors. Hollinger et al. [34] address path planning for autonomous underwater vehicle (AUV) to collect data from an underwater sensor network. Each AUV needs to collect as much data as possible while considering fuel expenditure, i.e., the travel time. Salarian et al. [35] propose a path planning approach for the mobile sink called weighted rendezvous planning (WRP) to collect data from the set of rendezvous points (RPs) with the aim of minimizing energy consumption. Li et al. [36] propose a path planning strategy for the UAV/UGV based on genetic algorithm with the goal of building a ground map and plan an efficient and feasible path for disaster rescue. Sangare et al. [37] propose a Markov decision process (MDP) based path planning approach for a Mobile Energy Station (MES) to recharge wireless-powered sensors in order to minimize the average data loss of sensors due to out of power.

Research on mobile sinks to accomplish other tasks is also well investigated. Ma et al. [38] use one SenCar as a mobile sink for data collection in a static sensor network. Their research reveals the effects of traveling path on network lifetime in a given network. They propose a heuristic algorithm for planning the traveling path such that traffic load can be balanced. Turgut and Bölöni [39, 40, 41] investigated transmission scheduling problem in order to make the right decisions since it has significant impact on the performance and lifetime of the node. Rahmatizadeh et al. [42, 43] study sink mobility in virtual coordinates domain and propose a routing strategy to minimize energy consumption while notifying the nodes about the latest location of the sink. Wang et al. [44] introduces m-limited forwarding algorithm to reduce the power consumption of the nodes and improve the routing performance

through forwarding packets to the limited set of nodes. Solmaz and Turgut [45, 46] propose positioning approaches for multiple mobile sinks to optimize event coverage using WSNs. Basagni et al. [27] investigate the problem of maximizing value of information in underwater sensor networks. They formulate the problem using an Integer Linear Programming (ILP) model for path planning of underwater vehicles. Their method achieves better results in terms of value of information compared to a greedy heuristic.

2.4 Monitoring animals in sensor networks

Monitoring animals can be viewed as specific applications of object tracking problems. Its main goal is to track certain animals in a monitored area and reporting their location and other information to the applications users [47]. Many tracking technologies have been proposed and implemented by engineers and wildlife researchers [48, 47, 49, 50]. One main technology is the wearable GPS-based animal tracking devices. Juang et al. [48] present their ZebraNet project in which a low-power wireless system is built for position tracking of zebras. Tracking devices are placed on zebras and record zebras' GPS positions periodically. In their research, they investigate system design ideas, communication protocols between tracking devices, and how sensor specifications such as battery lifetime and weight limit the system performance. In addition, some recent research [51, 52, 53] on animal behavior gather animal movement data by using wearable GPS devices. Although remote sensing can be used for sensing different types of data from a large area [54, 55], sensor networks seem to be a more feasible and reliable choice for animal monitoring.

In recent years, camera sensor networks emerge due to the advancements in hardware technology which provides sufficient bandwidth for transferring multimedia data. Camera sensor networks greatly promote wild-life research by providing much more animal related infor-

mation such as image, sound and video. He et al. [56] develop integrated camera-sensor networking systems and deploy them at large scales for collaborative wildlife monitoring and tracking. They aimed to solve animal species recognition problem by using machine learning methods to train a model based on large number of images. Similar studies based on camera sensor networks are conducted in [57] [58].

In addition, UAVs are being increasingly used as sensor nodes for monitoring various species in nature. Xu et al. propose a Markov decision process (MDP) based path planning strategy for an unmanned aerial vehicle (UAV) under the application scenario animal monitoring [59]. In their another work, path planning approach for the UAV based on animal appearance distribution prediction [60] is also presented. Akbas et al. [61] capture and monitor the social interactions of the complex social network of gorillas - resulting fAPEbook, a generated social directory of the ape troop. Tuna et al. [62] propose to use UAVs for deployment of sensor nodes for post-disaster monitoring. Hodgson et al. [63] use ScanEagle UAVs to survey marine mammals. Their results indicate that UAVs are not limited by sea conditions as sightings from manned surveys. Chamoso et al. [64] propose to use UAVs for scanning large areas of livestock systems.

Using visual recognition techniques, the recorded images are used to count and monitor animal species. Akbas et al. [65, 66] propose the use of aerial sensor networks consisting of UAVs for volcanic eruption monitoring. They propose positioning approaches for multiple UAVs based on the Valence Shell Electron (VSEPR) model of molecular geometries. Our study differs from the aforementioned ones as we propose using UAV as a major element of the WSNs for monitoring purpose. Brust et al. [67] propose 3D virtual forces clustering algorithm (VBCA), which is inspired by the VSEPR model [66], for autonomous positioning of aerial drone networks. The virtual forces gives an opportunity to drones to to self-organize.

CHAPTER 3: TAXI DEMAND AND DESTINATION PREDICTION

In this chapter, we propose two learning models to capture the patterns of historical taxi demand and destination distributions in each area over a city [11, 1, 12]. The trained models can make real-time prediction on future taxi demands and destinations distribution for the whole city.

Traditional transportation systems with all the comfort that provides to humans, still face serious challenges due to the rapid growing traffic and inefficient dispatch operation. For instance, taxi drivers often drive for a long time to pickup a passenger and passengers often need to wait for a long time before a taxi picks them up. In addition to the wasted times, this will lead to a wide variety of problems such as more fuel consumption, traffic congestion, and air pollution [21]. To address these problems, intelligent transportation systems such as vehicle rebalance and ridesharing systems are proposed by the researchers. To make these systems more efficient, we need to understand and predict the demand of the passengers since these systems highly rely on the future demand patterns. Previous studies [5, 6, 19, 7] have shown that historical taxi trip data can provide rich insights about how taxi demand varies from area to area and time to time. In addition, the destination prediction also plays an important role in a transportation system as it provides more detailed vehicle fleet distribution information for a dispatch center. We can also look a bit ahead and consider the future where self-driving vehicles need to autonomously decide where to look for passengers and also to balance the supply-demand ratios over the city without human help [1]. To achieve these, robust and efficient prediction models are necessary and also can be of great help to passengers, human drivers, and autonomous vehicles.

3.1 Network Setting and Data Process

In this application, we use the historical taxi data [68] from New York City (NYC) to demonstrate and evaluate our prediction models. Table 3.1 shows some raw records of taxi pickups from the NYC taxi trip dataset.

Table 3.1: Pieces of raw taxi pickup data.

Pickup_datetime	Pickup_latitude	Pickup_longitude
2016-06-01 02:46:38	40.695178985595703	-73.930580139160156
2016-06-01 02:55:26	40.792552947998047	-73.946929931640625
2016-06-01 02:50:36	40.823955535888672	-73.944534301757813
2016-06-01 02:57:04	40.823871612548828	-73.95220947265625

We first divide the entire city into small areas. There are several ways for such division such as dividing based on zip code. However, the resulting areas by this type of division are too large. For instance, the area size of Brooklyn in New York City is $180km^2$, while there are 37 different zip codes in Brooklyn. This leads to an average of 4.86 km^2 for each zip code area. It is desired to predict taxi demand in small areas so that the drivers know exactly where to go. However, on the other side, learning to predict taxi demand in very small areas is difficult. So, we need to select an area size which is both easy to predict and sufficiently accurate for the drivers.

In this application, we use the Geohash library [69] which can divide a geographical area into smaller subareas with arbitrary precision. Geohash is a geocoding system that has a hierarchical spatial data structure which subdivides space into buckets of grid shape. An example of using Geohash library to encode different pairs of $(latitude, longitude)$ data:

$$g.encode(lat,\ long,\ precision = (1 - 12)) \tag{3.1}$$

$$\begin{cases} g.encode(40.69517898,\ -73.93058013,\ 6) \\ g.encode(40.69517898,\ -73.93058013,\ 7) \\ g.encode(40.6951789801,\ -73.9305801301,\ 7) \\ g.encode(40.6951789899,\ -73.9305801399,\ 7) \end{cases} \tag{3.2}$$

In Eq. 3.2, $(latitude, longitude)$ pairs encoded with precision 7 will be converted to $'dr5rt8m'$, and the one encoded with precision 6 will be identified by code $'dr5rt8'$. In Geohash, each code represents a divided area. After encoding all the taxi trip GPS information, coordinates locate in the same area have the same geohash code - this can be seen from the last three pairs in Eq. 3.2. In addition, neighboring areas share the same code prefix.

In this way, we can quickly divide the entire city into areas with arbitrary size and have the taxi demand sequence in each area. In our experiment, we use taxi data from 1/1/2013 through 6/30/2016 at NYC, which includes around 600 millions taxi trips after data filtering. We divide the entire city into around 6500 small areas with precision 7 and for each area, we count the number of taxi pickups at each time-step. We also test our model on precision 6 which divides the city into about 1000 areas.

3.2 Taxi demand prediction

In this section, we propose a taxi demand predictor that can predict taxi demand in any target area of the city in the next hours, days and weeks. For any given area, the past taxi demand can be treated as a sequence.

Fig. 3.1 shows taxi demand at two different places in New York City over a period as long as a week. We observe that in a specific area of the city, the historical taxi demand shows a predictable sequential pattern every week. Motivated by observing this pattern, we design a sequence learning model that learns the demand patterns from the sequential data.

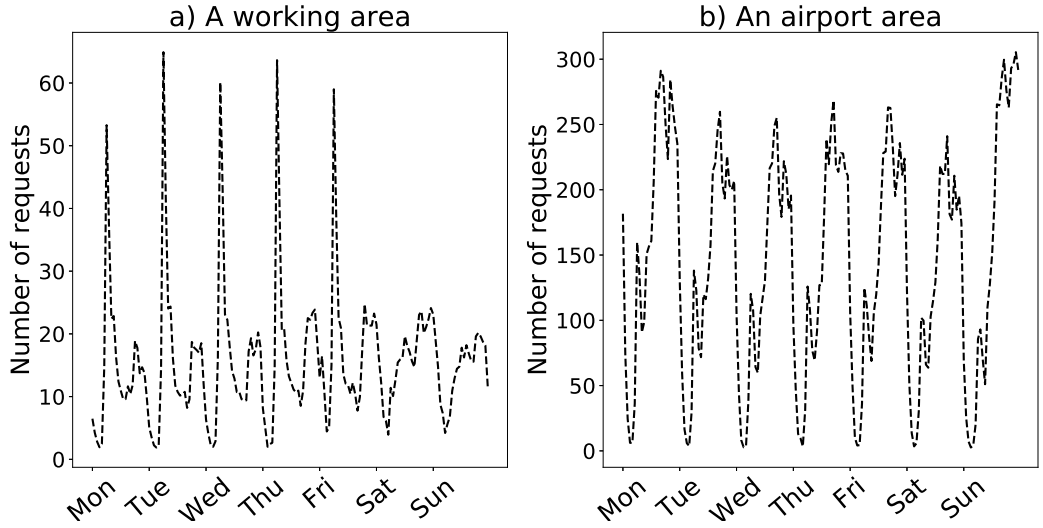

Figure 3.1: Taxi demand patterns in two different areas.

We first divide a day into discretized time-steps $\{t_0, t_1, ...t_{max}\}$ where t_i is the i^{th} time-step of a day. Note that the time-step length is a hyper-parameter. Second, for each area, we count the number of taxi requests in each time-step. Fig. 3.2 shows the input and output data structures in one time-step. For time-step t_i, the input data x_i consists of two parts: $[f_i, e_i]$. f_i represents potential affecting factors such as date, day of the week, time-step in the day and weather. We use the official historical weather information of NYC from National Oceanic and Atmospheric Administration (NOAA). e_i represents the number of pickups in each area and its length is the number of small areas in the entire city.

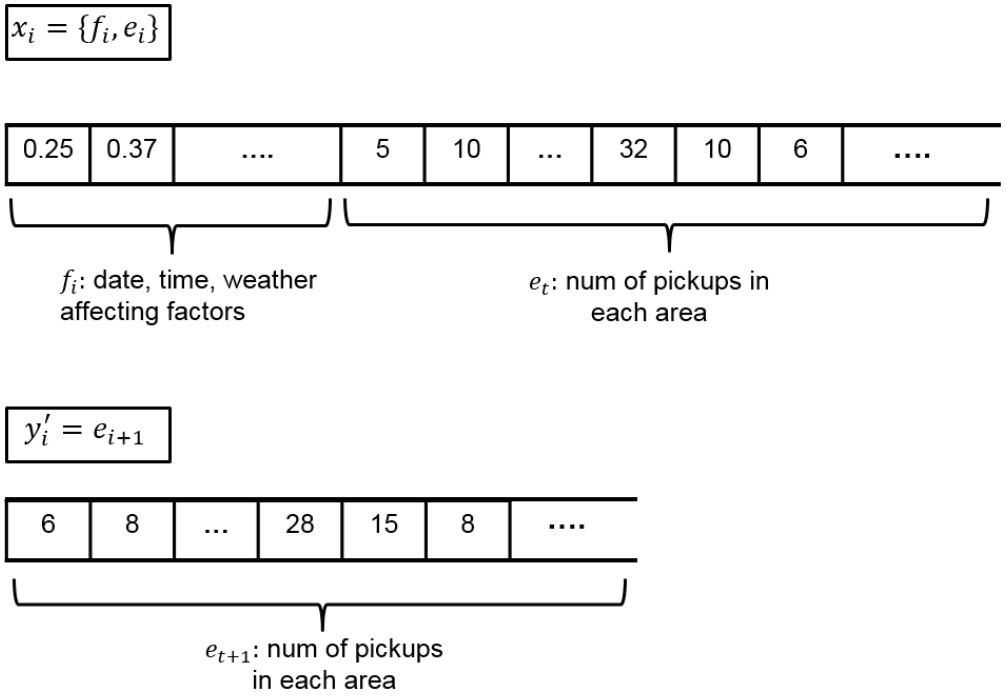

Figure 3.2: Input and output data structure for taxi demand prediction.

3.2.1 Taxi demand prediction - LSTM model

To build a sequence learning model, we use one of the best Recurrent Neural Networks (RNNs): Long Short Term Memory (LSTM). As shown in Fig. 3.3, the input data to the model at time-step t_i is $[x_{i-seq}, ...x_{i-1}, x_i]$. The meaning of *seq* here is that we are using previous *seq* time-steps data to predict next time-step data. *seq* is a hyper-parameter that is set large enough to enable the network to learn long-term dependencies. Given the input data at t_i, the network predicts the output y_i', the number of requests in each area at the next time-step. To train the network using stochastic gradient descent, we try to minimize the mean squared difference between the predicted y_i' and the ground-truth demand y_i.

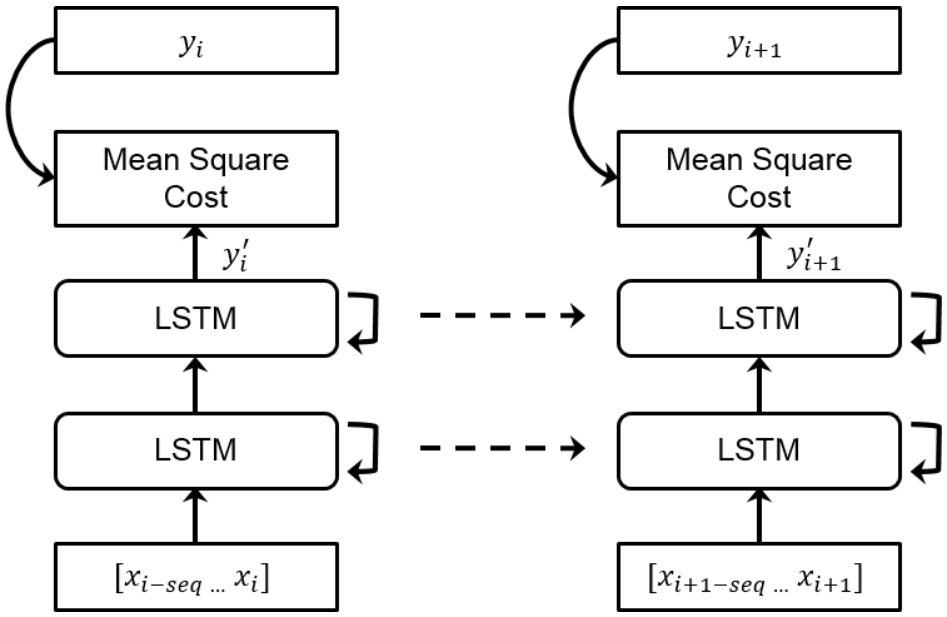

Figure 3.3: The LSTM taxi demand pattern learning model.

The regression model above can provide great prediction results towards future taxi demand. However, we can further improve its performance by considering stochastic factors in real-valued data.

3.2.2 Taxi demand prediction - LSTM-MDN model

The most successful application of neural networks has been achieved on classification tasks. When it comes to predicting real-valued data, the choice of network structure is very important. The idea of mixture density networks (MDNs) [3] is to use the outputs of a neural network to parameterize a mixture distribution. Unlike the model with mean squared error (MSE) cost which is deterministic, MDNs can model stochastic behaviors. They can be used in prediction applications in which an output may have multiple possible outcomes.

In our application, rather than directly predicting the number of taxi requests, the neural network outputs the parameters of a mixture model. These parameters are the mean and variance of each Gaussian kernel and also the mixing coefficient of each kernel which shows how probable that kernel is. Given the parameters of the mixture distribution, we can draw a sample from it and use this sample as the final prediction.

The sequence learning model is created based on an LSTM recurrent neural network and the MDNs. Fig. 3.4 shows the structure of the unrolled LSTM-MDN learning model.

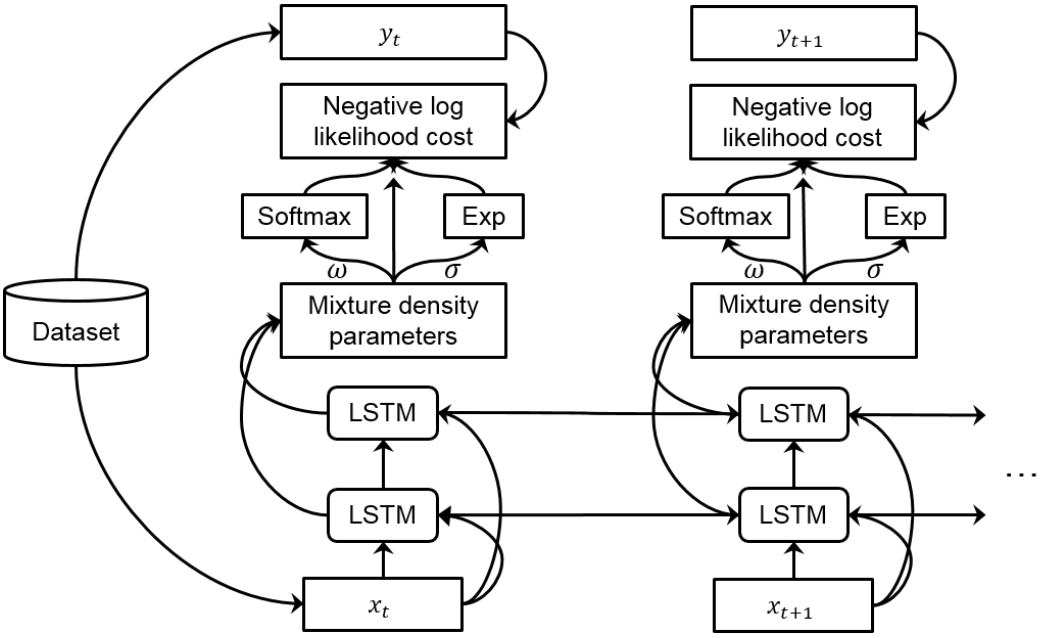

Figure 3.4: The LSTM-MDN learning model unrolled through time-steps.

In Fig. 3.4, the total unrolling length is a hyper-parameter that can be set according to testing scenario. LSTM can encode the useful information of the past in a single or multiple layers. The input to each layer is the output of the previous layer concatenated with the network input. Each LSTM layer predicts its output based on its current input and its

internal state. The concatenation of outputs of all layers will be used to predict the output of the network which will be compared with y_t, the real demand value from the dataset, to form the error signal. We use two LSTM layers in which each layer contains $1200 - 1500$ neurons based on the specific testing scenario.

As shown in Fig. 3.4, the output of LSTM layers are mixture density parameters with the total number of $M \times (N + 2)$ in which M is the number of Gaussian kernels, and N is the number of areas in the city. For each Gaussian kernel, we have N neurons for the means $\mu_k(x)$, one neuron for the variances $\sigma_k(x)$, and another neuron for the mixing coefficient $w_k(x)$. To satisfy the constraint $\sum_{k=1}^{M} w_k(x) = 1$, the corresponding neurons are passed through a softmax function. The softmax function is regularly used in multiclass classification methods to "squash" a vector of n arbitrary real values z into a set of values that add up to 1, and which can be interpreted as probabilities:

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{n} e^{z_k}} \tag{3.3}$$

The neurons corresponding to the variances $\sigma_k(x)$ are passed through an exponential function and the neurons corresponding to the means $\mu_k(x)$ are used without any further changes. The probability density of the next output y_t can be modeled using a weighted sum of M Gaussian kernels:

$$p(y_t|x) = \sum_{k=1}^{M} w_k(x) g_k(y_t|x) \tag{3.4}$$

where $g_k(y_t|x)$ is the k^{th} multivariate Gaussian kernel. Note that both the mixing coefficient

and the Gaussian kernels are conditioned on the complete history of the inputs till current time-step $x = \{x_1 \dots x_t\}$. The multivariate Gaussian kernel can be represented as:

$$g_k(y_t|x) = \frac{1}{(2\pi)^{N/2}\sigma_k(x)} \exp\left\{-\frac{\|y_t - \mu_k(x)\|^2}{2\sigma_k(x)^2}\right\} \tag{3.5}$$

where the vector $\mu_k(x)$ is the center of k^{th} kernel. We do not calculate the full covariance matrices for each kernel, since this form of Gaussian mixture model is general enough to approximate any density function [70]. Finally, we can define the error function in terms of negative logarithm likelihood:

$$E_t = -ln\left\{\sum_{k=1}^{M} w_k(x) g_k(y_t|x)\right\} \tag{3.6}$$

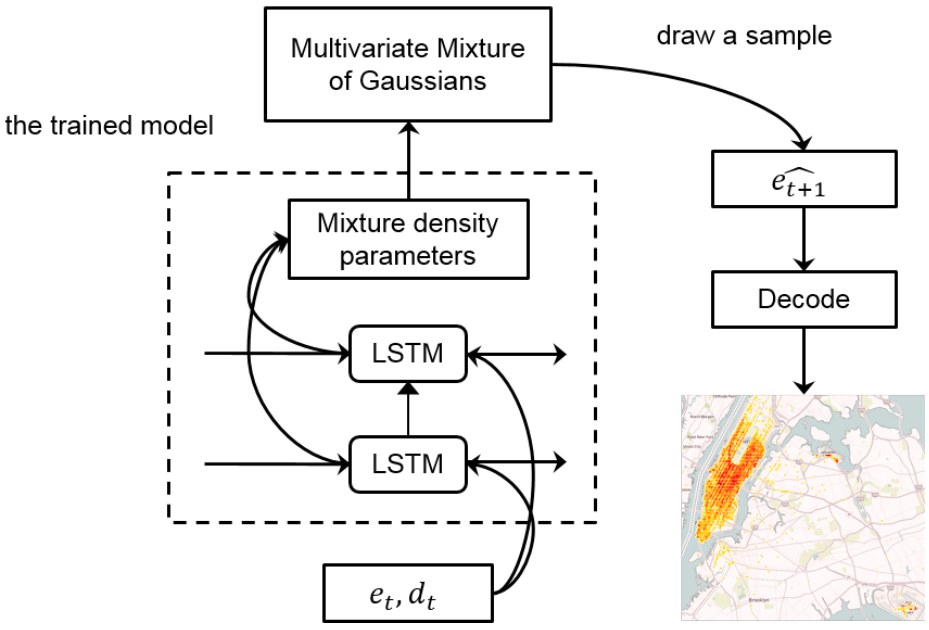

Figure 3.5: The LSTM-MDN model perform one prediction for time $t + 1$.

After the model is trained, we can make a prediction for time-step $t+1$ by inputting taxi demand at time-step t. As we can see in Fig. 3.5, we use the output which is the mixture density parameters to parameterize a Gaussian mixture distribution. A sample can then be drawn from this distribution and this sample would be the prediction of the next time-step taxi demand, $\widehat{e_{t+1}}$. This prediction can be repeated in a loop to predict taxi demand for multiple time-steps.

3.2.3 Taxi demand prediction - LSTM-MDN-Conditional model

In the LSTM-MDN model, the probability distribution of taxi demands in all areas are predicted at the same time-step. This means that prediction in each area is conditioned on all areas of all previous time-steps. However, the taxi demand in an area might not only be related to the past, but also to the taxi demands of other areas in current time-step. So instead of outputting a joint distribution for all areas in a single time-step, we ask the network to predict the conditional distribution of each area at a single time-step. This approach has been adopted in other works such as in Pixel RNNs [71] and Neural Autoregressive Distribution Estimator (NADE) [72]. Fig. 3.6 shows the idea of generating y_t^i conditioned on all the previously predicted demands left. Here y_t^i represents the predicted taxi demand in area i at time-step t.

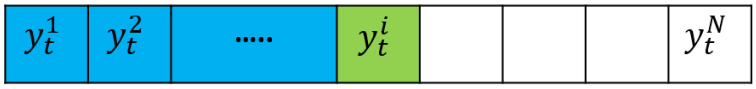

Figure 3.6: Generation of conditional distributions sequentially.

We call this approach LSTM-MDN-Conditional model. It has the same input x_t as the LSTM-MDN mode, but each x_t only leads to one area taxi demand output. Unlike predicting

taxi demands for all areas in LSTM-MDN mode, this model sequentially predicts demand for each area in a conditional way. Fig. 3.7 shows the unrolled LSTM-MDN-Conditional model structure for one time-step prediction.

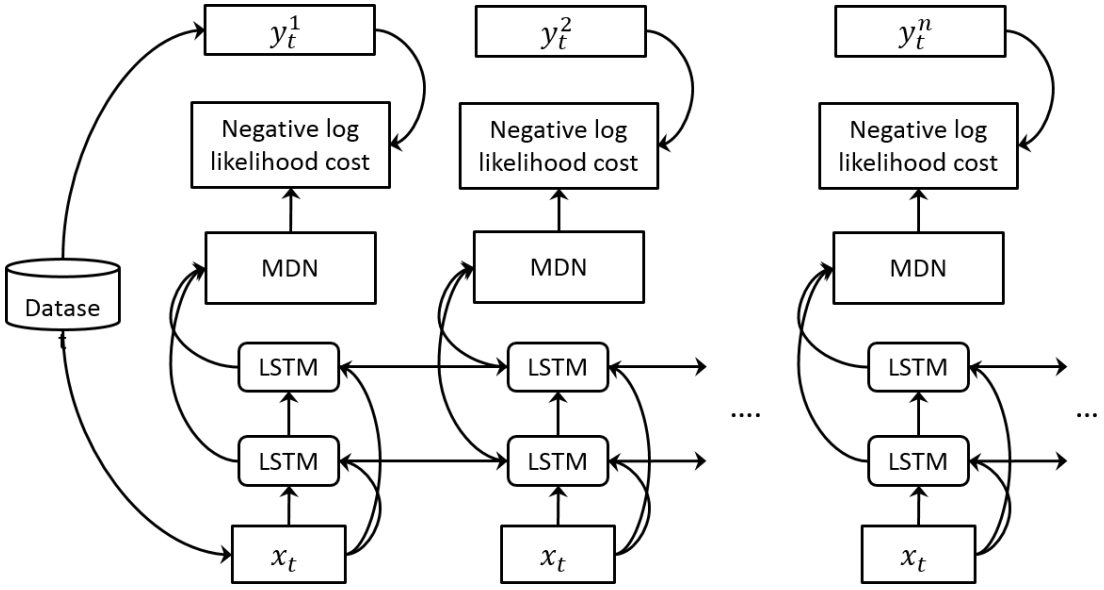

Figure 3.7: The unrolled LSTM-MDN-Conditional model for one time-step prediction.

This LSTM-MDN-Conditional model not only learns demand patterns from past taxi data, but also takes into account current demands in other areas. Training such a model takes much longer time than the LSTM-MDN model because the LSTM needs to be unrolled for much longer time-steps. For a city with N areas, the model needs to be unrolled N times more compared to the LSTM-MDN model.

With the trained model, we can predict taxi demands for all areas in future. Fig. 3.8 shows a density map of real and predicted taxi demands over the entire city. As we can see that red areas show high demand while yellow areas show lower demand. The figure illustrates that the difference between the predicted and the real demand is very small.

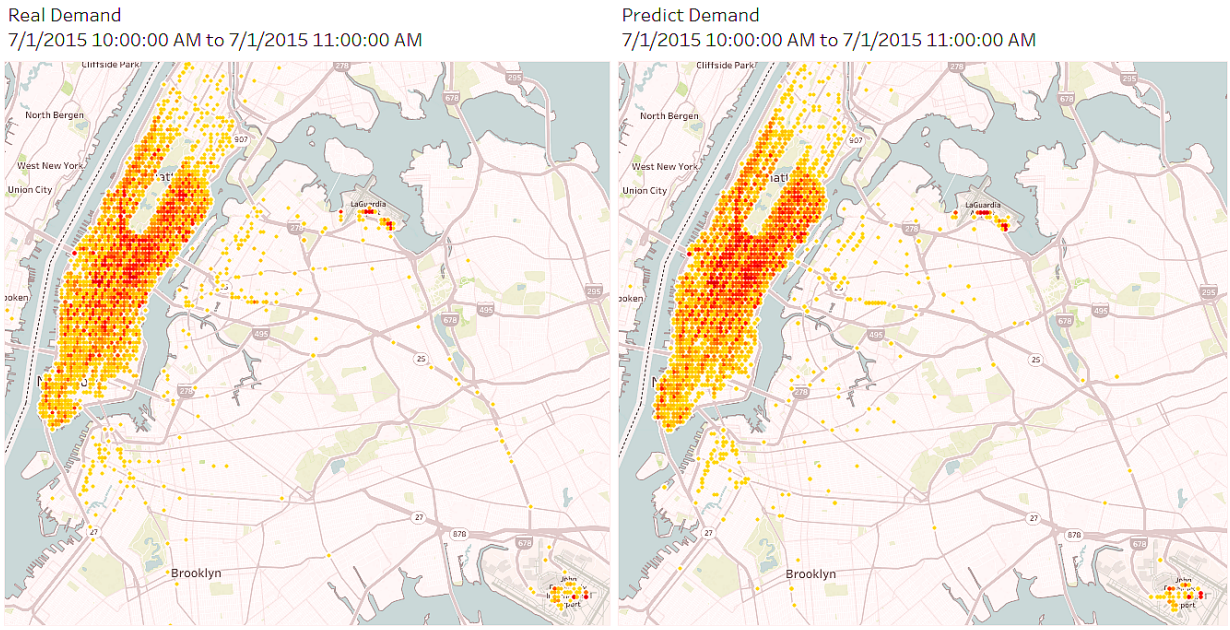

Figure 3.8: The density map of real demand and the predicted demand.

3.3 Taxi destination prediction

Destination prediction is much harder than the demand prediction because it contains much more uncertainty. Some existing works [9] are using a small window of GPS traces to predict the destination of each trip. We consider a different scenario in which we predict possible destinations for future taxi trips without relying on their GPS traces. We solve it as a distribution prediction problem from a statistical perspective.

Consider a trip request from one of the areas, its destination can be any area in the city. The data distribution is multi-modal, i.e. for each input there are multiple possible outputs. This motivates us to use Mixture Density Networks (MDNs), developed by Christopher Bishop [3] that is designed to model real-valued multi-modal distributions. The idea behind MDN is to use the output of a neural network to predict the parameters of a mixture Gaussian kernels.

Note that Gaussian kernels have a different set of parameters in each area. With the learned parameters, we can sample the destination predictions for each taxi trip.

To build a distribution learning model, we first extract each trip information from historical taxi dataset, which contains time-stamp, pickup location, and dropoff location. Then we encode it into a data structure shown in Fig. 3.9. One of the advantages of using Geohash library is that the encoded neighboring areas will stay close when we sort them. Each trip is converted into a pair of data point $[x_k, y_k]$, where k represents the trip index in the dataset. x_k consists of the pickup area and the corresponding factors such as time step since the beginning of the day, day of the week and weather. y_k represents the destination area of this trip.

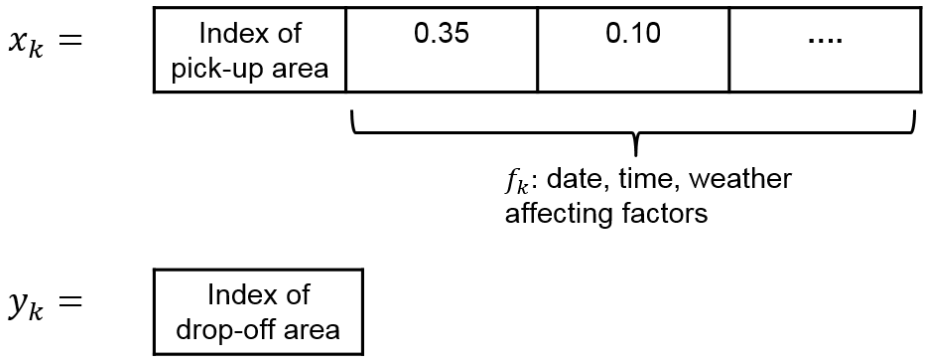

Figure 3.9: Input and output data structure for destination prediction.

Fig. 3.10 shows the designed distribution learning model. The goal is to learn the parameters of mixture of Gaussians for each area. As shown in Fig. 3.10, we feed x_k to a fully connected neural network. The expected output is a vector of distribution parameters with length $3 \times M$. M is the number of Gaussian kernels, which is a hyper-parameter. Each Gaussian kernel consist of 3 variables $[\omega, \mu, \sigma]$. ω is the mixing coefficient, μ is the mean and σ is the standard deviation.

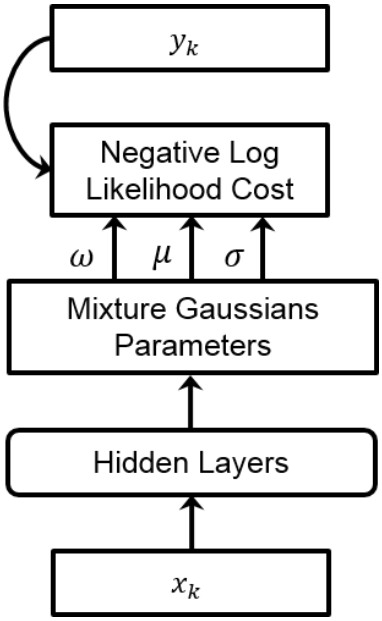

Figure 3.10: Taxi destination distribution learning model.

A suitable loss function is to minimize the logarithm of the likelihood of the distribution of the training data:

$$Cost = -ln\left\{\sum_{m=1}^{M} w_m(x)\phi_m\{y, \mu_m(x), \sigma_m(x)\}\right\} \tag{3.7}$$

where $\phi_m\{y, \mu_m(x), \sigma_m(x)\}$ is the m^{th} Gaussian kernel. It can be represented as:

$$\phi_m\{y, \mu, \sigma\} = \frac{1}{2\pi\sigma_m(x)} \exp\left\{-\frac{|y - \mu_m(x)|^2}{2\sigma_m^2(x)}\right\} \tag{3.8}$$

For each pair $[x_k, y_k]$ in the training dataset, we can calculate the cost based on the predicted distribution versus the actual value, and then attempt to minimize the sum of all the costs combined. Fig. 3.11 (a,b) show two examples of destination distributions start from two different areas. The time period is 30 $mins$. We cluster neighboring areas sorted by Geohash

into different bins for better visualization. The horizontal axis represents the area index while the vertical axis represents the number of dropoffs in the area. Fig. 3.11 (c,d) shows the corresponding predicted distributions. As we can see, there are some differences compared to the real distributions in Fig. 3.11 (a,b) but overall the differences are small.

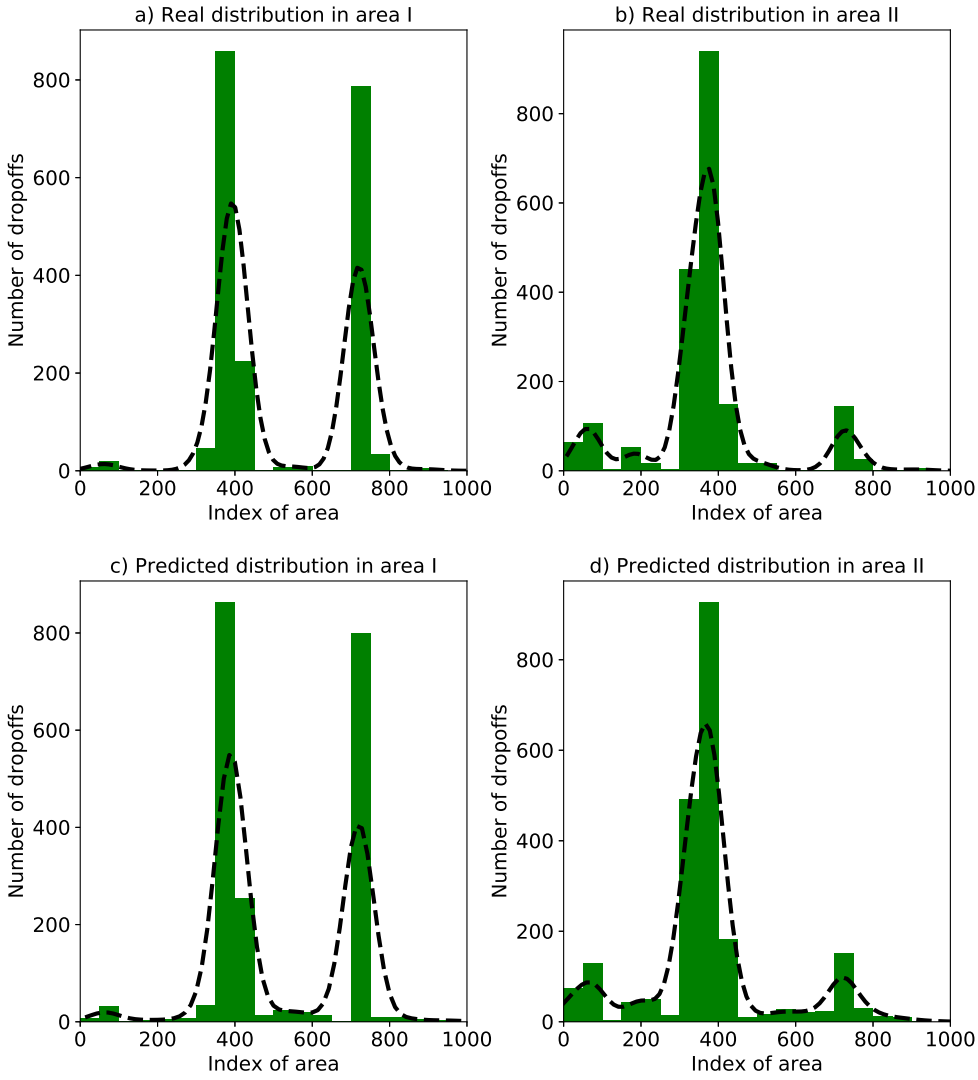

Figure 3.11: Real and predicted destination distributions of two different start areas.

3.4 Prediction models evaluation

In this section, we evaluate the proposed taxi demand and destination prediction models. The evaluation is conducted with the New York City (NYC) taxi trip dataset [68]. There are two kinds of cabs in NYC: the yellow cabs, which operate mostly in Manhattan, and the green cabs, which operate mostly in the suburbs. The dataset contains taxi trips from January 2009 through June 2016 for both yellow and green cabs. Each taxi trip has a pickup time and location information.

3.4.1 Experimental setup

In this evaluation, we use the most recent 3.5 years data in NYC taxi dataset: from January 2013 through June 2016, which contains over 600 million taxi trips after data filtering. We use 80% of the data for training and keep the remaining 20% for validation. The network model is implemented in Blocks [73] framework that is built on top of Theano [74]. We stop the training when the validation error does not change for 20 epochs. The training takes 2-4 hours on a GTX 1080 for each of the experiments. After the training, the time that takes for the network to predict the demand is less than a second. Note that the prediction time is more important than the training time. This is because the model can be trained once but once deployed it needs to predict the demand in a loop to provide this information in real-time.

Theoretically, LSTM can be trained using arbitrary sequence lengths. However, constrained by the computational power, we use every one week data as a sequence and cut it into time-steps with different lengths. For example, if the time-step length is $60mins$, the sequence length would be 24×7. If the time-step length is $20mins$, the sequence length would be

$24 \times 3 \times 7$. For the $60mins$ case, the encoded input data shape is $(182, 168, 6494)$ in which 182 is the total number of sequences in the dataset, i.e., number of weeks in the 3.5 years, 168 is the sequence length: 24×7, and 6494 is the number of features consisting of number of areas, impacting factors such as date, day of the week and other information. Table 3.2 includes the list of parameters in the experiments.

Table 3.2: Experimental parameters

Area/grid size	$\leq 153m \times 153m$
Data of each sequence	1 week data
Time-step length	5/10/20/30/60 mins
Sequence length	2016/1008/504/336/168
Number of sequences	182
Number of areas (N)	6424
Number of hidden layers	1-2
Number of nodes in each hidden layer	1200-1500
Number of mixture Gaussian kernels	10-20

3.4.2 Performance metrics and baselines

To systematically examine the performance of our prediction approach, we include results with two widely used prediction error metrics: Symmetric Mean Absolute Percentage Error (sMAPE) [7, 75] and Root Mean Square Error (RMSE) [76, 77]. $Y_{i,t}$ is the real taxi demand in area i at time-step t, while $\widehat{Y}_{i,t}$ is the predicted taxi demand. The sMAPE and RMSE in area i over time $[1 - T]$ would be:

$$sMAPE_i = \frac{1}{T} \sum_{t=1}^{T} \frac{|Y_{i,t} - \widehat{Y}_{i,t}|}{Y_{i,t} + \widehat{Y}_{i,t} + c} \tag{3.9}$$

$$RMSE_i = \sqrt{\frac{1}{T}\sum_{t=1}^{T}\left(Y_{i,t} - \widehat{Y}_{i,t}\right)^2} \tag{3.10}$$

The constant c in Eq. 3.9 is a small number ($c = 1$ in this application) to avoid division by zero when both $Y_{i,t}$ and $\widehat{Y}_{i,t}$ are 0. Similarly, when evaluating the prediction performance over the entire city, the sMAPE and RMSE of all areas at time-step t would be:

$$sMAPE_t = \frac{1}{N}\sum_{i=1}^{N}\frac{|Y_{i,t} - \widehat{Y}_{i,t}|}{Y_{i,t} + \widehat{Y}_{i,t} + c} \tag{3.11}$$

$$RMSE_t = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_{i,t} - \widehat{Y}_{i,t}\right)^2} \tag{3.12}$$

Here N is the total number of areas in the city. RMSE shows the difference of the predicted value from the real value while the sMAPE describes a percentile error.

To evaluate the performance of the proposed models, we compare the outcomes with prediction approaches based on another two strategies: the fully connected feed-forward neural networks and naive statistic average.

Fully connected feed-forward neural network predictor

Feed-forward neural networks are commonly used for classification and regression problems. Feed-forward neurons have connections from their input to their output. The main difference between feed-forward neural networks and recurrent neural networks is that in RNNs, the recurrent connection from the output to the input at next time-step makes the network capable of storing information. In this approach, the layers are fully connected which means

that neurons between two adjacent layers are all connected together.

Naive statistic average predictor

This approach predicts based on the mean value of past demands in a sliding-window. For example, if it is 10:00 am on Monday, the predicted value would be the average of demands at 10:00 am in the past 5 Mondays. While we use the term "naive", even this approach requires the maintenance of long term, detailed statistics of both spatial and temporal distributions of pickups. It is very likely a good approximation of what taxi companies are currently deploying.

3.4.3 Prediction performance

First, we report prediction sMAPE and RMSE over the entire city (all prediction areas). Second, we show the prediction performance at some specific areas as time passes. Lastly, we analyze the importance of different impacting factors on prediction performance. For the four different predictors based on LSTM-MDN-Conditional, LSTM-MDN, fully connected feed-forward neural networks and naive statistic average, we respectively use *LSTM-C, LSTM, FC* and *Naive* for short. We do not include the LSTM-C predictor in the entire city performance comparison. We only evaluate the LSTM-C model on specific areas. This is because conditioning only neighboring areas on each other should be enough. Two areas that are far from each other most probably will not affect each other. In addition, if we condition more areas on each other, LSTM will have a difficult time relating the events that happen many time-steps away from each other.

Performance over the entire city

To evaluate the prediction performance over the entire city which includes about 6500 ar-

eas, we compare the performance of the LSTM predictor, the FC predictor and the Naive predictor in terms of RMSE and sMAPE from Eq. 3.11 and Eq. 3.12.

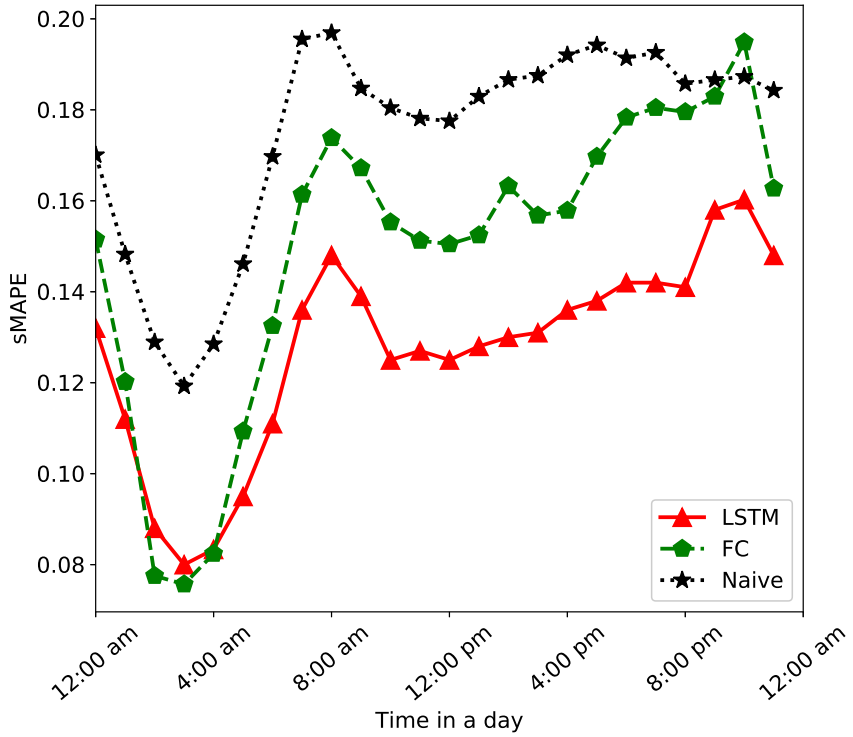

Figure 3.12: Demand prediction performance of different approaches according to sMAPE.

We report sMAPE and RMSE over the entire city in Fig. 3.12 and Fig. 3.13. As we can see, though they are different prediction error metrics, they share some common patterns. For instance, both of them reach the minimum values at about 3am and peak at about 8am and 10pm. In both figures, LSTM shows better performance in prediction than the FC and Naive predictors.

In Fig. 3.12, sMAPE shows the mean absolute percentage error, which gives us a way to calculate the prediction accuracy and observe that it is more than 80%. In Fig. 3.13, RMSE

shows the root mean squared difference between the predicted demands and the real demands. Note that, to the real demands in different areas, we have $min = 0$, $max = 535$ and standard deviation $\sigma = 12.0$. The time-step length is $60mins$.

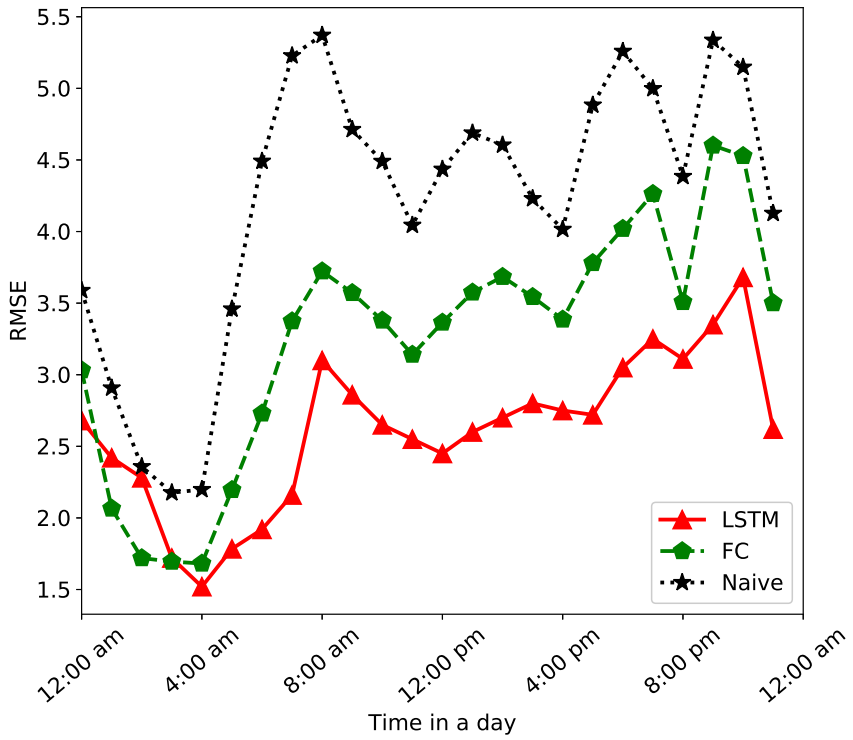

Figure 3.13: Demand prediction performance of different approaches according to RMSE.

Fig. 3.14 reports the error bar of prediction RMSE over the entire city, with the standard deviation in one week. We show this RMSE with different time-step lengths in the LSTM predictor. Basically, smaller time-step length means smaller number of pickups in each time slot, which does affect the final RMSE. To avoid this, we sum up the predicted number of pickups every $60mins$. As we can see in Fig. 3.14, the model has the minimum RMSE at time-step length either 10 or $20mins$. Overall, the RMSEs are very close under different time-step lengths.

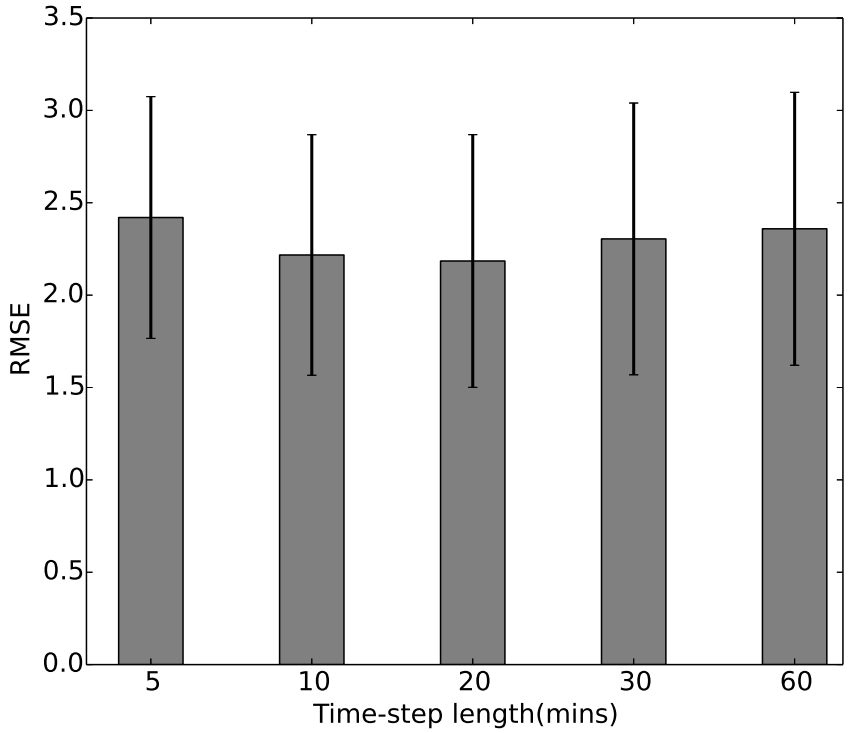

Figure 3.14: Demand prediction RMSE with different time-step lengths. With the real number of pickups, $min = 0$, $max = 535$ and standard deviation $\sigma = 12.0$.

Fig. 3.15 reports the destination prediction performance. It can be seen that Gaussian mixture model performs better than the normalized average-based method. Besides, since our prediction is sampled from a continuous distribution, if we accept neighboring areas as correct predictions, we achieve better results as shown in Fig. 3.15 (Prediction-GM-Neighbors). Note that predicting neighboring areas as destination is still acceptable and useful for our taxi dispatch application since the taxi is very close to the target area.

On the other hand, considering Fig. 3.12 and Fig. 3.15 we observe that interestingly both of them have a small value at around 4:00am. This however means that demand prediction achieves the best performance (lowest error) while destination prediction hits its worst accu-

racy during that time period. The reason could be that the total number of requests is low and mainly from crowded areas such as airports and bars, but the corresponding destinations can be anywhere around the town and are not that predictable.

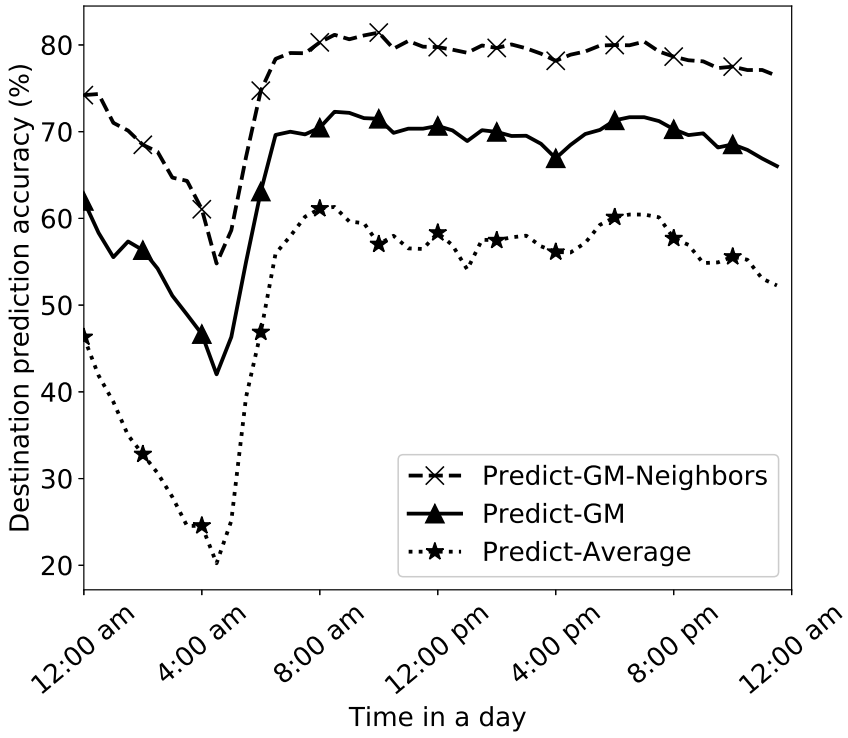

Figure 3.15: Destination prediction performance of different approaches according to accuracy.

Performance at specific areas

We compare the prediction performances of LSTM-C, LSTM, FC and the Naive predictors in specific areas. First of all, we select two areas that their taxi demands in a week are shown in Fig. 3.16-a and Fig. 3.16-b. The reason we select these two areas is that both of them show regular demand patterns on both weekdays and weekends. The first area is a working area

while the second area is one of the most popular areas (in terms of taxi requests) according to our analysis of all the past taxi data in NYC. The first area is close to the intersection of West 40rd Street and 9th Ave. in downtown while the second area is close to the intersection of West 33rd Street and 7th Ave.

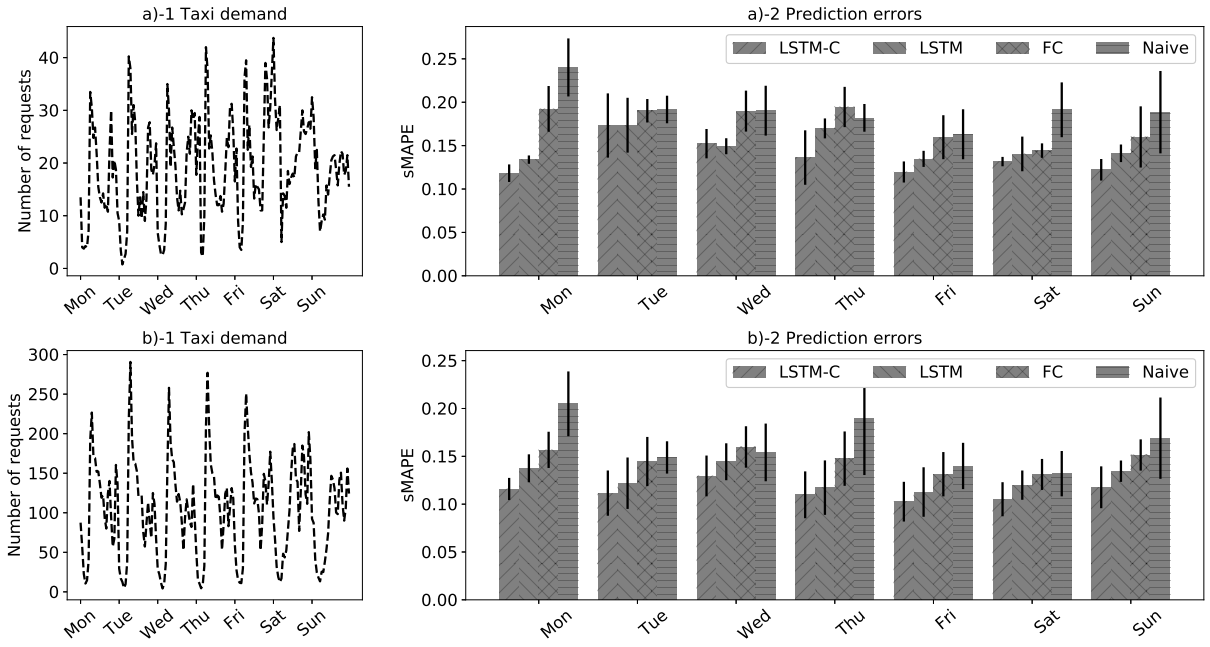

Figure 3.16: Comparison in areas with different demand patterns.

The time-step length used here is $60mins$ in both areas. The right part of Fig. 3.16 shows the prediction sMAPE of each day in both areas. We include continuous 5 weeks prediction results and show each standard deviation with an error bar.

As we can see from Fig. 3.16, in both areas, LSTM-C and LSTM models outperform FC and Naive models in most days. This proves LSTM is good at learning sequential information, even though the sequence length is as long as a week. LSTM-C can give more accurate prediction results than the LSTM, because it considers both past information and current

conditional information of other areas. But it requires a lot of computational power to train such a conditional model. FC sometimes results in larger errors than the Naive predictor due to the irregularities in sequence patterns.

Importance of impacting factors

The impacting factors in our model include *date & time, day of week, weather and drop-offs*. *Date & time* includes year, month, date and time of each time-step. *Day of week* represents which day of week that time-step is. *Weather* includes 4 different types: rain, snow, fog and thunder. We get the official weather information of NYC from National Oceanic and Atmospheric Administration (NOAA). It includes climate observations from three land-based stations in the city. We also include the number of *drop-offs* as an impacting factor to see if there is any relation between the pickups and drop-offs in each area. To show the impacts of these inputs on prediction performance, we conduct two experiments which are implemented on the LSTM-MDN model as we evaluate the prediction performance for the entire city.

Table 3.3: Model with different impacting factors: I

Model	Model input	Model output
Model A	Pickups	Next time-step demand
Model B	Date & Time	Next time-step demand
Model C	Day of week	Next time-step demand
Model D	Weather	Next time-step demand
Model E	Drop-offs	Next time-step demand

In experiment I, we show the impact of each single factor on prediction performance. We design different models, shown in Table 3.3, with each single factor as model input. The control group is the model with only past taxi pickups as input. All models are expected to output the next time-step taxi demand in the city. We the train each model and the prediction performance in shown in Fig. 3.17.

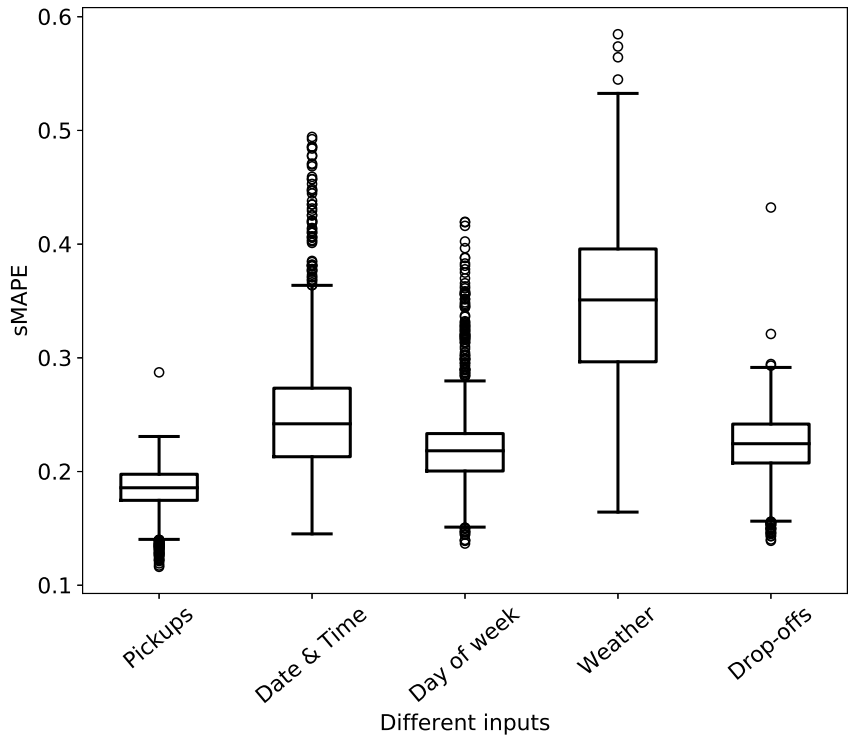

Figure 3.17: Prediction performance on different single impacting factors.

As shown in Fig. 3.17, pickup is the most valuable information in making future taxi demand prediction. In model B and C, since no past taxi trip information is provided, the models are trying to remember the mapping from the input to the real taxi demand at each time-step. In this case, LSTM works similar to the feed-forward neural networks. Model D has the worst performance due to two reasons: it is hard for the model to make a taxi demand prediction only based on a weather information and having climate information from three land-based observations only may not be a good descriptor of the whole city. In model E, we use number of drop-offs in each area as input. It is interesting to find out it can give a prediction performance close to model C, which means that the drop-offs pattern has a close relation to the pickups pattern. However, the drop-off information is not as useful as pickup information. This is because the network needs to remember the drop-off information for

later use. But, the pickup information is the most informative feature probably because it does not change much in one time-step (from input to output).

Table 3.4: Model with different impacting factors: II

Model	Model input	Model output
Model A	Pickups	Next time-step demand
Model B	Pickups + Date & Time	Next time-step demand
Model C	Pickups + Day of week	Next time-step demand
Model D	Pickups + Weather	Next time-step demand
Model E	Pickups + Drop-offs	Next time-step demand
Model F	Pickups + All factors	Next time-step demand

In experiment II, we add impacting factors to the number of pickups as model inputs to see if they really improve the prediction accuracy. Table 3.4 shows the input to each model. In the last group model F, we include pickups and all impacting factors as input.

Fig. 3.18 shows each model's prediction performance. It can be seen that all the models have close performances. The reason is that pickup information is so informative that makes the effect of other factors very small. Model D and E have the worst performance compared with other models. In model F, we include all impacting factors. As we can see, the median prediction error is about 17%, which means a median accuracy of around 83% can be obtained.

Overall, the experimental results show that LSTM-C and LSTM outperform the other prediction approaches. This is because LSTM can see and process information in the previous time-steps. For instance, if a group of people request taxis to go to a concert, it will remember this information and use it to predict that after a couple of hours there would be almost the same number of requests in the concert area. The FC network can find the best mapping from the time and geographical information to the number of requests without having

access to the demand in the previous time-steps. So this limitation causes larger errors in its prediction. The Naive approach is even more restricted since it has access to only a small history of the demand in one area unlike the FC which is trained on all the historical data of all areas.

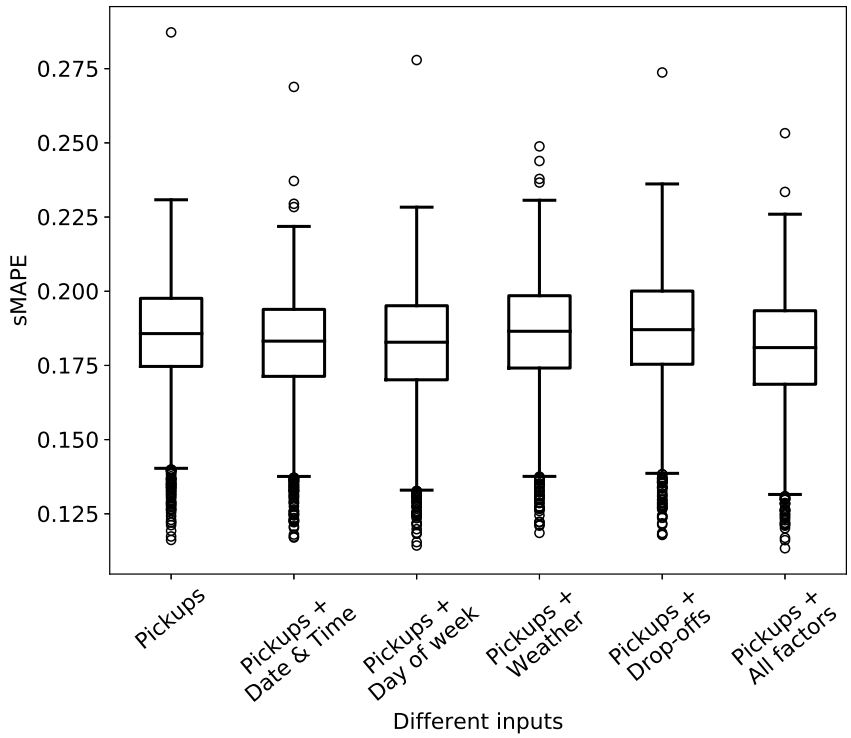

Figure 3.18: Prediction performance on different combination of impacting factors.

To sum up, for better prediction, we need to use a model that is powerful and properly conditions the output on all the available information. In addition, the best prediction performance is achieved when all the impacting factors considered in this work are available as input to the network.

CHAPTER 4: TAXI DISPATCH SYSTEM WITH DEMAND AND DESTINATION PREDICTION

In Chapter 3, we propose an approach for predicting both the taxi demand and destination distributions. In this chapter, we present a taxi dispatcher system that takes advantage of these predictions. The goal is to balance taxis supply-demand ratio over the city [11, 1, 12].

The emergence of ride-sharing systems had disrupted the public transportation model in many areas in the world. They brought clear improvements to the travelling experience and lowered the cost to the users [78]. However, the increasing number of ride-sharing vehicles on the roads also came with increased fuel consumption, air pollution and traffic congestion [21].

One possible way to make taxi services and ridesharing systems more efficient is to understand and predict passenger demand patterns. The studies such as [5, 6, 19, 7] have shown that historical taxi trip data can provide rich insights about the temporal and spatial variation of the taxi demand, that is the points at which the taxi trips are *originated*. In addition, the prediction of the *destination* of the trips also plays an important role in a transportation system, because it help predict the spatial distribution of the vehicle fleet at some time in the future [79]. These problems will still remain if human drivers are replaced with self-driving cars, as the passenger demand does not depend on the (possibly automated) drivers but on the human passengers.

We propose different predictors for both the taxi demand and destination distributions in Chapter 3. Using these predictors we describe a taxi dispatch system that balances the supply-demand ratio throughout the city. The system optimizes the taxi assignment and reallocation by solving a mixed integer programming (MIP) problem with the goal of mini-

mizing the average waiting time of the passengers and the idle driving distances of the taxis. The system takes into account the variable speed of the taxis which depend on the time of the day and differ on weekdays and weekends.

In our experiments, both the demand and destination prediction models were trained on the New York City taxi trip dataset [68] for the entire 2015. We evaluated the performance of the prediction models and the dispatch system with taxi data in 2016 from the same dataset. We show that both the individual predictors and the dispatching system outperform simpler, mean based predictors.

4.1 Network Setting

As we discussed in the Chapter 3, we divide the whole city into a list of small areas, all_areas. For distances between different areas, we generate a distance matrix for each pair of areas $(a_i, a_j), i, j \in N$, where $N = len(all\_areas)$ is the total number of areas. We assume a_i is the pickup area while a_j is the dropoff area. The trip distance between a_i and a_j is written as d_{ij}.

As shown in Fig. 4.2, for pairs of areas (a_i, a_j) that there are recorded trips between them, the d_{ij} is the average distance for all trips between a_i and a_j from historical taxi data. For pairs of areas (a_i, a_j) that there are no recorded trips between them, we use a bidirectional search to find intermediate areas between them and then the shortest path $argmin(d_{ip} + d_{pq} + d_{qj})$ is returned as the distance d_{ij}. Here p and q are intermediate areas indexes. Algorithm 1 show the process and pseudo code we used to generate distance matrix. Note that, the bidirectional search is rerun many times until each distance converges.

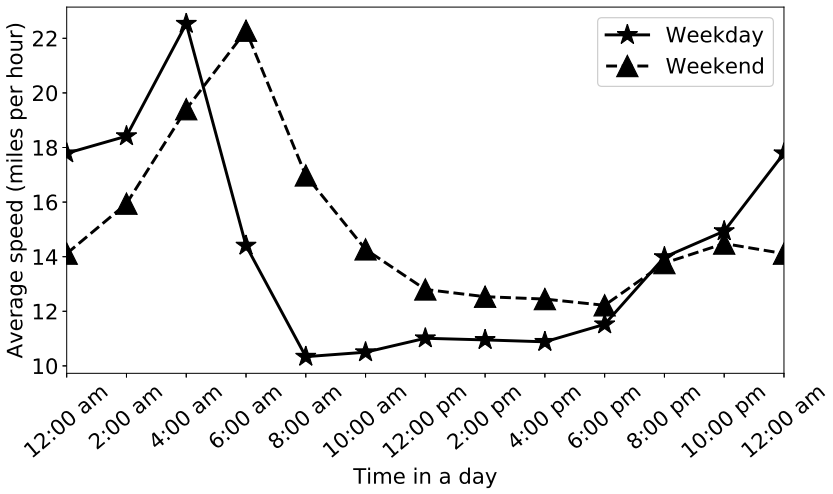

Figure 4.1: Different speeds on weekday and weekend.

In addition to the distance matrix, we also estimate the mean taxi speeds based on the historical dataset. Inspired by work [10, 80], we consider different taxi speeds in weekday and weekend, and different time periods in a day. Fig. 4.1 shows the mean taxi speed in each time period extracted from the historical dataset. With the distance matrix, for each taxi request received by the system, the corresponding travel time can be estimated.

4.2 Future demand and destination prediction

We use the prediction models described in Chapter 3 to predict future taxi demand and destination distribution in the city. In this application, the entire city is divided into about 1000 areas with Geohash [69] library. For each area, we encode taxi trips into specific sequential data structure for the learning models. We use the recurrent neural network and mixture density network (LSTM-MDN) structure to learn the taxi demand pattern at each area. Affecting factors such as date, time steps in the day, day of the week and weather

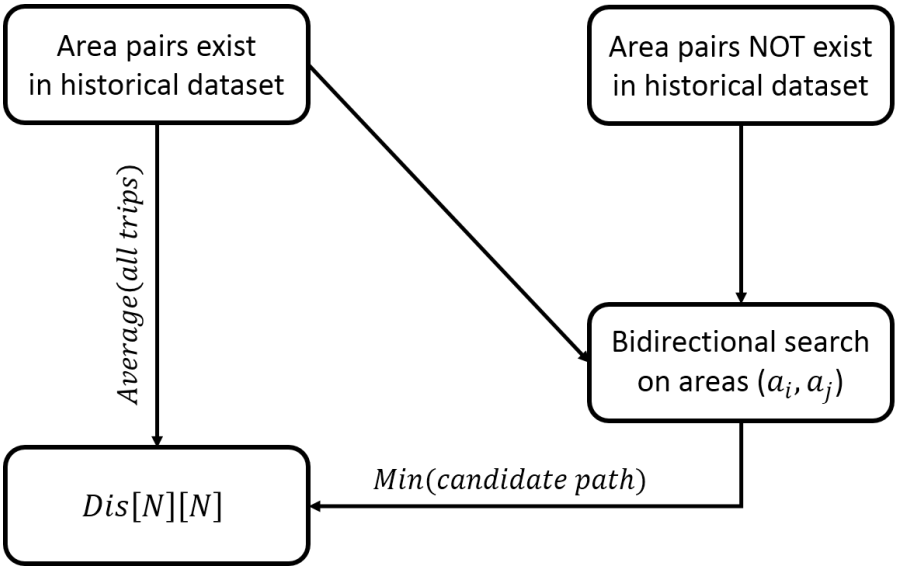

Figure 4.2: Distance matrix generation flow.

information are used to train the model.

For the destination prediction model, we treat it as one input multiple possible outputs problem. We use the feed-forward neural network and mixture density network (FN-MDN) structure to model the outputs distribution. Instead of learning and predicting a destination area directly, the proposed model learns the destination pattern and outputs a distribution over all the areas in the city.

We train both the demand and destination prediction models with one year taxi data in 2015 from the New York City taxi trip dataset [68]. We evaluate the performance of the prediction models and the dispatch system with taxi data in 2016 from the same dataset.

Algorithm 1: Distance matrix generation

```
all_areas, list of all areas
N = len(all_areas)
all_trips, grouped by trip start area
Distance matrix Dis = [N][N]
/* area pairs with trip records */
for (a_i, a_j) in all_areas do
    trips = get_trips(a_i, a_j, all_trips)
    Dis[i][j] = mean(trips.distance)
end
/* area pairs without trip records */
for (a_i, a_j) in all_areas do
    candidatepaths =
    bidirectionsearch(a_i, a_j, Dis)
    Dis[i][j] = min(candidatepaths)
end
Re-run until Dis converges
return Dis
```

4.3 Dispatch system

Similar to the current mobile app based taxi services, we consider a scenario where passengers can send real time taxi requests to the system with start and destination locations. Goals of our dispatch system are given as follows:

- Serve all the taxi requests.
- Minimize idle driving distances of taxis.
- Minimize passengers average waiting time (time between sending request and pickup).

Given the estimated distance matrix and taxi speeds, for each taxi request received by the server, the corresponding travel time can be estimated. We divide a day into time steps $\{0, 1, ..t, ..max\}$ such that t means the t^{th} time step in a day. We use 1 min as one time unit,

then $max = 24 \times 60$ in this case. Based on these settings, we build the dispatch system. For any area $a, a \in all\_areas$, we represent its taxi demand at time step t as $demand_t^a$:

$$demand_t^a = new\_demand_t^a + unassigned_t^a \tag{4.1}$$

where $new\_demand_t^a$ represents the newly received taxi requests in area a at time step t, while $unassigned_t^a$ represents the unassigned taxi requests from previous time steps.

Similarly, the available number of taxis is represented as $available_t^a$ which consists of 2 parts: the $idle_t^a$, represents the number of original idle taxis in area a at time step t, and the $arrival_t^a$ which represents the number of estimated arriving taxis (non-occupied) to area a at time step t.

$$available_t^a = idle_t^a + arrival_t^a \tag{4.2}$$

The work flow of the dispatch system is shown in Fig. 4.3. At time step t, there are two steps to be executed in the dispatch system. Step I, for each area a, we sort taxi requests $demand_t^a$ by receiving time, then a greedy assignment with available taxis $available_t^a$ is executed. There is a possibility that some requests cannot be served due to limited number of available taxis at the same time step. Since we sort all the taxi requests by receiving time, a high priority is given to earlier requests and they will be fulfilled in the next time step.

After taxi assignment in step I, we continue to do taxi dispatch in step II with the goal to balance future supply and demand ratio. With the prediction models in section 4.2, first we sample each area's future taxi requests and their destinations within a number of *lookahead* time steps. Then, we virtually simulate the taxi assignment (step I) for each area

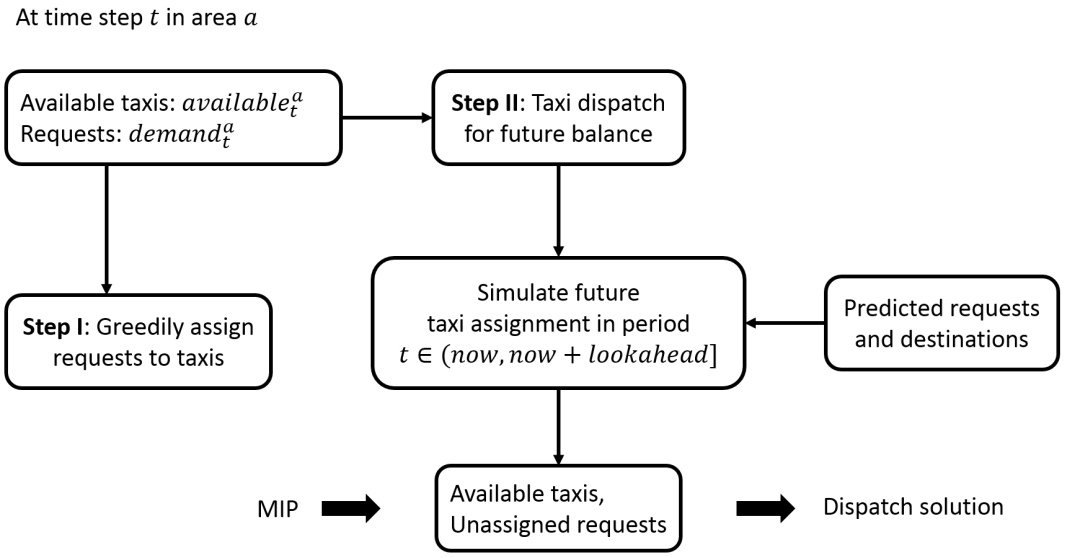

Figure 4.3: Destination distribution learning model.

in the next *lookahead* time steps. After this process, we obtain the remaining unassigned requests and available taxis in each area. A taxi dispatch process is conducted to balance the supply-demand curves in each area over the entire city. We optimize the dispatch strategy by solving a mixed integer programming (MIP). The objective of the MIP is to minimize the total driver's idle driving distances while serving all the coming requests. The details of dispatch process is shown in Algorithm 2. The taxi assignment process at each time step is shown in lines 7-10. For each area, we first assign available taxis to requests in a greedy fashion. Then, the unassigned requests together with updated available taxis are passed to the *dispatch* function for future rebalance. During the dispatch process (lines 14-26), the system first uses the predicted future demand and destination to model the greedy assignment according to future predictions. After that, all the remaining unassigned requests and the available taxis in the system are the targets to be matched. We optimize the matching by solving a MIP with the goal of minimizing the total idle driving distances. Finally the MIP solution is obtained and a real taxi dispatch is executed to improve the

future demand balance.

Algorithm 2: Taxi dispatch process

all_areas, list of all areas
Distance matrix Dis
Future rebalance time steps $lookahead$
Demand prediction model M_{demand}
Destination prediction model M_{dest}
for $t \in [begin, end]$ **do**
    $available_t = idle_t + arrival_t$
    /* Step I: greedy assign taxis to requests
    $unassigned_t = sort\_assign(demand_t, available_t)$
    $update(idle, arrival)$
    /* Step II: taxi dispatch for future balance
    $dispatch(t, unassigned_t, idle, arrival)$
end
Function `dispatch`$(t, unassigned, idle, arrival)$`:`
    $real\_available = idle + arrival$
    for $i \in (t, t + lookahead]$ **do**
        $\widehat{demand}_i \leftarrow M_{demand}.predict(i)$
        $\widehat{dest}_i \leftarrow M_{dest}.predict(i, \widehat{demand}_i)$
        $update(arrival) \leftarrow (\widehat{dest}_i, Dis)$
        $available_i = idle_i + arrival_i$
        $\widehat{demand}_i = \widehat{demand}_i + unassigned$
        $unassigned = sort\_assign(\widehat{demand}_i, available_i)$
        $update(idle, arrival)$
    end
    $solution = MIP(unassigned, real\_available)$
    $execute(solution)$
end

4.4 Dispatch system evaluation

In this section, we evaluate our taxi demand and destination prediction models as well as the performance of the dispatch system.

4.4.1 Experimental setup

We validate the performance of the proposed network model with the New York City taxi trip dataset [68]. There are two kinds of taxi cabs in NYC: the yellow cabs, which operate mostly in Manhattan, and the green cabs, which operate mostly in the suburbs. The dataset contains daily recorded taxi trips executed by more than 15000 taxis for the whole city. The total number of taxi trips varies everyday. We list one week data in 2016 as an example, from Monday to Sunday: $[374305, 395678, 408184, 432087, 453192, 480818, 418237]$. We train our taxi demand and destination prediction models with one year taxi data in 2015 and validate the prediction models as well as the dispatch system with taxi data in February 2016.

For the taxi demand prediction, we use the recurrent neural network and mixture density network structure (LSTM-MDN). We discretize the requests into time steps of 15 *mins* for each area in 2015. The input data shape is $(365 \times 96, 96, 997 + 10)$ in which 365×96 is the total number of time steps, 96 in the second dimension is the sequence length (one day or 24×4), 997 in the last dimension is number of areas in the whole city and 10 is the number of affecting factors including date, time step of the day, day of the week and weather type.

For the destination prediction, we use the feed-forward neural network and mixture density network structure (FN-MDN). But different from the demand prediction, the input to the destination model is each trip information, which consists of the start area of the trip and its affecting factors. The affecting factors are the same as the demand prediction, which includes date, time step of the day, day of the week and weather type.

For the dispatch system, we initialize the taxi distribution based on a sum of historical requests in each area. The demand and destination prediction are conducted by the proposed LSTM-MDN model and the FN-MDN model respectively. Distance matrix Dis and different

taxi speeds in weekday and weekend, see Fig. 4.1, are used to estimate the travel time between areas in the city. The time unit in the dispatch system is 1 min, the predicted taxi demand is averaged into 1 min scale. Table 4.1 includes the list of parameters in the experiments.

Table 4.1: Experimental parameters

Area/grid size	$\leq 1.2km \times 0.61km$
Number of areas	997
Dispatch system time unit	1 min
Number of layers for neural networks	2
Number of Gaussian kernels	Hyper-parameter

4.4.2 Performance metrics

Demand prediction

For demand prediction, we use error metric symmetric Mean Absolute Percentage Error (sMAPE) to evaluate the prediction performance over the entire city [1]. From the statistic perspective, sMAPE describes a percentile prediction error and can be defined as follows:

$$sMAPE_t = \frac{1}{N} \sum_{n=1}^{N} \frac{|Y_{n,t} - \widehat{Y}_{n,t}|}{Y_{n,t} + \widehat{Y}_{n,t} + c} \tag{4.3}$$

Herein t is the t^{th} time step in a day and N is the total number of areas in the city. $Y_{n,t}$ represents the real taxi demand in area a_n at time-step t while $\widehat{Y}_{n,t}$ is the predicted taxi demand. The constant c in Eq. 4.3 is a small number ($c = 1$ in this application) to avoid division by zero when both $Y_{n,t}$ and $\widehat{Y}_{n,t}$ are 0.

Destination prediction

For the destination prediction, we show the classification accuracy by using the number of correct predictions divided by the total number of requests at time step t.

$$Accuracy_t = \frac{Correct_t}{Correct_t + Incorrect_t} \tag{4.4}$$

Dispatch system

For the dispatch system, we show the performance in terms of two metrics:

- Taxi passengers' average waiting time.
- Taxi drivers' average idle driving distance.

Here, passengers' waiting time means the time between sending out the request and being picked-up. The drivers' idle driving distance means the distances taxi run in non-occupied mode, while serving all the requests in the city. In addition, both of the performance metrics are shown with different values of hyper parameters: future dispatch *lookahead* time steps and total number of taxis in the city.

4.4.3 Proposed models & systems

Demand prediction

- *Moving mean.* This approach predicts taxi demand based on the mean value of past demands in a sliding-window. For example, if it is to predict taxi demand at 10:00

am on the next Monday, the predicted result would be the mean value of demands at 10:00 am in the previous 5 Mondays.

- *LSTM*. This approach uses Long Short Term Memory (LSTM) recurrent neural network with a mean square error (MSE) minimization strategy to learn the sequence patterns of taxi demand from historical data. Then, the learned patterns will be used to make predictions.

- *LSTM-MDN*. This approach uses LSTM and mixture density networks (MDNs) to learn the taxi demand distribution from historical data. Then, the learned distribution will be used to make predictions.

Destination prediction

- *Moving sampling*. This approach uses a frequency distribution in a sliding-window to provide a destination distribution. For example, if it is to predict destination of taxi trips start from area n at 10:00 am on the next Monday, the estimated destination distribution would be a frequency distribution of all destinations start from area n at 10:00 am in past 5 Mondays. Then, this normalized distribution is used for sampling future destinations.

- *FN-MDN*. This approach uses a feed-forward neural network and MDNs to learn destination distribution from historical data. Then, the learned model can predict distribution for each area, which is used for sampling future destinations.

- *FN-MDN-neighbors*. In the FN-MDN approach, the predicted results are sampled from a continuous distribution. FN-MDN-neighbors is the same approach as the FN-MDN but accepting neighboring areas as correct predictions. Note that predicting

neighboring areas as destination is still acceptable and useful for our taxi dispatch application since the taxi is very close to the target area.

Dispatch system

According to different demand and destination prediction models used in the system, we propose 5 dispatch systems to compare their performances. Note that, *FN-MDN-neighbors* is not included since it is the same prediction approach as *FN-MDN* but different evaluation method. All dispatch systems use the same dispatch strategy shown in Fig. 4.3. The details of each dispatch system is shown in Table 4.2.

Table 4.2: Dispatch systems

Dispatch system	Demand prediction	Destination prediction
System I	Moving mean	Moving sampling
System II	LSTM	Moving sampling
System III	LSTM	FN-MDN
System IV	LSTM-MDN	Moving sampling
System V	LSTM-MDN	FN-MDN

4.4.4 Performance results

First, we report demand prediction error sMAPE and destination prediction accuracy in the entire city. Second, we show the impact of using different number of Gaussian kernels in the prediction models. Lastly, we present performance metrics that characterize the dispatch system.

Prediction performance

For the demand prediction models based on *Moving mean*, *LSTM* and *LSTM-MDN*, we

show their performance in terms of the metric formulated in Eq. 4.3. For the destination prediction models based on *Moving sampling*, *FN-MDN* and *FN-MDN-Neighbors*, we show their performance in terms of the metric formulated in Eq. 4.4.

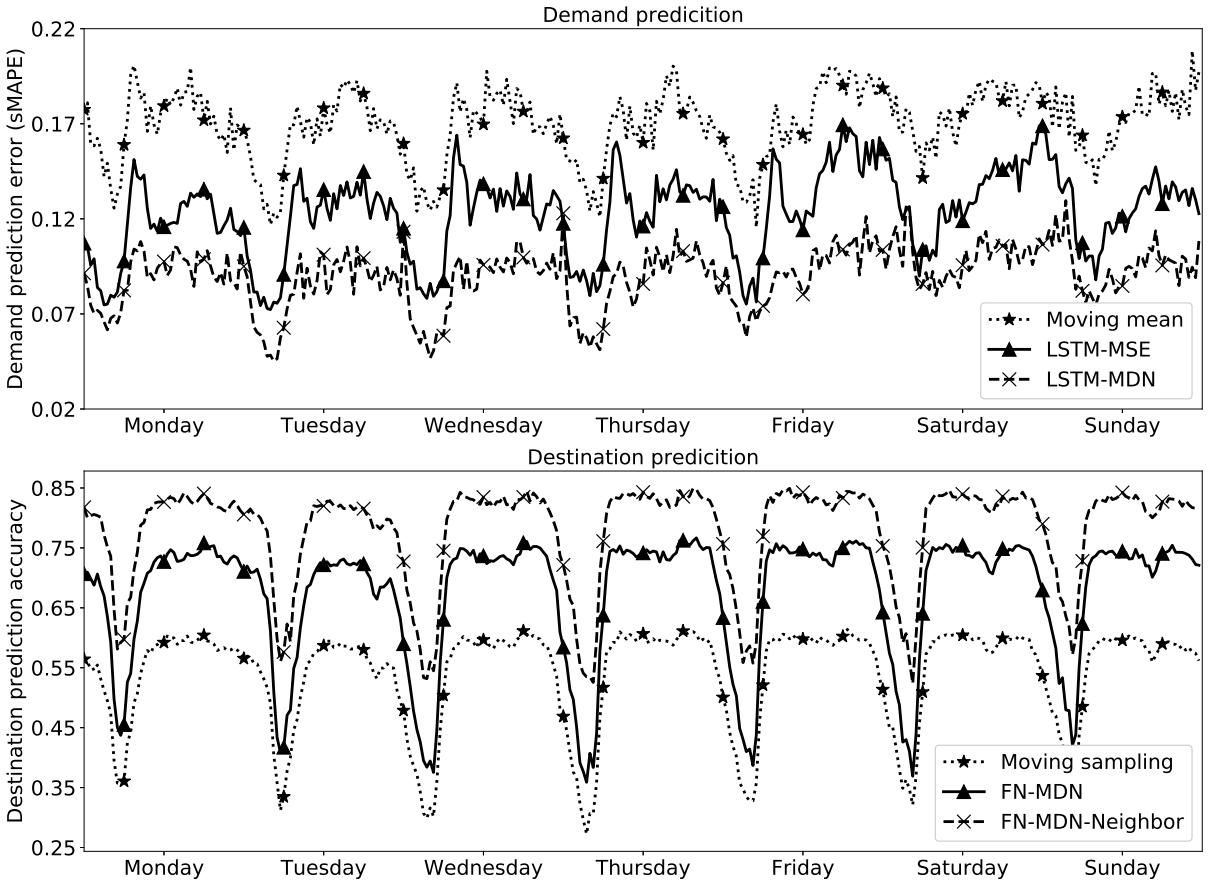

Figure 4.4: Demand prediction and destination prediction in one week by different prediction models.

Fig. 4.4 shows the prediction performance in one week by different demand and destination prediction models. In Fig. 4.4-*demand prediction*, the proposed LSTM-MDN model always outperforms the LSTM model. This proves that distribution learning is better than the MSE minimization when dealing with real-valued data prediction. In addition, both the LSTM-based models outperform the Moving mean model. Especially during rush hours, for

instance, at around 8:00 am in the morning, the LSTM-based models still perform very well while the Moving mean model becomes much worse. This is because the LSTM takes into account long term dependencies of past information while predicting the future.

Fig. 4.4-*destination prediction* shows the performance of the destination prediction models. It can be seen that Gaussian mixture models, i.e., the FN-MDN and FN-MDN-Neighbors models, perform much better than the Moving sampling model. As we discussed before, if we accept neighboring areas as correct predictions, we achieve better results as *FN-MDN-Neighbors*. On the other hand, considering both figures in Fig. 4.4, we observe an interesting result that both of them have a small value at around 4:00 am. This however means that demand prediction achieves the best performance (lowest error) while destination prediction hits its worst accuracy during that time period. The reason could be that the total number of requests is low and mainly from some busy areas such as airports and bars, but the corresponding destinations can be anywhere around the town and are not that predictable.

Importance of Gaussian kernels

To further showing the importance of using mixture density networks (MDNs) for distribution learning, we report the performance of using different number of Gaussian kernels in our prediction models. As we discussed in Chapter 3, the idea of MDNs is to use the outputs of a neural network to parameterize a mixture distribution. Given enough number of Gaussian kernels, theoretically we can model any data distributions. The MSE minimization approach can be regarded as similar to the special case of the mixture Gaussian model when $M = 1$ (but not exactly the same).

Fig. 4.5 reports the performance when using different number of Gaussian kernels in the demand prediction model. As shown in Fig. 4.5, when the number of Gaussian kernels increases, the prediction error gradually decreases, i.e., the prediction performance improves

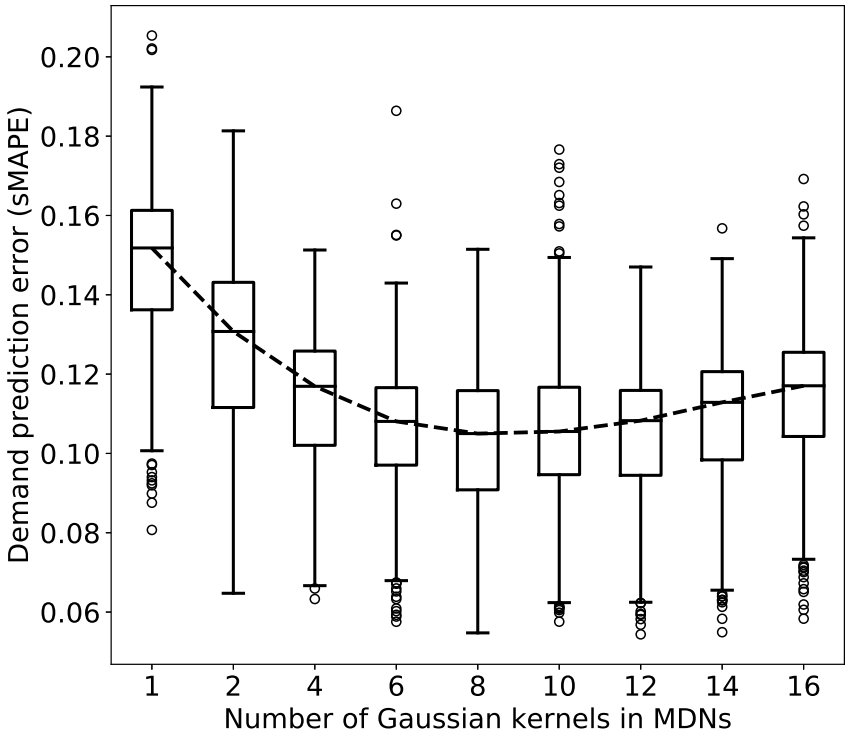

Figure 4.5: Number of Gaussian kernels in MDNs for demand prediction.

when using more Gaussian kernels. This again proves that using distribution modeling can further improve the prediction performance when dealing with real-valued data. However, because it is a recurrent model, every added Gaussian kernel will obviously increase the model size. When the number is big, this recurrent model is hard to be trained. This explains why the performance decreases (error increases) gradually when the number increases.

Fig. 4.6 reports the performance when using different number of Gaussian kernels in the destination prediction model. Different from the recurrent model in demand prediction, this feed-forward model allows us using more Gaussian kernels without worrying about the model size. As shown in Fig. 4.6, the prediction performance improves as the number of Gaussian kernels increases and becomes stable after using more than 15 Gaussian kernels.

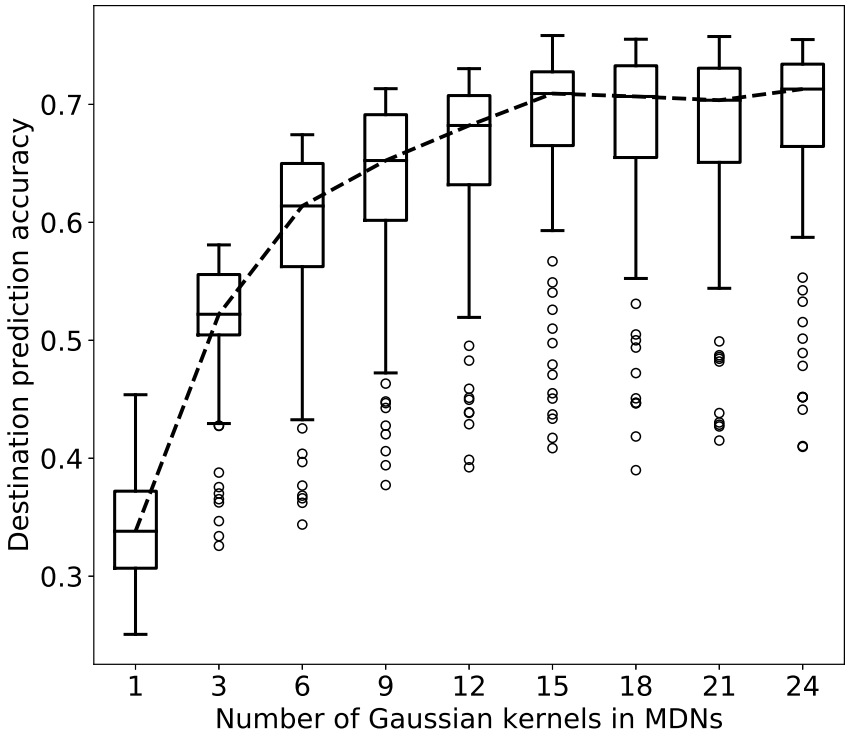

Figure 4.6: Number of Gaussian kernels in MDNs for destination prediction.

Dispatch system performance

According to different demand and destination prediction models used in the system, we propose 5 different dispatch systems. The details of each system are shown in Table 4.2. For each dispatch system, we show its performance in terms of two metrics: taxi passengers' average waiting time and taxi drivers' average idle driving distance.

Fig. 4.7 shows the error bars with standard deviation for the two performance metrics on different future dispatch *lookahead* time steps. As we can see in Fig. 4.7-a, by increasing the *lookahead* time duration, shorter average waiting time for passengers can be obtained. When there is no future dispatch ($None$) at all, all dispatch systems become the same and result in the same performance. Large standard deviation can be seen if there is no future

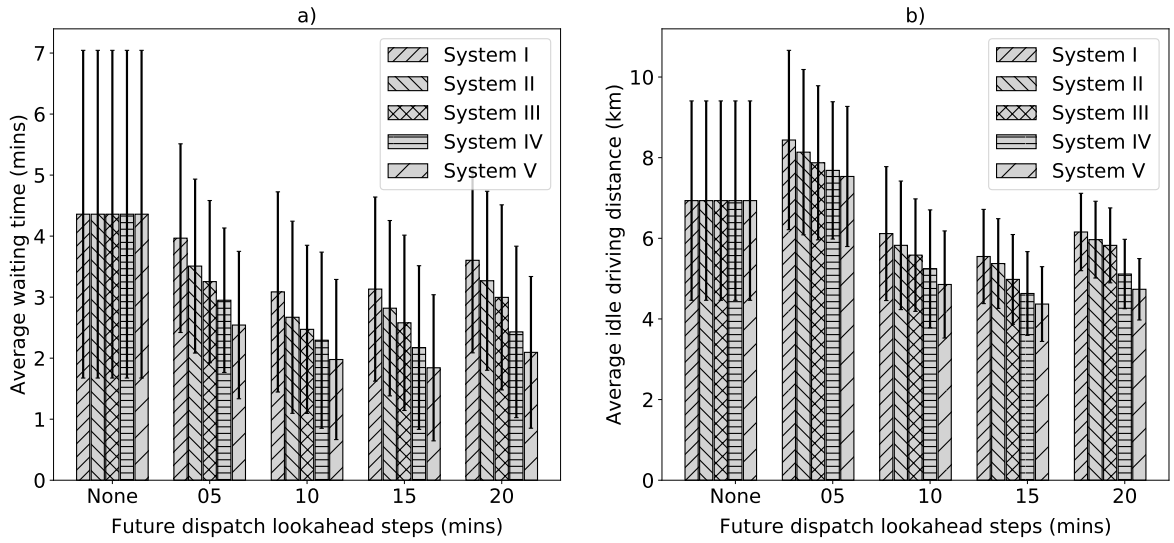

Figure 4.7: Performance of passengers average waiting time, taxi average idle driving distance on different *lookahead* time duration. Settings of each dispatch system: *System I: Moving mean + Moving sampling*, *System II: LSTM + Moving sampling*, *System III: LSTM + FN-MDN*, *System IV: LSTM-MDN + Moving sampling* and *System V: LSTM-MDN + FN-MDN*. Results obtained via using total number of 4000 taxis in the city.

dispatch. The reason is that some requests are assigned to taxis in far areas due to no taxi availability nearby. Besides, we observe that when the *lookahead* time is bigger than 10 *min*, the average waiting time on each system gradually increase. Let us remember that each prediction model contains prediction error that accumulates when *lookahead* time increases. This also explains why the values in *System IV* and *System V* continue to decrease at $lookahead = 15$ due to their better prediction performance.

In Fig. 4.7-b, compared to no future dispatch (*None*), a slight increase on the average of taxi idle driving distance is observed when first introducing future dispatch mechanism ($lookahead = 5$). The reason could be that introducing future dispatch results in some idle driving distances. The benefit of future dispatch at $lookahead = 5$ is small and cannot overcome such a disadvantage. This is reasonable and note that it also gradually decreases

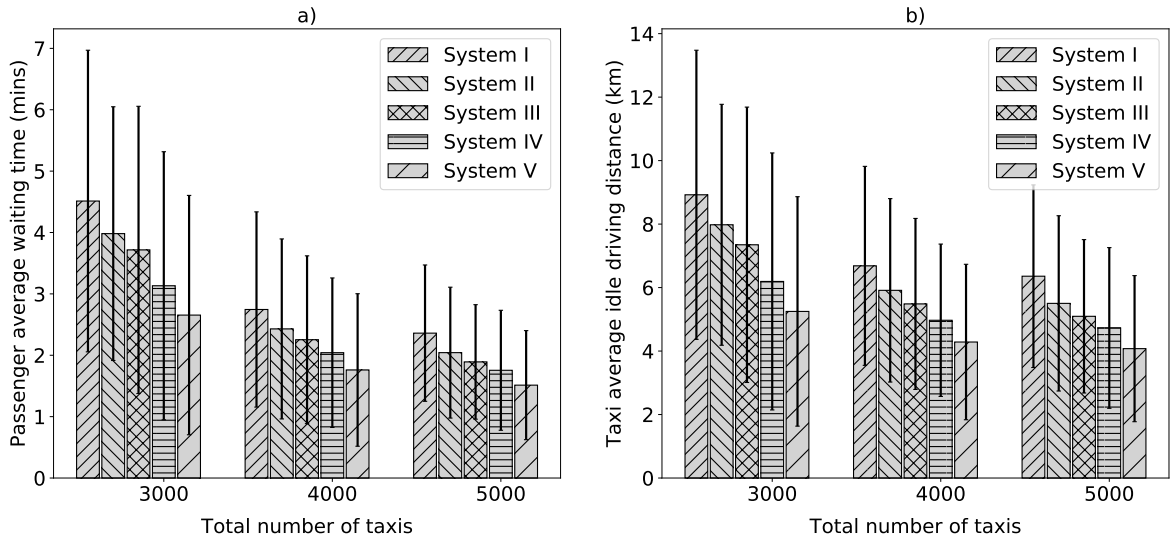

Figure 4.8: Performance of passengers average waiting time, taxi average idle driving distance on different total number of taxis in the city. Settings of each dispatch system: *System I: Moving mean + Moving sampling*, *System II: LSTM + Moving sampling*, *System III: LSTM + FN-MDN*, *System IV: LSTM-MDN + Moving sampling* and *System V: LSTM-MDN + FN-MDN*. Results obtained via using $lookahead = 15$ in each system.

to a level less than the $lookahead = None$. This is because the result of the future dispatch avoids most long distance pickups.

Fig. 4.8 shows the error bars with standard deviation for the two performance metrics on different total number of taxis in the city. When providing more number of taxis, obvious improvement on passengers' average waiting time and drivers' average idle driving distance can be achieved. Similar result patterns are observed in both Fig. 4.8-a and b. This is reasonable because more taxis in the city enables more available taxis to be pre-dispatched to different areas in the city. On the other hand, only increasing total number of taxis in the city cannot effectively solve the taxi supply-demand ratio.

Lastly, we show the real-time number of running taxis in the system throughout a day in

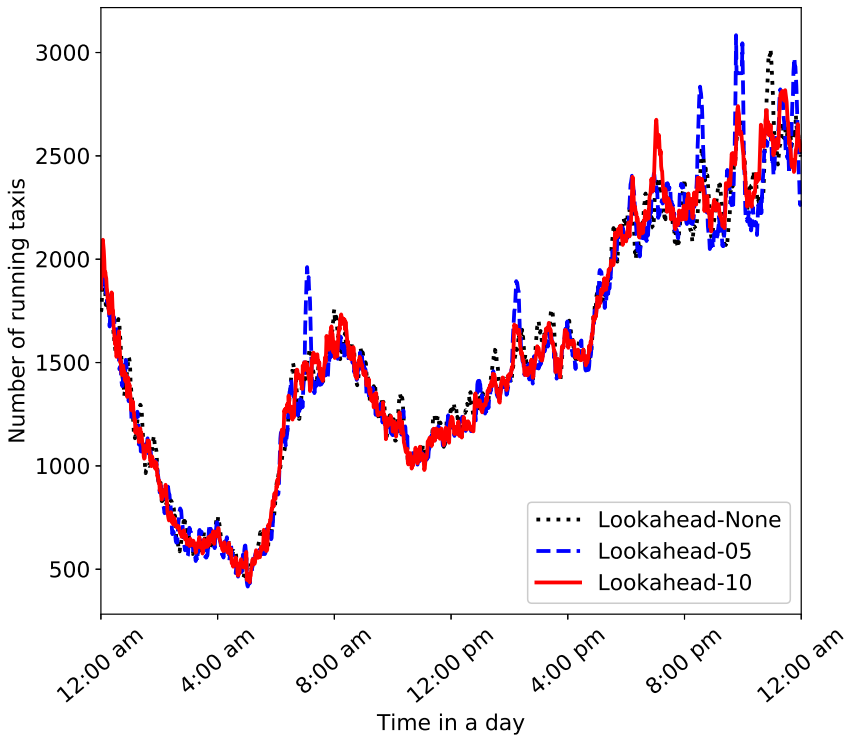

Figure 4.9: Number of real-time running taxis in the system throughout a day. The total number of taxis in the system is 4000 for each of the method.

Fig. 4.9. We do not see a big difference among them except a slight increase in busy hours when introducing future rebalance. The reason could be that the number of running taxis in the system highly relies on the real-time demand while only a small number of taxis are involved in the future rebalance.

Overall, based on the experimental results, a better dispatch performance could be obtained after introducing taxi future dispatch mechanism. Let us note that as the *lookahead* time duration grows, the dispatch performance slightly decreases due to prediction errors accumulation.

CHAPTER 5: REINFORCEMENT LEARNING BASED PATH PLANNING FOR ANIMAL MONITORING APPLICATIONS

In this chapter, we focus on the animal monitoring and tracking in large wild areas. We propose a Markov decision process (MDP) based path planning strategy for an unmanned aerial vehicle (UAV) under animal monitoring application scenario [59, 81].

Tracking the movement of wild animals allows us to identify the areas most critical for conservation and can reveal the effects of climate change, the presence of humans and non-native species. In an unknown large area, it is difficult and sometimes infeasible to find wildlife animals to attach wearable tracking devices. In this case, low cost sensors can be widely deployed throughout the observation area to detect and recognize the wildlife appearance. Currently, sensors can identify animals by different types of inputs such as smell, sound and image. With deployed sensors, we can get animal appearance information in wildlife areas, however, long distance wireless communication in large remote wildlife areas is costly and impractical. In general, these sensors communicate locally to accumulate their readings at a clusterhead node. The data is periodically collected by a visiting unmanned air vehicle (UAV). The path and schedule of the UAV critically impacts the timeliness of the tracking of the animals' movement.

By observing animals, we find that animal activities usually have some specific features such as living in groups and having more activities around habitats. These features allow animal movements to have some specific patterns. Examples of animal movement trajectories can be seen in Fig. 5.1, instead of having activities in a large area, animals usually have activities only in one or several small areas. If we focus more on these "hot" areas, we have a much higher probability to track and predict the movements of the animal. Inspired by this idea,

we divide the observation area into small virtual grids with sensors. Based on this grid structure network, we design our system model by utilizing a UAV to explore and learn these "hot" small virtual grids. By treating each grid as a cluster of sensor nodes, data collection by the UAV is reduced to visiting the clusterhead of each grid.

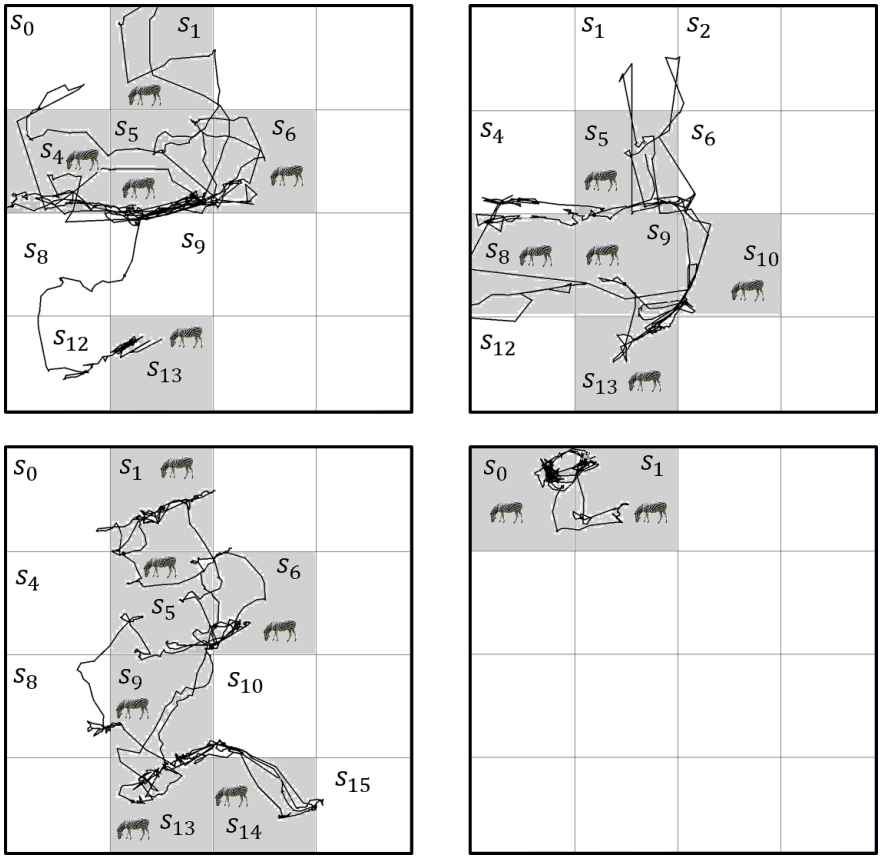

Figure 5.1: The movement trajectories of 4 zebras over 3 days from the ZebraNet dataset. The grid cells that contribute most of the VoI to the sensor readings are colored gray.

We propose to use Markov decision process (MDP) model to do path planning for the UAV as each grid represents a state of the MDP. We solve the MDP model using Q-learning algorithm by letting UAV to receive rewards from grids when animal data is reported. The goal of our path planning approach is to collect animal data efficiently and then predict future animal movements. The UAV is self-learning and can dynamically plan and adapt

according to environment changes.

To quantitatively weigh the value of the sensed animal appearance information, the proposed path planning approach utilizes the value of information (VoI) [26]. The basic idea of VoI is that the sensed information has the highest value when it is first generated and the value decreases as time moves on. In such a way, the task of the UAV can be reduced to maximizing the overall value of information obtained from the entire network. We give a mathematical model for calculating the VoI in animal monitoring operation. Lastly, we evaluate the performance of the proposed network model and the path planning approaches with real animal datasets: ZebraNet [48] and vultures [82].

5.1 Network model

Considering a large observation area with animals living in it, the goal is to monitor specific animals' appearance. First, a set of sensor nodes with monitoring functionality are deployed and one UAV is assigned to collect data from the deployed sensors. One option for the UAV to collect data is to visit each sensor directly. However, visiting each sensor node becomes impractical when the number of sensors is large. So, we divide the area into virtual grids and select one clusterhead for each grid cell.

5.1.1 Sensor nodes

In our scenario, sensor nodes are deployed by uniform random distribution in all parts of the observation area. As illustrated in Fig. 5.2, sensors in a grid can be treated as a cluster and a clusterhead is selected periodically. Sensors inside a virtual grid can communicate with the clusterhead directly or via a hop-by-hop communication. Clusterhead is responsible for

receiving event messages from the sensors and then submitting all the event messages to the UAV when it comes. While routing and clustering are not in the scope of this paper, they are well-investigated problems and there exist various efficient mechanisms [35] [83]. For instance, Hamidreza et al. [35] propose a clusterhead (rendezvous points) selection method by jointly considering node degree and hop distance to minimize energy consumption and improve load balancing. They also use virtual rendezvous points to increase the performance. In our application, clusterhead selection is based on an energy balancing policy similar to Shu and Kuntz [83].

We assume that there is no long distance networking infrastructure available, thus there is no communication between the clusterheads. The clusterhead continuously collects and buffers the data collected from the individual nodes. This data is collected by a UAV that periodically visits the grid cell and downloads the buffered data from the clusterhead using short-distance point-to-point communication.

5.1.2 Unmanned Aerial Vehicle

UAVs have been widely used in various applications due to their several advantages such as flexibility, high speed and good endurance. In this WSN application, UAV is used as an autonomous mobile sink for gathering time sensitive information. Apart from previously mentioned advantages, using UAV for animal monitoring not only overcomes the geographical challenge but also has minimal effects on animals. Assumptions of the network model are given as follows:

- There is a single UAV as it is the costly element of the network.

- The UAV has no energy constraints, while it is not the case for the sensor nodes.

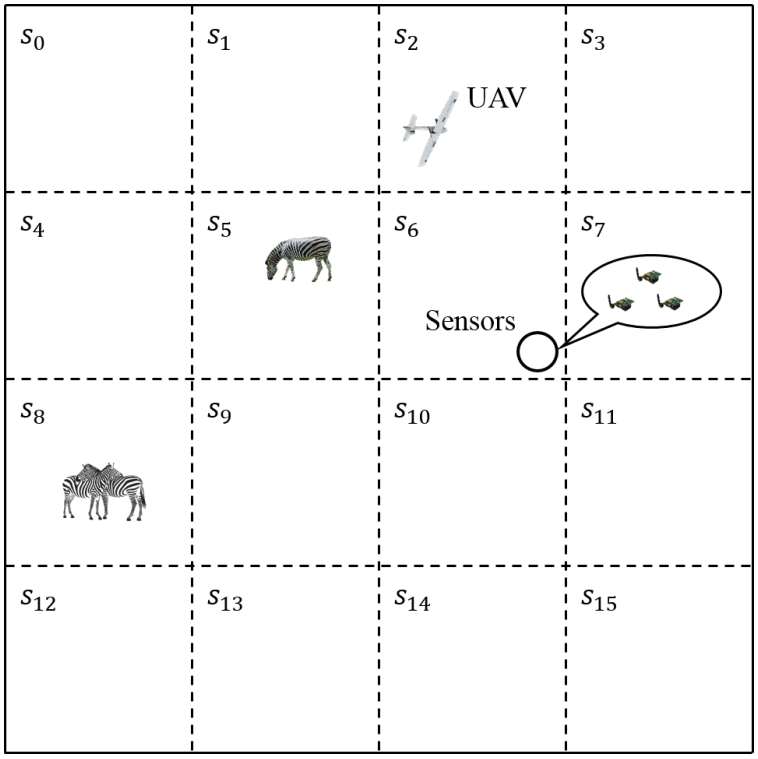

Figure 5.2: The area of interest, divided into a rectangular grid. Each grid cell s_i contains a cluster of sensors with a clusterhead that collects the recorded data. The UAV periodically visits the grid cells.

- The UAV flies with a fixed speed and it only communicates with clusterheads.

5.2 Performance Metric

In this section, we describe the performance metric used to evaluate the proposed animal monitoring model.

5.2.1 Value of information

How do we compare two animal monitoring systems? The final goal of the user is the timely tracking of the location of animals. In practice, the user will receive sensor observations as a stream of chunks of information $D_1, D_2, \ldots$ at time points $t_1, t_2, \ldots$. Intuitively, not all the chunks of the information have the same *value*; for instance, we are more interested in information about the presence of the animals than their absence. Earlier the information chunk is received, the more valuable it is.

We thus argue that the total *value of information* (VoI) collected by the user is a suitable performance metric for an animal monitoring system. We contrast this with low-level metrics such as total sensing bandwidth or transmission latency, which do not adequately capture the intent of the user.

The total VoI collected by the UAV clearly depends on the path of the UAV which determines the order in which the data chunks are collected and the time when they are received by the user. We are thus interested in a UAV path that maximizes this metric[1].

In order to make this a formalizable optimization problem, we need to find a mathematical representation of the VoI that captures our intuition. The concept of VoI had been introduced in game theory [84] where it is associated with the value an optimal player would pay for a certain piece of information. Thus value, in contrast to quantity, is not a property of the information chunk itself, but it is determined by the specific context and the degree to which the information supports the intended actions of the user [26]. For instance, the second identical copy of an animal sighting reaching the user will have zero or maybe a very low confirmation value, despite it being identical to the valuable first copy that conveyed a

[1]Naturally, the total VoI can be also increased by dedicating additional hardware resources to the problem (more sensors, more and faster UAVs and so on).

novel information. Another aspect of VoI is its *urgency*. If the data is intended to be used long time after its collection, for statistical purposes, the chunk of information has the same value if it is collected one minute after its sensing or one week later. If, however, the data is used to localize a threatened animal for an intervention, the sooner the information chunk reaches the user, the higher the VoI is. In some applications there is an expiration threshold: information chunks received after that have zero value.

Finally, there is a certain initial value of information $V_0(D)$ that describes the VoI assuming that the information chunk reaches the user immediately after sensing. This value depends on the nature of the event captured in the data chunk as well as the type and accuracy of the sensor – for instance, an image is worth more than a pressure reading.

For the purposes of this dissertation, we will make the following assumptions about the VoI model:

- The VoI is *additive over sensors and clusters of sensors*. The VoI of the cluster is the sum of the VoIs of the reporting sensors, and the total VoI is the sum of the VoI accumulated over the path of the UAV. This assumption states that there are no redundancies across the clusters, and whatever redundancies exist within the cluster sensors, they will be eliminated by the clusterhead before forwarding the data.

- The *initial VoI of a data chunk only depends on the presence of the animals* in the specific grid cell. If there are no animals in the grid cells, no value is accumulated. This implies that we have a homogeneous set of sensors in each cluster, and we don't consider the knowledge of the presence of animals in one grid cell to be more valuable than in other grid cells. This restriction can be eliminated without changes to the algorithm.

- The VoI of a specific information chunk D exhibits an *exponential decay*

$$\mathcal{V}(t) = \mathcal{V}_0 \cdot e^{-Bt} \tag{5.1}$$

The decay constant B depends on the application, but does not vary from sensor to sensor or observation to observation. Exponential decay is a widely used model in many similar problems, it offers the benefit of having a convenient mathematical form, and of being continuous and differentiable throughout its domain. Fig. 5.3 shows the VoI curve for three different combinations of V_0 and B values.

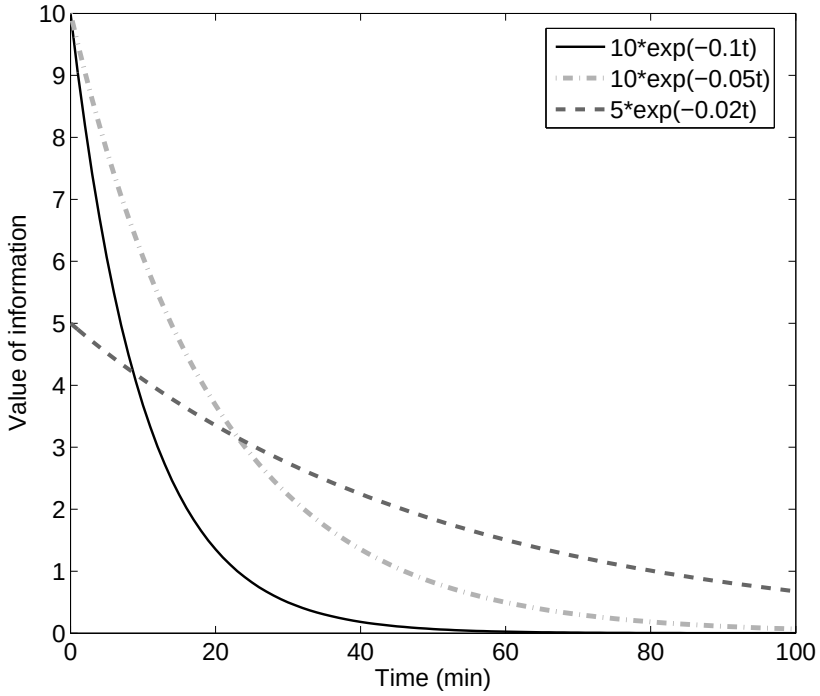

Figure 5.3: The temporal evolution of VoI for three events with different initial value $\mathcal{V}_0$ and B parameters.

With these assumptions, we are now ready to describe the mathematical form of the VoI.

Let us consider that the UAV starts from a location S at time $t = 0$ and follows the path $(S, 0), (s_{p_1}, t_{p_1}), \ldots (s_{p_i}, t_{p_i}) \ldots$, that is, the UAV visits grid cell s_{p_i} at time t_{p_i}. Let us denote with $t_{p_i}^{prev}$ the time of the previous visit of the UAV at the same grid cell. The VoI expression will be:

$$\mathcal{V}_{total} = \sum_{i} \sum_{c \in C(s_i, t_{p_i}^{prev}, t_{p_i})} \mathcal{V}_0 \cdot e^{-B(t_{p_i} - t_c)} \tag{5.2}$$

where $C(s, t_{start}, t_{end})$ is the set of information chunks (events) collected in grid cell s in the time interval $(t_{start}, t_{end}]$ and t_c is the collection time of event c.

5.3 Path planning algorithms for a VoI aware animal monitoring system

MDP model and reinforcement learning baseline

Let us now consider the problem of choosing a path for the UAV such that the collected VoI is maximized over time. A simple choice would be a periodic path where all grid cells are visited at equal intervals. By optimizing the length of such a path (e.g. by choosing it to be the solution of the traveling salesman problem), we can minimize the interval between the regular UAV visits. This would be the optimal choice if the animals are distributed uniformly in the area, and thus each cell contributes the same VoI in a given time interval. However, most of the animals of interest are large mammals, usually operating in groups that are likely to stay in a certain location, roam around in a small area and revisit the same area multiple times.

In these scenarios, the majority of the VoI is provided by a small number of cells which are currently favored by the animals (see Fig. 5.1). Thus, a path that visits these cells more often would collect a higher VoI. Unfortunately, at deployment time we don't know

the grid cells favored by the animals; furthermore, this distribution is not stable in time. In conclusion, we will need to *learn* the current distribution of the animals for a more efficient monitoring, while we also need to maintain an *exploration strategy* to discover changes to this distribution.

To model this scenario, we use a Markov Decision Process (MDP)[85], a standard tool in modeling decision problems in the presence of uncertainty. An MDP is described as the combination of a set of states $S = \{s_1 \dots s_m\}$, a set of actions $A = \{a_1 \dots a_k\}$, a probabilistic transition function $T(s, a, s')$ describing the probability that taking action a in state s leads to state s', and a probabilistic reward function $R(s)$ describing the distribution of the reward. The agent of the MDP is assumed to take actions based on a policy $\Pi(s) \to a$.

Solving an MDP usually means finding a policy function Π^*, which maximizes either expectation of the sum of rewards $\mathbb{E}(\sum_i r_i)$ or the expectation of the sum of discounted rewards $\mathbb{E}(\sum_i \gamma^i r_i)$, where $\gamma \in (0, 1]$ is a discount factor that favors earlier rewards compared to later ones.

Let us now consider how this model can be applied to our scenario. The active agent of the MDP is the UAV. We say that the state of the MDP is s_i if the UAV is currently in grid cell i. The actions of the UAV can be $A = \{north, north - east, east, \dots, north - west, hover\}$.

We assume that the clusterhead organizes the sensor readings about the animals in the form of *events* e observing the following rules. It generates an event only after the presence of the animals had been confirmed for at least t^e_{min} time - this ensures that only confirmed sightings are reported. If the animals remain in the area, a new event will be generated only after an interval t^e_{int}, limiting the amount of traffic generated by a static situation.

The reward is the VoI collected by the UAV from the current cell according to Formula 5.2.

$$R(s) = \begin{cases} \sum_{i \in events} \mathcal{V}_i & \text{if animals are present} \\ R_{penalty} & \text{no animals present} \end{cases} \tag{5.3}$$

Our goal is to find a policy $\Pi_{UAV}(s_i) \rightarrow a_i$, in which any grid cell can tell the UAV where it should move next. If the transition function $T(s, a, s')$ and the reward function $R(s)$ are known, the optimal policy can be found using the well known techniques of value iteration or policy iteration. However, in our scenario, the reward function $R(s)$ is unknown at the beginning of the scenario. One solution for finding a policy in such a setting is *reinforcement learning* (RL): the agent goes out to the real world, performs a series of actions and updates the policy using the rewards actually collected. Q-learning [86] is an influential RL algorithm based on updating the value $Q(s, a)$ that describes the expectation of the future discounted reward if the agent is currently in state s, the first action it takes is a and follows an optimal policy after that. The optimal policy can be extracted from the Q values by choosing $\Pi(s) = \arg\max_a Q(s, a)$.

Q-learning starts with a randomly initialized Q values. The agent follows a policy dictated by the current value of Q, it encounters various rewards and at every step updates the Q value through the formula:

$$Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') \right) \tag{5.4}$$

where α is the learning rate.

This formula was used in our previous work [59]. In the following, we describe how a better performance can be obtained by building upon the generic Q-learning model.

Episode-based Q-value update

The MDP describing the animal monitoring problem has certain peculiarities that can be exploited to increase the performance compared to the baseline Q-learning. In many RL problems the main source of difficulty is the transition function T, while the reward function is either static or is stochastic with a static distribution. In the case of our problem, the transition function is usually well-behaved: there is little uncertainty about the fact that if the UAV would want to move to the grid cell to the north, it will actually reach it. On the other hand, the reward function is not only stochastic, but its distribution can change in the long run with the change of the preferences of the monitored animals.

We can exploit the fact that the transition behavior is well behaved by adapting a technique through which we are propagating the reward to learn the Q-value not only of the current grid cell, but of the previous grid cells on the trajectory of the current episode as well, an idea inspired from Monte Carlo reinforcement learning [87]. When the UAV gets a reward from a state, instead of just updating the Q-value of the previous state, we update Q-values of the past several states. Let $\{(s_0, a_0), (s_1, a_1), \dots (s_n, a_n)\}$ be the state-action pairs of the current episode. The Q-value of the current state will be updated as in Equation 5.4, while for the previous states the update formula will be:

$$Q(s_{n-k}, a_{a-k}) \leftarrow (1 - \alpha^k) \cdot Q(s_{n-k}, a_{a-k}) + \alpha^k \cdot \left(R(s_n) + \gamma \max_{a'} Q(s_n, a')\right) \tag{5.5}$$

Fig 5.4 illustrates this process, while Algorithm 3 shows the recursive implementation we used. Note that the learning rate decays exponentially with the distance to the current state. Thus the rewards have more impact on recent states and less on the earlier states.

Algorithm 3: Updating the Q-values on an episode

Function `Q-UPDATE`$(s_n, R(s_n), N)$**:**
  $k \leftarrow 0$
  $s \leftarrow s_n$
  while $k < N$ **do**
    $(s, a) \leftarrow s.prev()$
    $Q(s, a) \xleftarrow{\alpha^k} R(s_n) + \gamma \max_{a'} Q(s_n, a')$
    $k \leftarrow k + 1$
  end
end

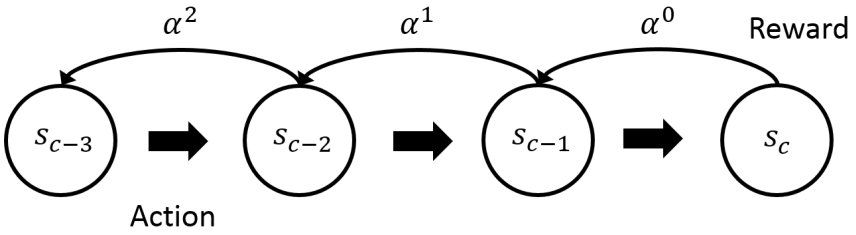

Figure 5.4: Updating the values.

Permanent exploration

Reinforcement learning is useful when there are interesting patterns in the reward function that we can learn. The distribution of animal locations in Fig. 5.1 shows that there are useful patterns to be learned over medium-term (of about a week). Over the long run, however, these patterns would change and their average will asymptotically approach a uniform coverage of the area. Taking advantage of the fact that the UAV moves faster than the change in the animal behavior, we can benefit from learning this medium-term pattern, but we also need to adapt the policy with the animal behavior change.

One way to accomplish this adaptation is by repurposing exploration, a well known technique in RL. The learning agent usually follows the current-best policy during learning. However,

this might confine the agent to certain areas of state space, without learning about potentially high rewards available elsewhere. To encourage the agent to fully explore the state space, it is a common technique to take actions different from the one recommended by the current-best policy. Finding the right balance between exploration and the exploitation of the current-best policy is a fundamental theoretical problem. In practice, we often start with more exploration and gradually shift the balance to exploitation, with the final policy being exploitation only.

We propose a *permanent exploration* technique, where the agent can maintain a degree of exploration that allows it to adapt the policy to the changes of the reward. We designed and investigated two variations of this strategy.

The *Permanent Exploration Q-learning ε-greedy* (PEQL- ε) algorithm combines the episode-based update with the ε-greedy technique of balancing the exploration and exploitation in reinforcement learning. In contrast to the usual practice, the $\varepsilon \in [0, 1]$ value is a static parameter of the algorithm which does not change during its operation (See Algorithm 4). At every step, the algorithm generates a random number and compares it with the ε value. If the number is larger than ε, the algorithm follows the current optimal policy induced by the Q values, otherwise it takes a random action from the ones feasible in the current state. The UAV will take a completely random action if $\varepsilon = 1$ while deterministically choosing the best possible next grid when $\varepsilon = 0$.

The *Permanent Exploration Q-learning probability matching* (PEQL-prob) algorithm is described in Algorithm 5. Unlike the PEQL-ε approach, when the UAV needs to decide the next action, the Q-value of each action is normalized and used as the probability of choosing that action. In cases where some possible actions have zero or negative Q-values, we assign a small positive value to them. This encourages exploration when Q-values are zero or neg-

Algorithm 4: PEQL-ε

```
1  Function PEQL-ε(s_0, ε, N):
2      s ← s_0
3      while Termination condition not reached do
4          rnd ← Random(0, 1)
5          if rnd ≤ ε then
6              a ← Action[Random(Action.len)]
7          else
8              if maximum Q(s, a) is not unique then
9                  a ← choose randomly from maximum Q(s, a)
10             else
11                 a ← Action[max_a Q(s, a).index]
12             end
13         end
14         Take action a, get R(s), move to s'
15         Q-UPDATE(s, R(s), N)
16         s ← s'
17     end
18 end
```

ative, such as at the beginning of experiment or when there have been no animals in an area for a long time.

5.4 Experimental study

5.4.1 Simulation environment

The whole observation area is deployed by sensors and then classified into grids. One node in each grid is selected periodically as the clusterhead node responsible for collecting messages from other sensors and communicating with the UAV.

Dataset and UAV

We test our models with two real world animal datasets. The ZebraNet [48] dataset, which

Algorithm 5: PEQL-prob

```
1  Function PEQL-prob(s_0, N):
2      s ← s_0
3      while Termination condition not reached do
4          Q[all] = Action[all].Qvalue
5          if any Q[s, a] ≤ 0 then
6              Q[s, a] = 0.01
7          end
8          Prob[all] ← Normalize(Q[all])
9          a = Action[Random(Prob[all])]
10         Take action a, get R(s), move to s'
11         Q-UPDATE(s, R(s), N)
12         s ← s'
13     end
14 end
```

was used in our previous work [59], contains the location information of 5 zebras in 2005 at Nanyuki, Kenya. The sampling time of the GPS traces is 10 minutes and the total experiment time is 14 days. The vultures dataset [82] contains the movement traces of several groups of white backed & lappet-faced vultures in Namibia. The sampling time of the GPS traces is 10 minutes between 6:00 am-6:00 pm everyday. The experiment duration of this dataset is from 2008 to 2010. The difference between these two datasets is that vultures have much faster movement speed and usually have activities in a much broader area. Therefore, the vultures are harder to be tracked and monitored. In other words, animal movement patterns in the vultures dataset are more complex and unpredictable compared with the animal movement patterns in the ZebraNet dataset.

We mapped both datasets into our $100km \times 100km$ interest area. For the parameters of the UAV, we decided to take as reference the 30 km/h speed of the MUGIN 3M UAV, a cost-effective choice for civilian animal monitoring applications. Both animal datasets contain GPS traces at 10 minute intervals, which is considered as one "round" in our simulator. Thus, the UAV moves with a speed of 5km/round.

In modeling the clusterheads, we will assume that there are no events lost between the individual sensors and the clusterhead. However, to model the limited buffer of the clusterhead, we assume that events have an expiration time t_{exp}, after which they are dropped from the buffer.

We collect three metrics during our experiments: the VoI received by the user at the moment when an event is picked up, the message delay between the creation of the event and the moment when it is picked up, and the number of collected events, excluding the expired events.

In addition to PEQL-ε and PEQL-prob, we implemented three other baseline approaches. For the *random* approach, at each state, the UAV chooses a random grid cell to visit next. In the *traveling salesman problem (TSP)* based approach, the UAV cycles on the shortest path traversing all the grid positions. Finally, in the *greedy* approach the UAV chooses the grid cell providing the highest initial VoI at the current moment (breaking ties randomly). While the random and TSP approaches do not take into consideration the collected VoI, the greedy approach adaptively changes function of the VoI.

Table 5.1 includes the parameter values used in our experiments. For the performance evaluation of the proposed path planning approaches: PEQL-ε and PEQL-prob, we compare their outcomes with three other approaches: greedy, traveling salesman problem-based (TSP) and random. Each of these approaches independently conduct path planning for the UAV in order to efficiently collect event messages.

Table 5.1: Simulation parameters.

Area of interest	$100km \times 100km$
Grid size	4 x 4
Time unit (1 round)	600s
UAV speed	$5km/round$
VoI decay B	0.03
Initial reward $\mathcal{V}_0$	10.0
Penalty $R_{penalty}$	-1.0
t^e_{min}	2 *rounds*
t^e_{int}	24 *rounds*
Expiration time t_{exp}	72/96 *rounds*
Learning rate α	0.5
Discount factor γ	0.5

5.4.2 Simulation results

This section reports the experimental results of the simulation experiments for the five path planning algorithms considered. All experimental data was collected over 1000 simulation rounds (10,000 minutes) from both the zebra and the vultures datasets. The performance metrics were averaged over 1000 experiments. Unless specified otherwise the simulation parameters had the values described in Table 5.1.

Fig. 5.5 shows the accumulation of VoI values in time. The difference between the two animal behaviors is evident in the fact that for all algorithms, the vultures dataset yields a more variable curve. PEQL-prob marginally outperforms PEQL-ε on both datasets and both of them are clearly better than the other approaches on both datasets *after the first 2000 minutes*. We conjecture that this time is necessary for the reinforcement learning approach to converge and for the impact of the superior policy to become visible. As expected, the random approach performs consistently the worst. Interestingly, the greedy strategy outperforms TSP in the long run on the zebra datasets, but the order is reversed on the vulture dataset,

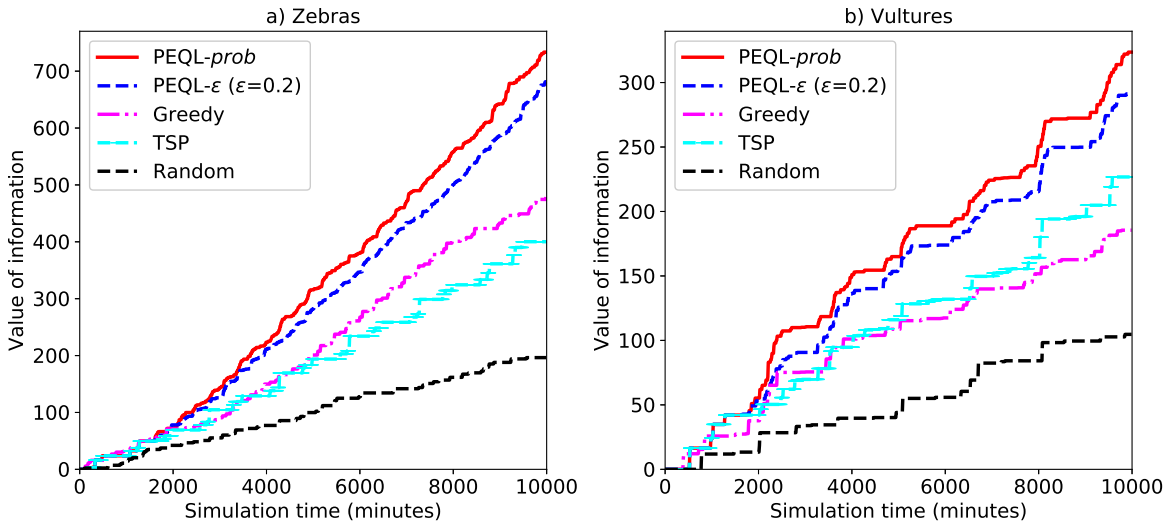

Figure 5.5: Accumulation of VoI in time for the zebra dataset (left) and vulture dataset (right).

a phenomena motivated by the fact that the unpredictability and fast movement of vultures trigger more incorrect decisions in the greedy strategy.

As the movement of the UAV interacts with the randomness in the movement of the animals, different events will be delivered to the user with different delays - even in the case of predictable UAV movement such as following the TSP path. We are interested in delivering the messages with the lowest average delay. Fig. 5.6 shows box plots for the minimum, lower quartile, median, upper quartile and the maximum of the message delays. As messages are dropped at the expiration time t_{exp} of 72 rounds for the zebra and 96 rounds for the vultures dataset, the distributions stop at these values. We notice that for all path planning algorithms, as expected, the message delays are longer and have a wider spread for the vultures dataset, where the higher variability of the animal movement makes it more difficult for any algorithm to pick the data chunks early. Between the algorithms, both PEQL-prob and PEQL-ε exhibit significantly shorter median message delays than the other algorithms

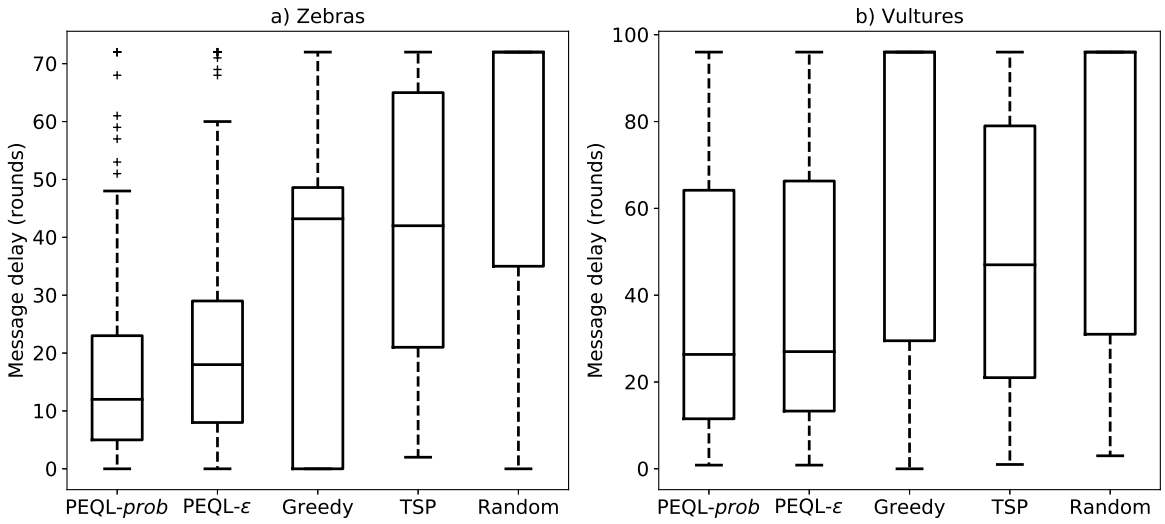

Figure 5.6: Box-plots of the message delay distribution for the zebra dataset (left) and vulture dataset (right). In situations where the median is not visible, the median, upper quartile and maximum are all at the top of the box.

on both datasets. In addition, for the more predictable zebra dataset, PEQL-prob evolves a policy with consistently shorter message delays (at all quartiles) compared with PEQL-ε. An interesting observation can be made about the greedy algorithm on the zebra dataset: the first quartile of the messages has a delay close to zero because in some situations the greedy algorithm simply gets "lucky". However, the median case is significantly worse than the PEQL variations. On the more unpredictable vultures dataset, it is much harder to get lucky, thus even the minimum value is worse than for the PEQL algorithms.

The percentage of events collected by the UAV are shown in Fig. 5.7. These are the messages collected before the expiration time t_{exp}. Under the assumptions, during the 1000 rounds of simulated time, there are 151 events generated in the zebra dataset and 98 in the vultures dataset. It would appear that the regular traversal of the nodes based on TSP would perform the best in this case; however, under experimental settings the expiration time is shorter

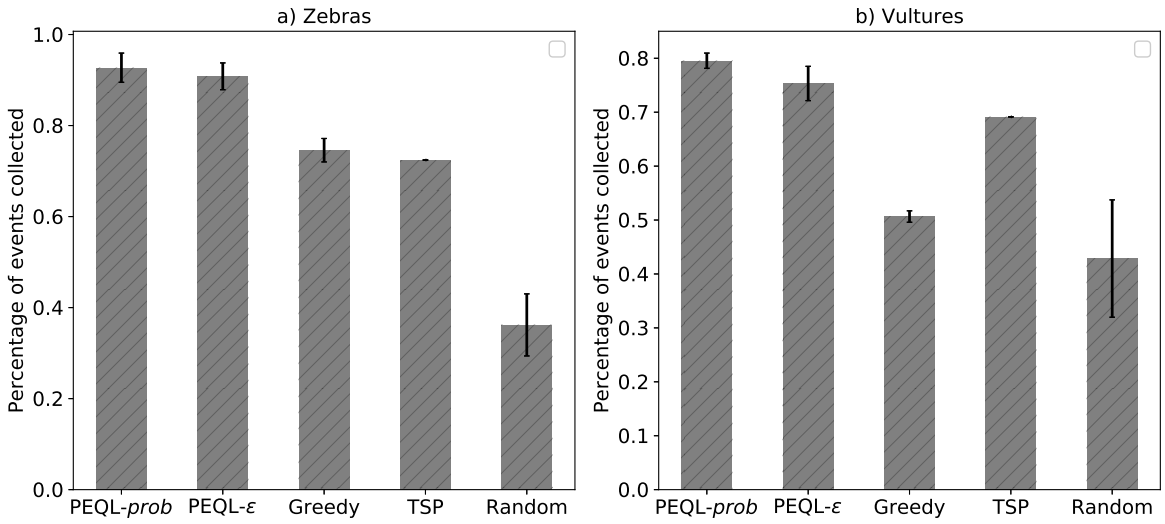

Figure 5.7: Percentage of events successfully collected before the expiration time t_{exp}.

than the time needed by the TSP algorithm to cover the whole area given the velocity of the UAV. This is a situation often encountered where large areas need to be covered with limited resources. The results show that PEQL-prob and PEQL-ε collect a larger percentage of events before their expiration compared to the other approaches, as learning the locations where animals are likely to be allows the UAV to cover these locations more often. As for the other metrics, the performance is better on the more predictable zebra dataset, but the learning-based approaches clearly outperform other models on the vulture dataset as well.

The impact of the algorithm parameters

In the previous section, we compared the two proposed approaches PEQL-ε and PEQL-prob against baseline approaches and we established that they outperform all baselines, except during the early stages of learning. In this section, we study how the parameters N and ε of these two algorithms impact the performance.

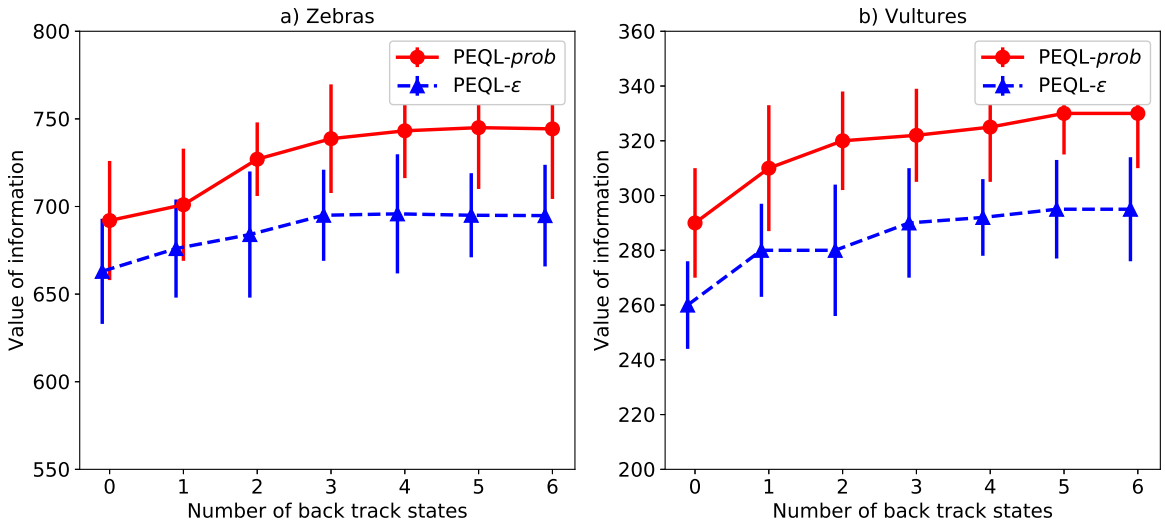

Figure 5.8: Impact of the episode length N on the collected VoI

For both the PEQL-ε and PEQL-prob approaches, in Equation 5.5, we introduced a technique to backpropagate the reward to the Q-values of an episode of length N. For $N = 0$, this approach falls back to the baseline Q-learning model. Fig. 5.8 shows the impact of the N value on the collected VoI at the end of the scenario. We find the episode-based Q-value update provides a clear performance improvement for both algorithms on both datasets. For small values of N, the collected VoI increases with N. The performance, however reaches a plateau at about $N = 4$ in the zebra dataset and $N = 5$ in the vulture dataset.

Another aspect in determining the performance is the exploration probability ε for the PEQL-ε algorithm. Figure 5.9 compares the VoI collected for different values of ε. A value of $\varepsilon = 0$ corresponds to a Q-learning algorithm with no exploration, while $\varepsilon = 1$ yields random movement. We found that for both datasets the best performance is obtained for $\varepsilon = 0.2$, the value we used in comparing PEQL-ε with other algorithms.

Our final experiment concerns with the size of the grid. For a given area of interest, we can

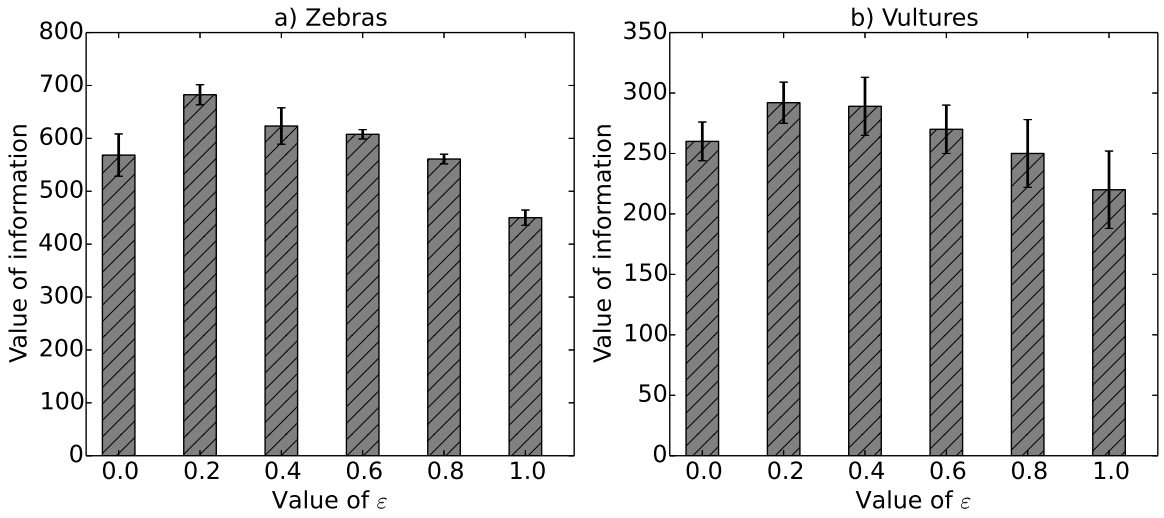

Figure 5.9: The impact of the exploration probability ε on the VoI collected using the PEQL-ε algorithm.

organize the sensors into grids of different sizes. Fig. 5.10 shows the impact of the grid size on the path planning performance of PEQL-ε and PEQL-prob in terms of VoI. We notice that the performance is lower when the grid size is either too small or too large. For smaller grid sizes the learning-based algorithms have little advantage compared to visiting all grid cells, while for large grid sizes, the quality of the learning decreases. We notice that for both approaches and on both datasets the best performance is obtained at a grid size of 4×4 or 5×5, in which the use of the 4×4 grid for the rest of our experiments is justified.

We note that the grid size has implications beyond the performance of the UAV path planning. For instance, a finer grid requires a larger number of nodes that can serve as clusterhead, while a coarser grid might be better served with clusterhead nodes with a larger message buffer. Exploring these settings is beyond the scope of this dissertation, as we assumed that the grid size is the only parameter that changes in our experiments.

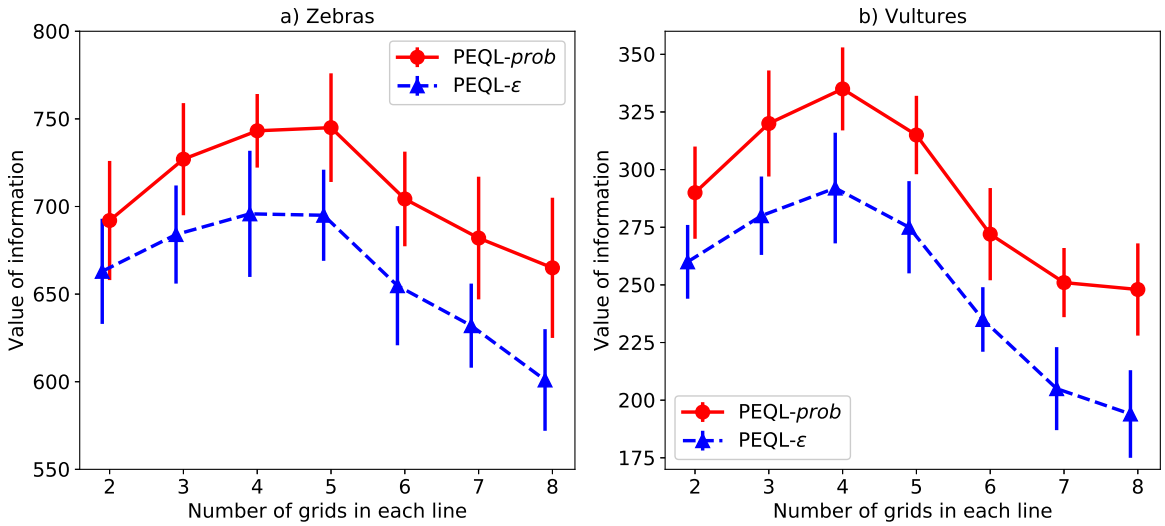

Figure 5.10: The impact of number of virtual grids in the network.

In this chapter, we investigated a wild animal monitoring scenario, where data is recorded by clusters of sensors deployed in a large area lacking infrastructure. The data is collected by a UAV visiting the clusterheads. We show that the path taken by the UAV is a critical contributor to the quality of the tracking. Based on ideas taken from reinforcement learning, we proposed two algorithms for the UAV path planning PEQL-ε and PEQL-prob. Through a series of experiments using real-world movement traces of zebras and vultures, we studied the performance of the approaches on the metrics of value of information, message delay and percentage of collected events.

While our experiments are based on wild animal monitoring, the proposed approaches are equally applicable to other situations where data needs to be collected about a phenomena in a geographic area that is evolving slowly compared to the velocity of the data collector. Examples might include glacier movement, underwater pipeline leaks, and others.

CHAPTER 6: PATH PLANNING BASED ON PREDICTED DISTRIBUTION FOR ANIMAL MONITORING APPLICATIONS

In this chapter, we present another path planning approach for the unmanned aerial vehicle (UAV) based on animal appearance distribution prediction [60]. The application scenario here is the same as the animal monitoring scenario introduced in the previous chapter [59, 81]. In Chapter 5, we define a mathematical metric – value of information (VoI) to calculate the collected data value as an exponential function that decays as time passes. We also propose a Reinforcement Learning-based path planning approaches for finding species such as zebras.

In this chapter, we propose a model predictive control (MPC) method to help the UAV plan its path. First, we train a neural network that learns the animal appearance patterns from the collected historical data. Second, we apply the trained model to predict the future probability distribution of animal appearance. Finally, we propose a traveling salesman problem (TSP)-based path planning approach for the UAV using the real-time predicted animal appearance distribution (TSP-D). Even though we focus on the animal monitoring application, the proposed MPC method is applicable to any time-sensitive data collection applications where historical data can be learned by our predictive model.

We evaluate the performance of the proposed predictive model and the path planning approach using a dataset [82] that includes traces of groups of white backed, lappet-faced vultures in Namibia. We compare the TSP-D path planning approach against greedy and Naive TSP heuristics and show that it outperforms other approaches in terms of VoI, message delay, and the percentage of events collected.

6.1 Network model

In this application, the objective is to monitor and track specific animals' appearance. The sensor nodes are deployed in the target wild area while a UAV operates as a mobile sink node for the data collection.

We divide the whole area into virtual grids. In this scenario, the sensors that in a particular grid are considered as a cluster and one of them acts as the clusterhead. Fig. 6.1 shows a snapshot of the divided virtual grids. The UAV visits the clusterheads for the data collection. After visiting a virtual grid, for instance Grid 6 in Fig. 6.1, the UAV can visit one of the neighboring grids or just hover over in the same grid.

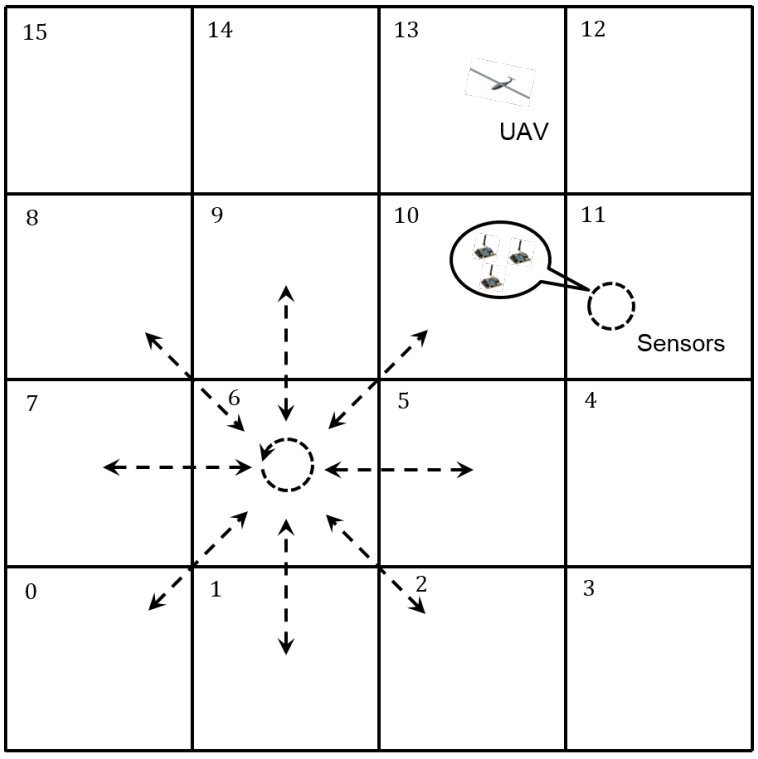

Figure 6.1: The movement choices of the UAV when it flies over Grid 6.

As specified in [59], the sensors are responsible for animal monitoring and sending the sensed events to the clusterhead. Sensor nodes are deployed by uniform random distribution in the large observation area. The sensors inside a virtual grid form a cluster and the clusterhead is selected periodically. Sensors can communicate with the clusterhead via direct or hop-by-hop wireless communication. A sensor creates a new *event message* when it detects an animal. The clusterhead is responsible for receiving event messages from other sensors and then reporting all the event data to the UAV when the UAV comes into its transmission range.

In this network model, the UAV is used as an autonomous mobile sink for gathering time-sensitive information. The usage of UAV brings advantages such as movement flexibility and high speed. Moreover, the UAV not only overcomes the geographical challenge but also has minimal effects on animals. Thus, our model is based on using a single UAV to collect data from the ground sensors, more specifically, the clusterheads of the virtual grids.

6.2 Animal distribution prediction

Directly detecting animals using UAV (e.g., using attached camera) in a large wild area is a hard problem. On the other hand, when we have sensed historical data from the target area, we can analyze the animals' movement patterns.

Fig. 6.2 shows examples of animal appearance pattern in two different virtual grids. As we can see, animals usually show up at specific hours in a day and a similar pattern is repeated every day. We start to build a model with the goal of learning the animal appearance in each virtual grid based on the observed regularities.

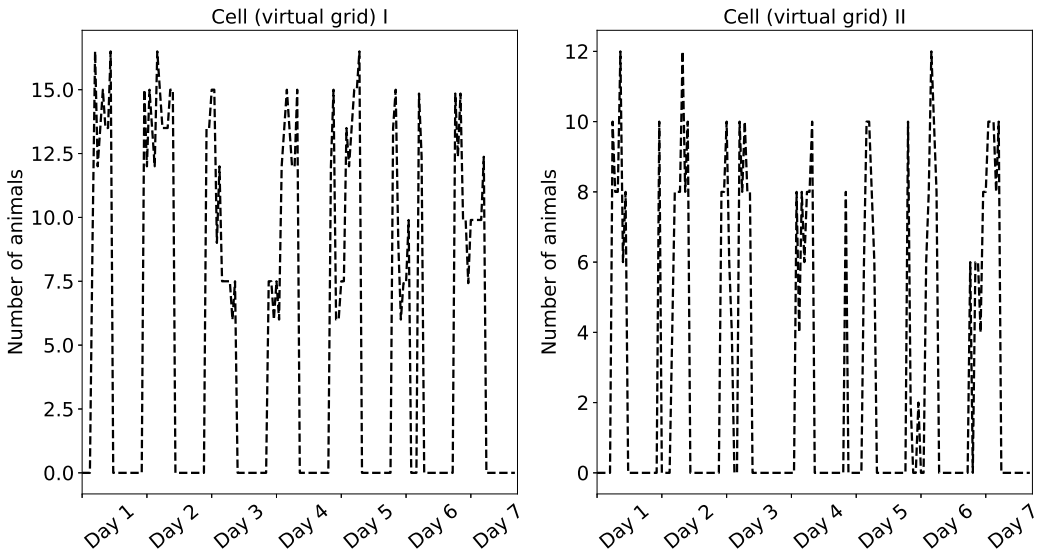

Figure 6.2: Animal appearance patterns in two virtual grids.

First, we divide a day into time-steps $\{t_0, t_1, ...t_M\}$ such that t_i means the i^{th} time-step of a day. Second, we count the number of detected animals at each time-step for every virtual grid. Fig. 6.3 shows the input and output used to train the predictive model. For time-step t_i, the input data x_i consists of two parts: d_i and e_i. d_i represents potential affecting factors such as time-step in the day, month. e_i represents number of animals detected in each virtual grid at time-step t_i. Given the input data x_i, we use a fully connected feed-forward neural network (Fig. 6.4) to predict the output data y_i' which is the number of animals in each virtual grid in the next time-step. We define the cost as the mean squared error between predicted y_i' and the ground-truth y_i and minimize it using stochastic gradient descent.

With the trained model, we can predict the animal distributions by inputting the predicted distribution of the previous time-step. In addition, as the UAV visits some grids, the corresponding real number of animals in those virtual grids can be updated, i.e., a partially

observed x_{i+1} can be obtained by updating some values of y'_i. Then, x_{i+1} is used as input to predict the next time step distribution y'_{i+1}. It is important to point out that with this model, we can repeat the prediction for an arbitrary number of time steps without accessing to real observations. However, as the time goes, the prediction error compounds and the accuracy may drop.

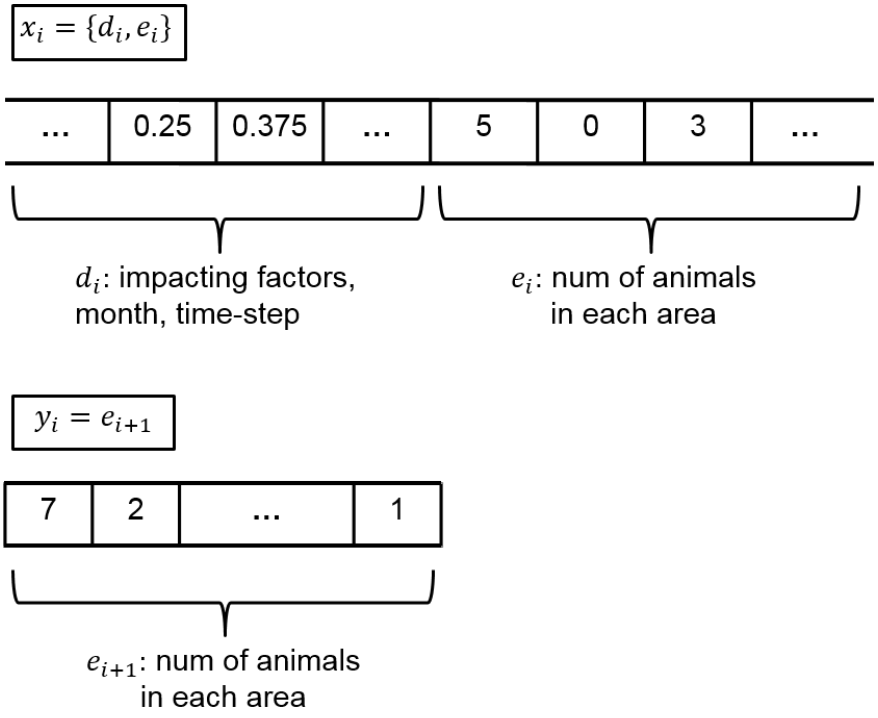

Figure 6.3: Input and output data structure.

6.3 Path planning with predicted animal distribution

In this application, we treat animal detection as an event and the corresponding event message should be collected as soon as possible.

We define the metric *value of information* (VoI) to evaluate the importance of the collected

information by the UAV. VoI is defined as the exponential decay function

$$F_{VoI}(t) = A \times exp\{-B * (t_{collect} - t_{initial})\}, \tag{6.1}$$

where A is the initial value of the event while B is the decay factor. Under this formulation, the objective of the path planning is to maximize the collected values.

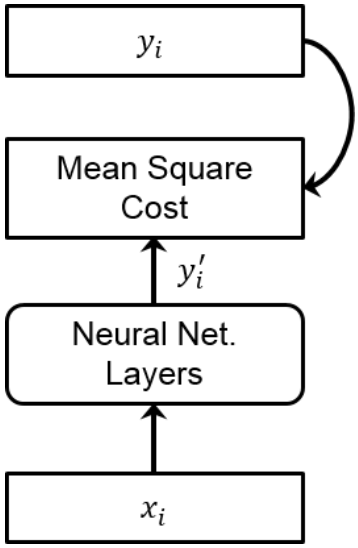

Figure 6.4: Learning model structure.

Given such a scenario, we propose a path planning approach based on the predicted animal distributions for each grid. Given a time period $[t, t+T]$ (where T is a hyper-parameter), finding the optimal path is similar to solving a traveling salesman problem (TSP) while the goal is to maximize the estimated rewards. We name our proposed path planning approach as *TSP-D* where "D" represents the continuously predicted animal distribution.

We implement a tree structure path exploration search to find the optimal path. As shown in Fig. 6.5, the numbers inside the tree nodes represent the indices of the virtual grids while the edges between them represent the travelling time of the UAV to a neighboring grid. This

traveling time information depends on the area of sensor deployment.

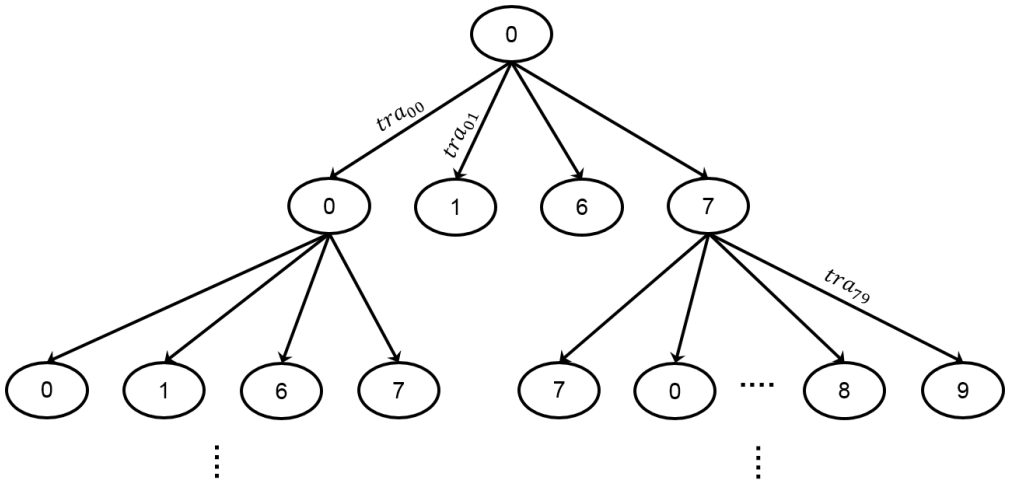

Figure 6.5: Tree structure of path exploration.

Note that grids in Fig. 6.5 are not connected to all other grids. Instead, they are only connected to a subset of them where the nodes are neighbors and it is possible for the UAV to fly to that neighbor (at most 9, including self). In addition, due to specific characteristics of animal movements, the distribution matrix is sparse. This allows us to further prune the tree. More specifically, for each layer in the tree, candidate paths that end in the same virtual grid and contain the same event IDs collected, are merged into one path with the maximum values. Finally, the candidate path with the maximum value is chosen by the UAV. The pseudo code for the path planning approach is given in Algorithm 6.

After visiting a grid, the UAV repeats the path planning algorithm to decide the next visiting grid. With the updated x (by the observation from current grid) and the *lookahead* parameter T, Algorithm 6 returns the next visiting grid.

Algorithm 6: Path planning of the UAV

```
Travel time look up table:
t_table = d_table/speed_UAV
Pre-trained distribution prediction model Model
Current time-step t
Current animal distribution dist_cur
Path planning lookahead time T
Event creation period Δt
Function CutBranch(CandidatePaths, t):
    for PN in CandidatePaths do
        if t - PN.time > maxtime then
            /* max time to neighbors from t_table */
            remove(PN)
        end
        if Duplicate(PNs.id and PNs.events) then
            KeepMaxOne(PNs)
        end
    end
end
CandidatePaths = []
while t < t + T do
    if t%Δt == 0 then
        dist_cur = Model.predict(dist_cur, t)
        CreateEvents(dist_cur, t)
    end
    List = []
    for PN in CandidatePaths do
        /* PN: PathNode, last grid id of a path */
        Candidate = IsArrivalNeighbor(PN, t_table)
        List.add(Candidate)
    end
    UpdateDist(dist_cur) ← List
    UpdatePath(CandidatePaths) ← List
    CutBranch(CandidatePaths, t)
    t = t + 1
end
Path = MaxRewardPath(CandidatePaths)
return NextGrid(Path)
```

6.4 Experimental Study

The proposed network model and the UAV path planning approach are evaluated in this section.

6.4.1 Simulation environment

Dataset and UAV

We test our models with a real-world vultures dataset [82] which contains the movement traces of several groups of white backed & lappet-faced vultures in Namibia. The GPS traces are recorded with a sampling time interval of 10 minutes everyday from 6am to 6pm in years 2008 to 2010. A single UAV with a fixed speed of $50km/h$ is simulated for event data collection from clusterheads. In the case of battery exhaustion, we assume that the UAV can be recharged so that it can continue to work.

Performance metrics and baselines

To quantitatively evaluate the performance of a path planning strategy, we report the results according to three performance metrics in our simulation study.

- *Value of information (VoI).* Maximizing the VoI is the primary goal in designing the path planning approaches.
- *Message delay.* Since event messages need to be kept in the sensor buffer until being sent to the UAV, message delay is an important metric. Long message delay may cause the loss of event messages.
- *Percentage of events collected.* Whenever an event is created by the sensor, we add a

valid time period, *isvalid*, for each event. In other words, this event expires and the data can not be collected anymore by the UAV after *isvalid* amount of time.

Table 6.1: Simulation parameters

Network size	$100km \times 100km$
Time unit	1 min
UAV speed	50 km/h
VoI parameters (A, B)	(10.0, 0.02)
Time step length	30 $mins$
Events generation period Δt	30 $mins$
Events expiration $isvalid$	300 $mins$

Table 6.1 includes the simulation parameter values used in our experiments. For the performance evaluation of the proposed path planning approach TSP-D, we compare its outcome with the theoretical optimal approach, greedy approach with predicted distribution, and naive TSP-based approach.

6.4.2 Compared approaches

The optimal approach (TSP-D-Optimal)

In this approach, we feed the UAV with the real animal distribution in which we extract from the dataset. Note that this approach is only for comparison purposes and in practice cannot be realized since we cannot predict the future movements of animals with 100% accuracy.

Greedy with predicted distribution (Greedy-D)

This approach also takes our predicted distribution into account; however, the UAV always pursues the highest reward among the neighboring virtual grids instead of expanding dif-

ferent possible paths. If all neighboring grids have the same reward, a random selection is considered.

Naive TSP without predicted distribution (Naive-TSP)

In this approach, we do not provide any prediction information to the UAV. The UAV visits each clusterhead in a fixed order.

6.4.3 Performance results

We report the performance of each path planning approach in terms of the three metrics: VoI, message delay and percentage of events collected. For our proposed TSP-D approach, we also show the performances with different time-step look-aheads.

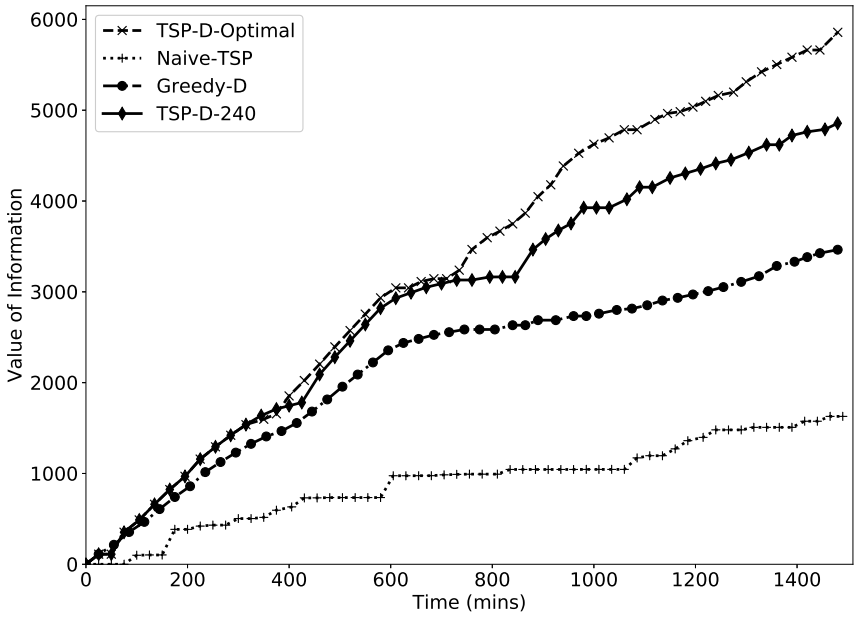

Figure 6.6: Value of information performance.

Fig. 6.6 shows the VoI performance collected by different path planning approaches. As we can see the Naive TSP is the worst approach in this case due to not considering any animal distribution information. The Greedy-D approach is much better than the Naive TSP since it takes the predicted animal distribution as input into path planning. But it always pursues the highest neighboring reward which limits its overall performance. With the same predicted animal distribution as input, our proposed TSP-D outperforms the Greedy-D approach because our solution decides next visiting grid based on an estimated rewards in next *lookahead* time (240 *mins* in Fig. 6.6). However, compared with the optimal solution, TSP-D's performance is a bit worse. Note that although the optimal approach achieves better results, it is not practical as we discovered before, therefore, the real winner is TSP-D approach.

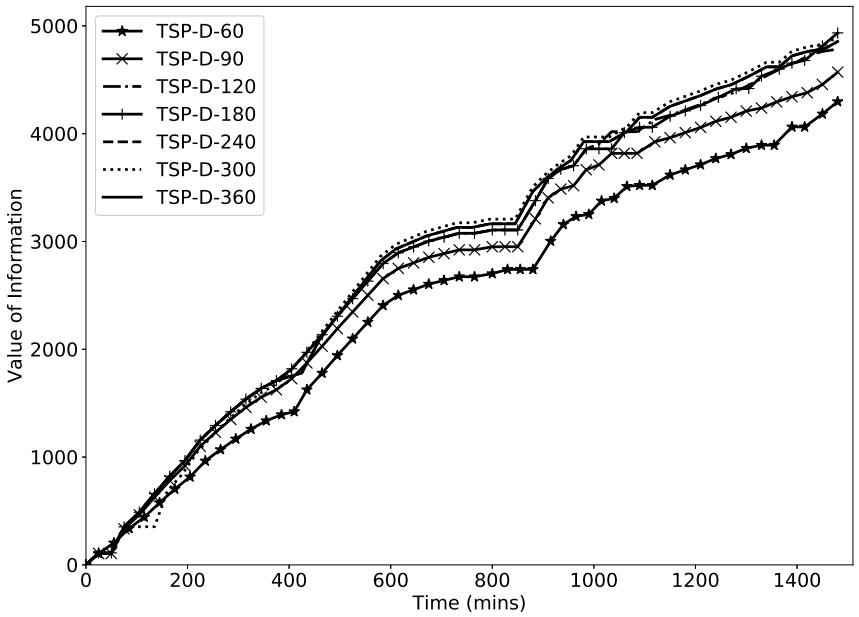

Figure 6.7: The VoI performance with different look ahead time.

Fig. 6.7 shows the VoI performance with different prediction durations. It can be seen that in TSP-D approaches, the performance with *lookahead* time 60 *mins* is lower than others.

The reason would be the same with the Greedy-D approach that the local maximum reward limits the overall performance. As the *lookahead* time increases, the performances of different approaches come close to each other. It might be due to the effect of compounding error in predicting long-term future that misleads the UAV to explore areas where there expectation will not be met.

Fig. 6.8 shows the box plot of message delay of each path planning approach. Naive TSP approach shows the worst performance due to not taking into account any animal information. It is interesting that the Greedy-D approach outperforms other approaches in this metric. The reason is that, in Greedy-D, the UAV always goes to the neighboring grid with highest probability of existing animals. It collects event data in a timely manner but it sacrifices the number of collected events, which can be seen in Fig. 6.9.

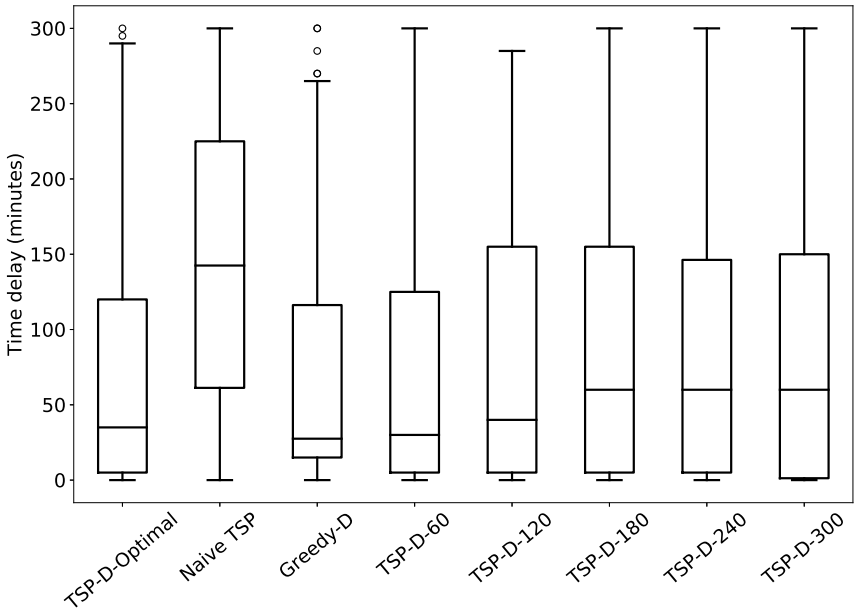

Figure 6.8: Message delay performance.

Fig. 6.9 shows the percentage of events collected by each path planning approach. Let us

remember that an event validation time *isvalid* is added to each event and if the event is not collected after *isvalid* time, it expires. It can be seen that the Naive TSP achieves the best performance in this metric because the UAV goes in a fixed path which guarantees that most events can be collected finally. Greedy-D shows the worst performance. As we explained in Fig. 6.8, it collects events in a timely manner but it does not collect sufficient number of events.

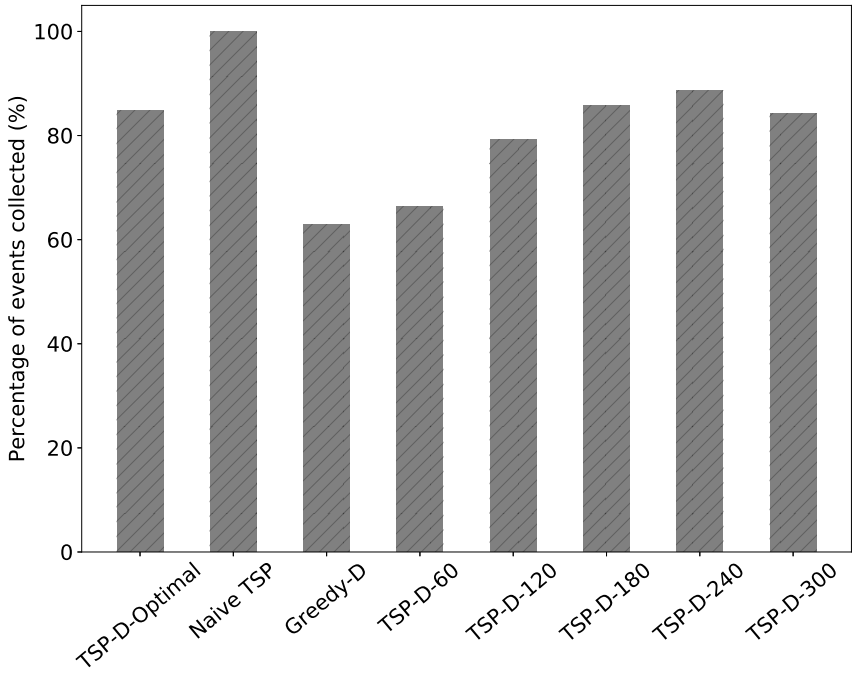

Figure 6.9: Event message collection performance.

Overall, it can be seen in the experimental results that TSP-D highly outperforms the other path planning approaches. Compared to the Greedy-D approach, TSP-D produces $(4855 - 3463)/3463 = 40\%$ increase in VoI and $(159 - 113)/113 = 41\%$ increase in number of events collected. Compared to the Naive TSP approach, TSP-D produces $(4855 - 1628)/1628 = 198\%$ increase in VoI and $(142 - 68)/142 = 52\%$ decrease in median message delay.

Although the Greedy-D path planning approach also relies on the predicted distribution, it always goes to the neighboring grid with highest potential rewards. As we can see in Fig. 6.8 and Fig. 6.9, Greedy-D mostly tries to minimize the message delay while Naive TSP tries to maximizes percentage of collected events. On the other hand, in TSP-D the UAV considers the overall rewards in the next *lookahead* amount of time. Note that as the UAV collects data from the grids, the predicted animal distribution can be partially updated with the observed values. In such a way, a higher performance path planning for the UAV can be obtained after every grid visit.

CHAPTER 7: CONCLUSION AND FUTURE WORK

In this dissertation, we present different path planning approaches based on designing different learning and prediction models on historical data. The goal is to learn state patterns from historical data and then predict future states of the path planning algorithms.

Chapter 3 presents taxi demand and destination prediction based on historical taxi data. With the predicted future taxi demand and destination, chapter 4 present a taxi dispatch center. The goal is to balance the taxi service demand and supply ratio over the city while minimizing passenger waiting time and taxi idle driving distance. We optimize the taxi assignment and rebalance by solving a Mixed Integer Programming (MIP). The objective of the MIP is to minimize the total idle driving distances while serving all the coming requests. We believe that our approach has a great potential to optimize the distribution of taxis throughout the city such that the number of taxis required is minimized.

Chapter 5 and Chapter 6 include two different path planning approaches under the animal monitoring application in which we use an unmanned aerial vehicle (UAV) to search and track animals in wild large area. The first path planning approach is based on the permanent learning from the environment while the second one relies on the recorded animal appearance data in the past.

In terms of future work, we can extend the taxi dispatch center to consider ride-sharing. It is a multi-variable optimization problem for taxi time scheduling and path planning. However, a well-performing demand and destination prediction model is still the key step towards reaching such an integrated and efficient transportation system.

LIST OF REFERENCES

[1] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. "Real-Time Prediction of Taxi Demand Using Recurrent Neural Networks". In: *IEEE Transactions on Intelligent Transportation Systems* 19.8 (Aug. 2018), pp. 2572–2581.

[2] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[3] Christopher M. Bishop. *Mixture density networks*. Tech. rep. Aston University, 1994.

[4] Petrika Gjanci, Chiara Petrioli, Stefano Basagni, Cynthia A. Phillips, Ladislau Bölöni, and Damla Turgut. "Path Finding for Maximizing the Value of Sensed Information in Multi-modal Underwater Wireless Sensor Networks". In: *IEEE Transactions on Mobile Computing* 17.2 (Feb. 2018), pp. 404–418.

[5] Kai Zhang, Zhiyong Feng, Shizhan Chen, Keman Huang, and Guiling Wang. "A Framework for Passengers Demand Prediction and Recommendation". In: *Proc. IEEE SCC*. June 2016, pp. 340–347.

[6] Kai Zhao, Denis Khryashchev, Juliana Freire, Cláudio Silva, and Huy Vo. "Predicting taxi demand at high spatial resolution: Approaching the limit of predictability". In: *Proc. IEEE BigData*. Dec. 2016, pp. 833–842.

[7] Luis Moreira-Matias, Joao Gama, Michel Ferreira, Joao Mendes-Moreira, and Luis Damas. "Predicting taxi passenger demand using streaming data". In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1393–1402.

[8] Neema Davis, Gaurav Raina, and Krishna Jagannathan. "A multi-level clustering approach for forecasting taxi travel demand". In: *Proc. IEEE ITSC*. Dec. 2016, pp. 223–228.

[9] Alexandre de Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. "Artificial Neural Networks Applied to Taxi Destination Prediction". In: *CoRR* abs/1508.00021 (2015).

[10] Javier Alonso-Mora, Alex Wallar, and Daniela Rus. "Predictive routing for autonomous mobility-on-demand systems with ride-sharing". In: *Proc. IEEE/RSJ Intelligent Robots and Systems*. Sept. 2017, pp. 3583–3590.

[11] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. "A Sequence Learning Model with Recurrent Neural Networks for Taxi Demand Prediction". In: *Proc. IEEE Local Computer Networks (LCN)*. Oct. 2017, pp. 261–268.

[12] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. "Taxi Dispatch Planning via Demand and Destination Modeling". In: *Proc. IEEE Local Computer Networks (LCN)*. Oct. 2018, pp. 377–384.

[13] Jing Yuan, Yu Zheng, Liuhang Zhang, Xing Xie, and Guangzhong Sun. "Where to find my next passenger". In: *Proc. ACM UbiComp*. Sept. 2011, pp. 109–118.

[14] Shuo Ma, Yu Zheng, and Ouri Wolfson. "T-share: A large-scale dynamic taxi ridesharing service". In: *Proc. IEEE ICDE*. Apr. 2013, pp. 410–421.

[15] Huigui Rong, Xun Zhou, Chang Yang, Zubair Shafiq, and Alex Liu. "The Rich and the Poor: A Markov Decision Process Approach to Optimizing Taxi Driver Revenue Efficiency". In: *Proc. ACM CIKM*. Oct. 2016, pp. 2329–2334.

[16] José Azevedo, Pedro M d'Orey, and Michel Ferreira. "On the mobile intelligence of autonomous vehicles". In: *Proc. IEEE NOMS*. Apr. 2016, pp. 1169–1174.

[17] Weipeng Jing, Likun Hu, Lei Shu, Mithun Mukherjee, and Takahiro Hara. "RPR: recommendation for passengers by roads based on cloud computing and taxis traces data". In: *Personal and Ubiquitous Computing* 20.3 (2016), pp. 337–347.

[18] Pablo Samuel Castro, Daqing Zhang, Chao Chen, Shijian Li, and Gang Pan. "From taxi GPS traces to social and community dynamics: A survey". In: *ACM Computing Surveys (CSUR)* 46.2 (2013), p. 17.

[19] Desheng Zhang, Tian He, Shan Lin, Sirajum Munir, and John A. Stankovic. "Taxi-Passenger-Demand Modeling Based on Big Data from a Roving Sensor Network". In: *IEEE Transactions on Big Data* PP.99 (2016), pp. 1–1.

[20] Fei Miao, Shuo Han, Shan Lin, John A. Stankovic, Desheng Zhang, Sirajum Munir, Hua Huang, Tian He, and George J. Pappas. "Taxi Dispatch With Real-Time Sensing Data in Metropolitan Areas: A Receding Horizon Control Approach". In: *IEEE Transactions on Automation Science and Engineering* 13.2 (2016), pp. 463–478.

[21] Ximing Chen, F. Miao, G. J. Pappas, and V. Preciado. "Hierarchical data-driven vehicle dispatch and ride-sharing". In: *Proc. IEEE Decision and Control (CDC)*. Dec. 2017, pp. 4458–4463.

[22] Jane Lin, Sandeep Sasidharan, Shuo Ma, and Ouri Wolfson. "A Model of Multimodal Ridesharing and Its Analysis". In: *Proc. IEEE Mobile Data Management (MDM)*. June 2016, pp. 164–173.

[23] Huanyang Zheng and Jie Wu. "Online to Offline Business: Urban Taxi Dispatching with Passenger-Driver Matching Stability". In: *IEEE Int'l. Conf. on Distributed Computing Systems (ICDCS)*. June 2017, pp. 816–825.

[24] Fei Miao, S. Han, S. Lin, Q. Wang, J. A. Stankovic, A. Hendawi, D. Zhang, T. He, and G. J. Pappas. "Data-Driven Robust Taxi Dispatch Under Demand Uncertainties". In: *IEEE Transactions on Control Systems Technology* 27.1 (Jan. 2019), pp. 175–191.

[25] Damla Turgut and Ladislau Bölöni. "A pragmatic value-of-information approach for intruder tracking sensor networks". In: *Proc. IEEE Int'l Conf. on Communications*. Ottawa, Canada, June 2012, pp. 4931–4936.

[26] Damla Turgut and Ladislau Bölöni. "IVE: improving the value of information in energy-constrained intruder tracking sensor networks". In: *Proc. IEEE Int'l Conf. on Communications*. Budapest, Hungary, June 2013, pp. 6360–6364.

[27] Stefano Basagni, Ladislau Bölöni, Petrika Gjanci, Chiara Petrioli, Cynthia A Phillips, and Danila Turgut. "Maximizing the value of sensed information in underwater wireless sensor networks via an autonomous underwater vehicle". In: *Proc. IEEE Int'l Conf. on Computer Communications*. Toronto, Canada, Apr. 2014, pp. 988–996.

[28] Fahad Ahmad Khan, Sehar Butt, Saad Ahmad Khan, Damla Turgut, and Ladislau Bölöni. "Value of Information based Data Retrieval in UWSNs". In: *Sensors* 18.10 (Oct. 2018).

[29] Fahad Ahmad Khan, Saad Ahmad Khan, Damla Turgut, and Ladislau Bölöni. "Greedy path planning for maximizing value of information in underwater sensor networks". In: *Proc. IEEE Workshops Local Computer Networks*. Edmonton, Canada, Sept. 2014, pp. 610–615.

[30] Fahad Ahmad Khan, Saad Ahmad Khan, Damla Turgut, and Ladislau Bölöni. "Scheduling multiple mobile sinks in Underwater Sensor Networks". In: *Proc. of IEEE 40th Conf. on Local Computer Networks (LCN)*. 2015, pp. 149–156.

[31] Ladislau Bölöni, Damla Turgut, Stefano Basagni, and Chiara Petrioli. "Scheduling data transmissions of underwater sensor nodes for maximizing value of information". In: *Proc. IEEE Global Communications Conference*. Atlanta, GA, USA, Dec. December 2013, pp. 438–443.

[32] Fahad Ahmad Khan, Saad Ahmad Khan, Damla Turgut, and Ladislau Bölöni. "Optimizing resurfacing schedules to maximize value of information in UWSNs". In: *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE. 2016, pp. 1–5.

[33] Chien-Fu Cheng and Chao-Fu Yu. "Data Gathering in Wireless Sensor Networks: A Combine-TSP-Reduce Approach". In: *IEEE Trans. Vehicular Technology* 65.4 (2016), pp. 2309–2324.

[34] Geoffrey A. Hollinger, Sunav Choudhary, Parastoo Qarabaqi, Christopher Murphy, Urbashi Mitra, Gaurav S. Sukhatme, Milica Stojanovic, Hanumant Singh, and Franz Hover. "Underwater Data Collection Using Robotic Sensor Networks". In: *IEEE Journal on Selected Areas in Communications* 30.5 (June 2012), pp. 899–911.

[35] Hamidreza Salarian, Kwan-Wu Chin, and Fazel Naghdy. "An Energy-Efficient Mobile-Sink Path Selection Strategy for Wireless Sensor Networks". In: *IEEE Trans. Vehicular Technology* 63.5 (June 2014), pp. 2407–2419.

[36] Jianqiang Li, Genqiang Deng, Chengwen Luo, Qiuzhen Lin, Qiao Yan, and Zhong Ming. "A Hybrid Path Planning Method in Unmanned Air/Ground Vehicle (UAV/UGV) Cooperative Systems". In: *IEEE Transaction on Vehicular Technology* 65.12 (Dec. 2016), pp. 9585–9596.

[37] Fahira Sangare, Yong Xiao, Dusit Niyato, and Zhu Han. "Mobile Charging in Wireless-Powered Sensor Networks: Optimal Scheduling and Experimental Implementation". In: *IEEE Trans. Vehicular Technology* 66.8 (Aug. 2017), pp. 7400–7410.

[38] Ming Ma and Yuanyuan Yang. "SenCar: An Energy-Efficient Data Gathering Mechanism for Large-Scale Multihop Sensor Networks". In: *IEEE Transaction on Parallel and Distributed Systems* 18.10 (Oct. 2007), pp. 1476–1488.

[39] Damla Turgut and Ladislau Bölöni. "Heuristic approaches for transmission scheduling in sensor networks with multiple mobile sinks". In: *The Computer Journal* 54.3 (Mar. 2011), pp. 332–344.

[40] Damla Turgut and Ladislau Bölöni. "Three heuristics for transmission scheduling in sensor networks with multiple mobile sinks". In: *Proceedings of International Workshop on Agent Technology for Sensor Networks (ATSN-08), in conjunction with the Seventh Joint Conference on Autonomous and Multi-Agent Systems (AAMAS 2008)*. May 2008, pp. 1–8.

[41] Ladislau Bölöni and Damla Turgut. "Should I send now or send later? A decision-theoretic approach to transmission scheduling in sensor networks with mobile sinks". In: *Special Issue of Wiley's Wireless Communications and Mobile Computing Journal (WCMC) on Mobility Management and Wireless Access* 8.3 (Mar. 2008), pp. 385–403.

[42] Rouhollah Rahmatizadeh, Saad Ahmad Khan, Anura P Jayasumana, Damla Turgut, and Ladislau Bölöni. "Routing towards a mobile sink using virtual coordinates in a wireless sensor network". In: *Proc. IEEE Int'l Conf. on Communications*. Sydney, Australia, June 2014, pp. 12–17.

[43] Rouhollah Rahmatizadeh, Saad Khan, Anura P. Jayasumana, Damla Turgut, and Ladislau Bölöni. "Circular update directional virtual coordinate routing protocol in sensor networks". In: *Proceedings of IEEE GLOBECOM'15*. Dec. 2015, pp. 1–6.

[44] Guoqiang Wang, Damla Turgut, Ladislau Bölöni, Yongchang Ji, and Dan Marinescu. "Improving routing performance through m-limited forwarding in power-constrained wireless networks". In: *Journal of Parallel and Distributed Computing (JPDC)* 68.4 (Apr. 2008), pp. 501–514.

[45] Gürkan Solmaz and Damla Turgut. "Optimizing Event Coverage in Theme Parks". In: *Wireless Networks (WINET) Journal* 20.6 (Aug. 2014), pp. 1445–1459.

[46] Gürkan Solmaz and Damla Turgut. "Event Coverage in Theme Parks Using Wireless Sensor Networks with Mobile Sinks". In: *Proceedings of IEEE ICC'13*. June 2013, pp. 1522–1526.

[47] Kae Hsiang Kwong, T. T. Wu, H. G. Goh, K. Sasloglou, B. Stephen, I. Glover, C. Shen, W. Du, C. Michie1, and I. Andonovic. "Implementation of herd management systems with wireless sensor networks". In: *IET Wireless Sensor Systems* 1.2 (June 2011), pp. 55–65.

[48] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet". In: *SIGARCH Comput. Archit. News* 30.5 (Dec. 2002), pp. 96–107.

[49] Peter Christiansen, Kim Arild Steen, Rasmus Nyholm Jørgensen, and Henrik Karstoft. "Automated Detection and Recognition of Wildlife Using Thermal Cameras". In: *Sensors* 14.8 (2014), pp. 13778–13793.

[50] Efren Lopes Souza, Andre Campos, and Eduardo Freire Nakamura. "Tracking targets in quantized areas with wireless sensor networks". In: *Proc. IEEE Local Computer Networks (LCN)*. Bonn, Germany, Oct. 2011, pp. 235–238.

[51] Ion Emilian Radoi, Janek Mann, and DK Arvind. "Tracking and monitoring horses in the wild using wireless sensor networks". In: *Proc. IEEE Wireless and Mobile Computing, Networking and Communications*. Abu Dhabi, UAE, Oct. 2015, pp. 732–739.

[52] Chakchai So-In, Comdet Phaudphut, Smarn Tesana, Nutnicha Weeramongkonlert, Kasidit Wijitsopon, Urachart KoKaew, Boonsup Waikham, and Saiyan Saiyod. "Mobile animal tracking systems using light sensor for efficient power and cost saving motion detection". In: *Proc. IEEE Communication Systems, Networks and Digital Signal Processing*. Poznan, Poland, July 2012, pp. 1–6.

[53] Jose Anand, Aida Jones, T. K. Sandhya, and Konthoujam Besna. "Preserving national animal using wireless sensor network based hotspot algorithm". In: *Proc. IEEE Int't Conf. on Green High Performance Computing*. Nagercoil, Tamilnadu, India, Mar. 2013, pp. 1–6.

[54] Andrea Santos-Garcia, María Marta Jacob, W Linwood Jones, William E Asher, Yazan Hejazin, Hamideh Ebrahimi, and Monica Rabolli. "Investigation of rain effects on Aquarius sea surface salinity measurements". In: *Journal of Geophysical Research: Oceans* 119.11 (2014), pp. 7605–7624.

[55] Hamideh Ebrahimi, Shadi Aslebagh, and Linwood Jones. "Use of Monte Carlo simulation in remote sensing data analysis". In: *Proc. IEEE Southeastcon*. Jacksonville, FL, USA, 2013, pp. 1–4.

[56] Zhihai He, Roland Kays, Zhi Zhang, Guanghan Ning, Chen Huang, Tony X. Han, Josh Millspaugh, Tavis Forrester, and William McShea. "Visual Informatics Tools for Supporting Large-Scale Collaborative Wildlife Monitoring with Citizen Scientists". In: *IEEE Circuits and Systems Magazine* 16.1 (Feb. 2016), pp. 73–86.

[57] Xiuhong Li, Xiao Cheng, Rongjin Yang, Haijing Zhang, and Jialin Zhang. "Design and implementation of an audio-visual integrated wireless sensor remote monitoring network on Wetland". In: *Proc. IEEE Int't Conf. on Mechatronic Sciences, Electric Engineering and Computer*. Shengyang, China, Dec. 2013, pp. 609–614.

[58] Luis Camacho, Reynaldo Baquerizo, Joel Palomino, and Michel Zarzosa. "Deployment of a Set of Camera Trap Networks for Wildlife Inventory in Western Amazon Rainforest". In: *IEEE Sensors Journal* 17.23 (Dec. 2017), pp. 8000–8007.

[59] Jun Xu, Gürkan Solmaz, Rouhollah Rahmatizadeh, Damla Turgut, and Ladislau Bölöni. "Animal monitoring with unmanned aerial vehicle-aided wireless sensor networks".

In: *Proc. IEEE Local Computer Networks (LCN)*. Clearwater, FL, USA, Oct. 2015, pp. 125–132.

[60] Jun Xu, Gürkan Solmaz, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. "Providing Distribution Estimation for Animal Tracking with Unmanned Aerial Vehicles". In: *Proc. IEEE Globecom*. Dec. 2018.

[61] Mustafa I. Akbas and Damla Turgut. "Lightweight Routing with Dynamic Interests in Wireless Sensor and Actor Networks". In: *Elsevier Ad Hoc Networks* 11.8 (Nov. 2013), pp. 2313–2328.

[62] Gurkan Tuna, TarikVeli Mumcu, Kayhan Gulez, VehbiCagri Gungor, and Hayrettin Erturk. "Unmanned Aerial Vehicle-Aided Wireless Sensor Network Deployment System for Post-disaster Monitoring". In: *Emerging Intelligent Computing Technology and Applications*. Vol. 304. Communications in Computer and Information Science. July 2012, pp. 298–305.

[63] Amanda Hodgson, Natalie Kelly, and David Peel. "Unmanned aerial vehicles (UAVs) for surveying marine fauna: a dugong case study". In: *PloS one* 8.11 (Nov. 2013), e79556.

[64] Pablo Chamoso, William Raveane, Victor Parra, and Angélica González. "UAVs Applied to the Counting and Monitoring of Animals". In: *Ambient Intelligence - Software and Applications*. Vol. 291. May 2014, pp. 71–80.

[65] Mustafa I. Akbas, Gürkan Solmaz, and Damla Turgut. "Molecular Geometry Inspired Positioning for Aerial Ad Hoc Networks". In: *Computer Networks (Elsevier)* 98 (2016), pp. 72–88.

[66] Mustafa Ilhan Akbas, Gürkan Solmaz, and Damla Turgut. "Actor positioning based on molecular geometry in aerial sensor networks". In: *Proc. IEEE Int'l Conf. on Communications (ICC-2012)*. Ottawa, Canada, June 2012, pp. 508–512.

[67] Matthias R. Brust, Mustafa I. Akbas, and Damla Turgut. "VBCA: A Virtual Forces Clustering Algorithm for Autonomous Aerial Drone Systems". In: *Proceedings of IEEE SysCon 2016*. Apr. 2016.

[68] NYC Taxi Limousine Commission. *Taxi and Limousine Commission (TLC) Trip Record Data*. URL: `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`.

[69] Gustavo Niemeyer. *Tips & Tricks about geohash*. 2008. URL: `http://geohash.org/site/tips.html`.

[70] Geoffrey J McLachlan and Kaye E Basford. *Mixture models: Inference and applications to clustering*. Vol. 84. New York: Marcel Dekker, 1988.

[71] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. "Pixel recurrent neural networks". In: *Proc. IEEE ICML*. June 2016, pp. 1747–1756.

[72] Hugo Larochelle and Iain Murray. "The Neural Autoregressive Distribution Estimator". In: *Proc. IEEE AISTATS*. Apr. 2011, pp. 29–37.

[73] Bart Van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio. "Blocks and fuel: Frameworks for deep learning". In: *arXiv preprint arXiv:1506.00619* (2015).

[74] The Theano Development. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv preprint arXiv:1605.02688* (2016).

[75] Pedro Lopez-Garcia, Enrique Onieva, Eneko Osaba, Antonio D. Masegosa, and Asier Perallos. "A hybrid method for short-term traffic congestion forecasting using genetic

algorithms and cross entropy". In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (2016), pp. 557–569.

[76] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. "Traffic flow prediction with big data: a deep learning approach". In: *IEEE Transactions on Intelligent Transportation Systems* 16.2 (2015), pp. 865–873.

[77] Menglong Yang, Yiguang Liu, and Zhisheng You. "The reliability of travel time forecasting". In: *IEEE Transactions on Intelligent Transportation Systems* 11.1 (2010), pp. 162–171.

[78] Wengen Li, Jiannong Cao, Jihong Guan, Shuigeng Zhou, Guanqing Liang, Winnie K. Y. So, and Michal Szczecinski. "A General Framework for Unmet Demand Prediction in On-Demand Transport Services". In: *IEEE Transactions on Intelligent Transportation Systems* (2018), pp. 1–11.

[79] Philippe C. Besse, Brendan Guillouet, Jean-Michel Loubes, and Francois Royer. "Destination Prediction by Trajectory Distribution-Based Model". In: *IEEE Transactions on Intelligent Transportation Systems* 19.8 (Aug. 2018), pp. 2470–2481.

[80] Xianyuan Zhan, Samiul Hasan, Satish V. Ukkusuri, and Camille Kamga. "Urban link travel time estimation using large-scale taxi data with partial information". In: *Transportation Research Part C: Emerging Technologies* 33 (2013), pp. 37 –49.

[81] Jun Xu, Gürkan Solmaz, Rouhollah Rahmatizadeh, Damla Turgut, and Ladislau Bölöni. "Internet of Things Applications: Animal Monitoring with Unmanned Aerial Vehicle". In: *CoRR* (2016). arXiv: `1610.05287`.

[82] Orr Spiegel, Wayne M. Getz, and Ran Nathan. "Factors influencing foraging search efficiency: why do scarce lappet-faced vultures outperform ubiquitous white-backed vultures?" In: *The American Naturalist* 181.5 (May 2013), E102–E115.

[83] Tao Shu and Marwan Krunz. "Coverage-Time Optimization for Clustered Wireless Sensor Networks: A Power-Balancing Approach". In: *IEEE/ACM Transaction on Networking* 18.1 (Feb. 2010), pp. 202–215.

[84] Ronald A Howard. "Information value theory". In: *IEEE Transactions on Systems Science and Cybernetics* 2.1 (1966), pp. 22–26.

[85] Richard Bellman. *A Markovian decision process*. Tech. rep. DTIC Document, 1957.

[86] Christopher J.C.H. Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (May 1992), pp. 279–292.

[87] Richard S. Sutton and Andrew G. Barto. "Reinforcement learning: An introduction". In: Cambridge: MIT press, 1998. Chap. Monte Carlo Methods.