

**SEMI-LAZY LEARNING APPROACH TO
DYNAMIC SPATIO-TEMPORAL DATA ANALYSIS**

ZHOU JINGBO

(B. Eng., Shandong University, China)

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING
NATIONAL UNIVERSITY OF SINGAPORE

2014

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

zha Jingbo

Zhou Jingbo

04 August 2014

Acknowledgement

I would like to acknowledge all the people who have provided support, advice, suggestions, guidance and help during my time as a graduate student in the School of Computing, National University of Singapore.

First and foremost, my sincerest gratitude goes to my supervisor, Prof. Tung Kum Hoe, Anthony, for his continuous support of my study and research. Prof. Tung is a brilliant, ingenious and smart professor, who is always able to provide inspiring and innovative ideas. His vast knowledge, various skills in many areas, plentiful experience about the research, and persistent guidance helped me throughout the duration of my research.

The work in this thesis is the result of collaboration with my coauthors who are Gang Chen, Sai Wu, Wei Wu and Wee Siong Ng. All of them are my seniors and mentors. I am especially grateful to Chang Fanxi Francis who generously shared valuable datasets to me for the study in the thesis. I would also like to give many thanks to Prof. Tay Yong Chiang and Dr. Bao Zhifeng who got me involved in another interesting research topic, that gave me precious research experience and system development practice.

Prof. Tan Tiow Seng deserves my special appreciations, for teaching me a lot of things, especially when I was a freshman of NUS. I profited from listening to such a wise man. I would like to thank Dr. Huang Zhiyong, Dr. Shen Li and Dr. Fong Wee Teck Louis who generously hosted me for a 2-month internship in the Institute for Infocomm Research (I²R) of A*Star.

I cannot give more thanks to my lab mates and friends for all the help and support from them and for all the fun we have had in the last five years, which will become a wonderful memory in my mind, forever.

Last but not least, my deepest love is reserved for my family, my mother Zhang Chuanfang, my father Zhou Zhanhua, my sister Zhou Leping, my grandmothers

Wang Xiulan and Wang Bingying, and my grandfathers Zhou Chuanwen and Zhang Renlu, for all their unconditional love and spiritual encouragement. And most of all, my special thanks go to my girl friend for her inspiration and support.

Contents

Acknowledgement	i
Summary	v
List of Publications	vii
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Background and Motivation	1
1.2 Challenge of Dynamic Spatio-Temporal Data Analysis	4
1.3 Semi-Lazy Learning Approach	7
1.4 Research Scope and Contributions	9
1.5 Thesis Outline	13
2 Preliminaries and Related work	15
2.1 Distance Function	16
2.2 Trajectory Prediction	21
2.3 Time Series Prediction	24
2.4 Itinerary Recommendation	26
3 Probabilistic Path Prediction in Dynamic Environments	28

3.1	Introduction	28
3.2	Overview and preliminaries	31
3.3	The Trajectory Grid and the Update Process	35
3.4	The Lookup process	39
3.5	The Prediction Filter and the Construction process	40
3.6	Experiments	49
3.7	System demonstration	58
3.8	Conclusions	60
4	Time Series Prediction for Sensors	62
4.1	Introduction	62
4.2	Overview	66
4.3	DTW kNN search with the GPU	70
4.4	Time series prediction via “semi-lazy” learning	84
4.5	Experiments	91
4.6	Comparison of R2-D2 and SMiLer	102
4.7	Conclusions	107
5	Dynamic Itinerary Recommendation for Traveling Services	108
5.1	Introduction	108
5.2	Overview	113
5.3	Pre-processing	116
5.4	Initialization-Adjustment algorithm	123
5.5	Experiments	132
5.6	Conclusions	142
6	Conclusions	144
6.1	Summary	144
6.2	Future work	147
	Bibliography	150

Summary

With a wide range of applications, spatio-temporal data analysis has been a timely and popular research topic in recent years. In this thesis, we investigate problems concerning dynamic spatio-temporal data analysis. The term “*dynamic*” can be interpreted from two perspectives. First, the underlying model generating spatio-temporal data is dynamic. Second, the analysis requirement is dynamic with respect to users’ diverse preferences.

Data analysis methods can be categorized into two classes: the eager learning approach and the lazy learning approach. However, none of the existing approaches are able to achieve eligible performance that is suitable for dynamic spatio-temporal data analysis. Most of the studies in data analysis focus on the eager learning approach. Nevertheless, as we will expound later, the eager learning approach fails to take the “dynamic” factor into account, which precludes its successful application in dynamic spatio-temporal data analysis. Although the literature on the lazy learning approach has shed some light on dynamic spatio-temporal data analysis, the lazy learning approach has been subjected to considerable criticism due to its undesirable performance.

The main aim of this thesis is to propose a new approach to dynamic spatio-temporal data analysis. In this regard, after carefully cogitating how the features of the eager learning and lazy learning approaches could influence analysis performance, we perceived, to our pleasure, that their strong points and weak points are just complementary. Hence, it would be highly imperative and persuasive to adopt their strong points to contrive a new approach. Consequently, we devised a novel “*semi-lazy*” learning approach which can take the “dynamic” factor into account in a similar fashion to the lazy learning approach and still keep good analysis functions like the eager learning approach.

Based on the semi-lazy learning approach, we exploited three concrete dy-

dynamic spatio-temporal data analysis problems, which are trajectory prediction, time series prediction and itinerary recommendation respectively. In summary, the specific objectives of this thesis are to:

- give an extensive study of the “*semi-lazy*” learning approach to dynamic spatio-temporal data analysis. The principal intuition behind inventing the semi-lazy learning approach is to empower the lazy learning approach to achieve eager learning-like analysis functions, while still preserving the benefits of both the lazy learning and eager learning approaches. We employ this approach to investigate three spatio-temporal data analysis problems, which are trajectory prediction, time series prediction and itinerary recommendation respectively.
- propose a semi-lazy approach to trajectory prediction in dynamic environments that builds a prediction model on the fly, using dynamically selected reference trajectories. A trajectory prediction demonstration prototype has been built to show the effectiveness and efficiency of our method.
- devise a time series prediction system for many sensors by exploiting the semi-lazy learning approach. Our system reveals a complete solution for tackling difficulties in time series prediction due to the dynamic properties of sensor data.
- design a dynamic itinerary recommendation system based on the semi-lazy learning approach. Instead of generating ready-to-use itineraries in a pre-processing stage like the eager learning methods do, our method is to dynamically recommend itineraries based on users’ preferences on the fly.

List of Publications

Some materials in this thesis are adapted from the following list of our previous publications:

- **Jingbo Zhou**, Anthony K. H. Tung, Wei Wu, Wee Siong Ng; “A Semi-Lazy Approach to Probabilistic Path Prediction in Dynamic Environments”; *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Pages 748-756, 2013 [140] ¹
- **Jingbo Zhou**, Anthony K. H. Tung, Wei Wu, Wee Siong Ng; “R2-D2: a System to Support Probabilistic Path Prediction in Dynamic Environments via Semi-Lazy Learning”; *Proceedings of the VLDB Endowment VLDB Endowment (PVLDB)*, Volume 6 Issue 12, Pages 1366-1369, 2013 (Demo paper) [141]
- Gang Chen, Sai Wu, **Jingbo Zhou**, Anthony K. H. Tung; “Automatic Itinerary Planning for Traveling Services”; *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, Volume 26 Issue 3, Pages 514-527, 2014 [26]

¹The citation appears in the bibliography at the end of this thesis.

List of Tables

3.1	Table of Notations for Chapter 3	33
3.2	Data sets of Chapter 3	49
3.3	Parameter settings for experiment of Chapter 3	50
3.4	Response time of R2-D2 with confidence threshold $\theta = 0.2$	55
3.5	Response time VS. d_{mic} on ST data set	56
3.6	Size of TG on ST data set	57
4.1	Table of Basic Notations of Chapter 4	66
4.2	Default parameter for experiment of Chapter 4	92
4.3	Effect of Enhanced Lower Bound. The “time” (in seconds) is the total time for all sensor, and the “number” is the number of unfiltered candidates per query per sensor.	95
5.1	Experiment Settings for Chapter 5	134

List of Figures

1.1	An illustration of the spatio-temporal data. If the “atom” is location, we name the spatio-temporal data as <i>trajectory</i> ; if the “atom” is observation value, we name it as <i>time series</i> ; if the “atom” is places of interest (POI), we name it as <i>itinerary</i> . We also use <i>time sequence</i> to refer the general case of the spatio-temporal data	2
1.2	General framework of the “semi-lazy” learning approach	7
1.3	Framework of the “semi-lazy” learning approach: (a) “semi-lazy” framework in trajectory prediction; (b) “semi-lazy” framework in time series prediction; (c) “semi-lazy” framework in itinerary recommendation	9
2.2	Euclidean Distance	18
2.3	An illustration of time shifting: d_c in Q is quite different from d_{cin} D . ¹	18
2.4	(a) An illustration for warping matrix with warping width and warping path; (b) Result alignment according to warping path. ¹ .	19
3.1	An application example of R2-D2 in vehicle path prediction	30
3.2	Path prediction based on reference trajectories.	30

3.3	The detailed architecture of R2-D2. It has an “Update” process (solid lines) and a “Prediction” process (dash lines). “Prediction” process has two sub-process: “Lookup” process and “Construction” process.	31
3.4	The framework of the semi-lazy learning approach to R2-D2.	32
3.5	An example of using R2-D2 to predict O^p 's path.	35
3.6	Overall structure of the Trajectory Grid.	35
3.7	Density update.	38
3.8	Example of traHash and its stored trajectory.	38
3.9	State generation at $t = t_0 + k\Delta t$	43
3.10	Comparison with competitors.	52
3.11	Effect of confidence threshold θ	53
3.12	Self-correcting continuous prediction: <i>ratio</i> < 1 indicates R2-D2 with self-correcting is better than R2-D2 without self-correcting.	54
3.13	Time profiling of R2-D2 with confidence threshold $\theta = 0.2$	56
3.14	Effect of parameters.	56
3.15	Screenshot of the main interface	59
4.1	An illustration for the semi-lazy learning approach to time series prediction	64
4.2	Overview framework of SMiLer	64
4.3	Overview framework of SMiLer, which has a Search Step (input A and output B) and a Prediction Step (input B and output C).	69
4.4	An overview of the Multiple k NN search on the SMiLer Index	73
4.5	An illustration for SMiLer Index	74
4.6	Reuse SD-Table of the SMiLer Index	83
4.7	An illustration of a Gaussian Process with predicted mean and variance	89

4.8	Time cost (log-scaled) of the Multiple k NN Search on all sensors with varying numbers of nearest neighbors k .	94
4.9	Time cost (log-scaled) of k NN search with varying query length d	95
4.10	MAE and MNLDP with varying h -step ahead prediction	98
4.11	Effect of ensemble and self-correction prediction	99
4.12	Total time cost of SMiLer (search and predict) on all sensors	100
4.13	Comparison of PSGP and SMiLer-GP:average training time per sensor of PSGP and their MAE	101
4.14	Distance error of R2-D2 with confidence threshold $\theta = 0$	104
4.15	Distance error of R2-D2 with confidence threshold $\theta = 0.05$	104
4.16	Prediction rate of R2-D2 with confidence threshold $\theta = 0.05$	105
5.1	A 4-Day Trip to Hong Kong	109
5.2	Framework of the semi-lazy learning approach to itinerary recommendation.	111
5.3	The Detailed architecture of dynamic itinerary recommendation system.	112
5.4	POI Graph	113
5.5	Itinerary Index	120
5.6	Example of Set-Packing	124
5.7	Yahoo POIs	133
5.8	User Reviews	133
5.9	Preprocessing Cost	135
5.10	Scalability of Preprocessing	135
5.11	Size of Single Day Itinerary	135
5.12	Indexing Cost	135
5.13	Scalability of Indexing	136
5.14	Size of Index	136
5.15	Effect of Graph Size (Processing Time)	137

5.16	Effect of Graph Size (Quality)	137
5.17	Effect of Selected POIs (Processing Time)	138
5.18	Effect of Selected POIs (Quality)	138
5.19	Effect of Traveling Time (Processing Time)	139
5.20	Effect of Traveling Time (Quality)	139
5.21	Effect of Adjustment (Processing Time)	140
5.22	Effect of Adjustment (Quality)	140
5.23	Buffer Size	140
5.24	Effectiveness of Single Hotel Selection	141
5.25	User study	141

Chapter 1

Introduction

1.1 Background and Motivation

With the rapid development of ubiquitous computing, wireless sensor networks and mobile computing technologies, spatio-temporal data has been pervasive in real-life applications. Thus, in order to exploit broad applications of this new dataset, spatio-temporal data analysis becomes an increasingly important research theme.

1.1.1 What is spatio-temporal data

Examples of spatio-temporal data include temperature readings for sensors, trajectories for people or animals, travelling locations for tourists and even videos from surveillance cameras. Formally, we give the following definition of spatio-temporal data (an illustration of the definition is shown in Figure 1.1):

Definition 1.1.1 (Spatio-temporal data). *The spatio-temporal data is a collection of sequences of timestamped “atoms”, where one “atom” records information observed at a specific timestamp.*

In this thesis, according to different types of the “atom”, we also label different types of spatio-temporal data with the following names:

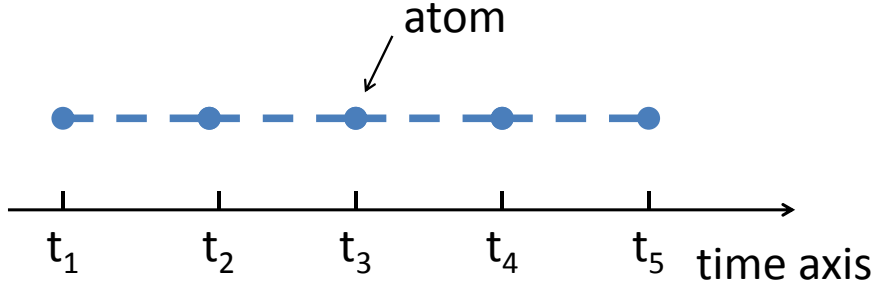


Figure 1.1: An illustration of the spatio-temporal data. If the “atom” is location, we name the spatio-temporal data as *trajectory*; if the “atom” is observation value, we name it as *time series*; if the “atom” is places of interest (POI), we name it as *itinerary*. We also use *time sequence* to refer the general case of the spatio-temporal data

Definition 1.1.2 (Specific spatio-temporal data type).

- **Trajectory**, which is a sequence of timestamped locations. Since a sequence of locations defines the movement of an object, we also name the trajectory as “*path*” to be consistent with the idiomatic expression.
- **Time series**, which is a sequence of timestamped observation values from a sensor (or more generally, an unknown system).
- **Itinerary**, which is a sequence of timestamped places of interest (POIs). An itinerary represents a detailed plan for a journey.

Hereafter, for convenience, we also use “**time sequence**” (or “**sequence**” for short) to represent the general case of the spatio-temporal data.

1.1.2 Research problems and motivation

Since the spatio-temporal data analysis covers many problems, in this thesis, we narrow down our research into three concrete and practical problems corresponding to the data types in Definition 1.1.2, which are:

- **Trajectory prediction**, which is to predict a sequence of locations of a moving object in future time. For more details, please refer to Section 1.4.1

and Chapter 3.

- **Time series prediction**, which is to predict reading values of sensors in future time. For more details, please refer to Section 1.4.2 and Chapter 4.
- **Itinerary Recommendation**, which is to recommend a set of itineraries to satisfy users' traveling demands. For more details, please refer to Section 1.4.3 and Chapter 5.

The main objective of spatio-temporal data analysis is to make “predictions”, which can further facilitate and benefit users' decisions. According to the predictive analysis target, we can expound the spatio-temporal data analysis from two perspectives: data-oriented analysis and user-oriented analysis. Trajectory prediction and time series prediction are considered data-oriented analysis problems, while itinerary recommendation is considered a user-oriented analysis problem.

The objective of the data-oriented analysis is to predict the next, or several, “atoms” of a time sequence in the future time. The data-oriented predictive analysis can provide perceptive insight for the potential associations of a hidden system generating the spatio-temporal data. Hence, this predictive analysis output can guide users to make better decisions for candidate transactions. Taking the trajectory prediction as an example, if we can predict that a car will pass a restaurant, the restaurant advertisement can be sent to the driver of the car ahead of time. In this case, trajectory prediction can improve the quality of actionable advertisement. Moreover, trajectory prediction still has many other applications such as navigation, traffic management, personal positioning, epidemic prevention [74], event prediction [107], anomaly detection [23; 78] and even spatial query optimization [30].

The objective of the user-oriented analysis is to predict the preference that the user would give to an item of the spatio-temporal data, i.e. recommendation. The user-oriented predictive analysis can reduce the difficulty that users encounter when making decisions, especially for inexperienced users with abun-

dant, and even inexhaustible, decision choices. For example, there are many possible locations to visit, whereas travelers want to maximize their travel experience without wandering around. This makes the decision of itineraries for a trip very complicated and troublesome. In this regard, the itinerary recommendation can significantly facilitate users' travel to an unfamiliar place [106; 32; 129; 138; 130].

1.2 Challenge of Dynamic Spatio-Temporal Data Analysis

While the spatio-temporal data carries abundant information and knowledge which is useful for a variety of applications, the new data analysis approach dedicated to spatio-temporal data deserves in-depth treatment due to the unique “dynamic” property of the spatio-temporal data.

The “dynamic” property of the spatio-temporal data analysis can be interpreted from the perspectives of data-oriented analysis and user-oriented analysis. First, from the perspective of data-oriented analysis, the process generating the spatio-temporal data is dynamic. The spatio-temporal data is usually generated by observing some complicated and sophisticated systems which may be evolving with time, affected by many irregular external incidents, and influenced by stochastic internal factors. For example, in urban space, the movement of objects (e.g., cars) is affected by some uncertain and aperiodic factors such as traffic signals, road congestion and weather conditions; therefore, the trajectories of cars in urban space are surely dynamic.

Second, from the perspective of user-oriented analysis, the analysis requirement on the spatio-temporal data is also dynamic. For an analysis task, different users may have contrary requirements with the same application. Moreover, even for the same person, preferences may be varied with different timings and scenarios. For example, to recommend itineraries to travellers, we should consider the user's

preferred places, duration and traveling budget.

Much energy has been devoted to developing new data mining technologies for spatio-temporal data analysis, which can be categorized into two classes: the eager learning approach and the lazy learning approach. The eager learning approach puts significant effort into a training process to construct machine learning models, and then uses these models for analysis tasks when the need arises. The representative eager learning models include the Support Vector Machine (SVM), Artificial Neural Network (ANN) and Decision Tree – just to name a few. In contrast, the lazy learning approach simply stores the entire data set and diverts all efforts to the analysis phase, conducting some simple computations on the “nearest neighbour instances” which are similar to the submitted query. The representative lazy learning models include k-nearest neighbors (kNN) regression, and memory-based Collaborative Filtering (for recommendation analysis).

1.2.1 Eager learning approach

The eager learning approach has drawn much attention for spatio-temporal data analysis in recent years. However, there exist several difficulties for this approach due to the “dynamic” property of the spatio-temporal data.

First of all, eager learning models may suffer from the concept drifting problem. The training process for the eager learning models usually demands a high time cost, up to several hours, and even days. However, the inherent dynamic property of the spatio-temporal data dooms the expiration of the machine learning models over time, which is a well-known problem that is also called “concept drifting”. If concept drifting occurs, it can render a model useless, wasting all the computational effort made to construct the model. Certain portions of the model might not even be utilized before it is rendered useless by concept drift.

Second, the eager learning approach is bounded to the information loss problem, which may result in a high potential risk of overfitting or underfitting models. The eager learning models must commit to a single global hypothesis model

that covers the whole spatio-temporal data domain, while the historical spatio-temporal data is completely abandoned after the training process. However, the spatio-temporal data is dynamic and complicated in spirit. Hence, for the eager learning models, the loss of some localizable and detailed information of the data is inevitable, which may be related to the analysis request. In addition, according to a recent study [132], the local behaviour is very important for the convergence of machine learning models.

1.2.2 Lazy learning approach

With the expounding of a spectrum of eager learning analysis technologies in the data mining community, the value of the lazy learning approach has been overlooked in the past few years. One possible reason is that the lazy learning approach incurs a high cost in answering queries with great storage requirements. Nevertheless, this concern should be assuaged with the development of modern architectures for parallel (e.g. Graphics Processing Units [41]) and distributed (e.g. Hadoop platform [120]) computing. Another obstacle hindering the wide use of lazy learning models is the lack of powerful predictive functions, such as probability confidence and theoretical bounded errors of the result.

However, the lazy learning approach also has several unique features that are promising for dynamic spatio-temporal data analysis. In contrast to the eager learning approach, the inherent “concept drifting” problem can be easily sidestepped for the lazy learning approach by simply updating the database. The lazy learning approach can also fully utilize historical data. While the eager learning approach strives to learn a single global model that is only acceptable on average, the lazy learning approach herds many local models to form an implicit global approximation over the whole dataset, which can capture locality and achieve high accuracy when the data is complex and dynamic [135].

1.2.3 Summary

While the eager learning approach suffers the problems of concept drifting and information loss, since it computes a global model before seeing the prediction query, the lazy learning approach suffers from simplistic predicting methods, although it can commit much richer sets of hypotheses (models) from the data. Hence, developing methods that have the strengths but not the weakness of both approaches is highly desirable.

1.3 Semi-Lazy Learning Approach

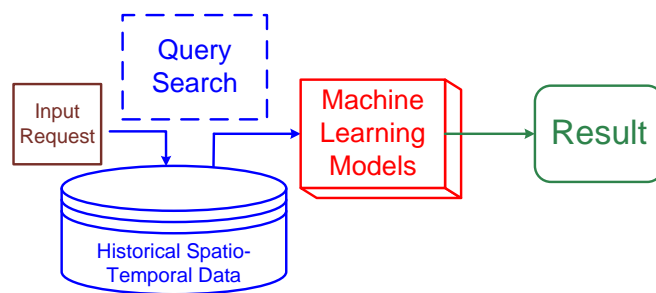


Figure 1.2: General framework of the “semi-lazy” learning approach

In this thesis, we propose a novel and general perspective to spatio-temporal data analysis that offers the benefits of both the eager and lazy learning approaches. We call this new approach the “*semi-lazy*” learning approach.

Figure 1.2 shows the general framework of the semi-lazy learning approach. Our semi-lazy learning approach essentially follows the lazy learning paradigm until the last step, where more sophisticated eager machine learning models are applied on a small set of the searched similar neighbors of the submitted query. In more detail, after receiving a query from a user, we first invoke the search process to retrieve similar neighbors, which are then forwarded to some pertinent machine learning models such as SVM and Neural Network. The models then digest the search results to produce predictive analysis results. To sum up, the semi-lazy approach goes as follows:

1. Like lazy learning, we do not commit to a global model but keep the whole historical spatio-temporal dataset intact.
2. Like lazy learning, given a user input request (typically a data object with some attribute value/s), we first invoke a query search process to retrieve a set of similar neighbors from the historical spatio-temporal dataset.
3. Like eager learning, we apply machine learning models (like SVM, Neural Network or Gaussian Process) on the set of retrieved similar neighbors to derive the predictive analysis result.

The semi-lazy learning approach is superior to the traditional eager learning and lazy learning approaches for dynamic spatio-temporal data analysis from several perspectives. First, the concept drifting problem on dynamic spatio-temporal data can be effortlessly eliminated since we only need to insert new incoming data into the historical data set to reflect irregular changes of underlying patterns over time. Second, there is neither information loss nor the problem of overfitting or underfitting, because we actually build a particular local model for each predictive analysis request on the whole dataset which preserves all information.

Third, this semi-lazy learning approach empowers the traditional lazy learning approach with advanced prediction functions corresponding to the predictive analysis result. For example, in trajectory prediction, we can attach a probability for each predicted trajectory to measure the prediction quality; in time series prediction, we can reckon the confidence interval (standard deviation) for each predicted value; and in itinerary recommendation, we can guarantee a theoretical error bound for the recommended itineraries. All of the above analysis information cannot be provided by a vanilla lazy learning approach.

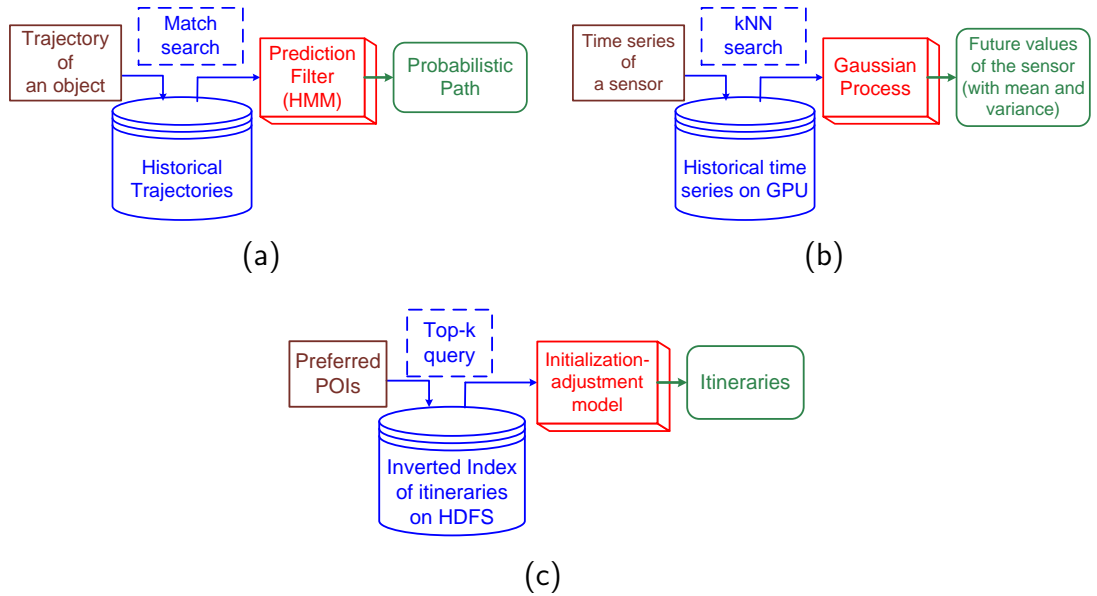


Figure 1.3: Framework of the “semi-lazy” learning approach: (a) “semi-lazy” framework in trajectory prediction; (b) “semi-lazy” framework in time series prediction; (c) “semi-lazy” framework in itinerary recommendation

1.4 Research Scope and Contributions

In this thesis, we have employed the “semi-lazy” learning approach to three practical spatio-temporal data analysis problems mentioned above: trajectory prediction [140; 141], time series prediction and itinerary recommendation [26]. The sketches that show how to apply the “semi-lazy” learning approach to these problems are illustrated in Figure 1.3 and summarised as follows:

- For trajectory prediction, as shown in Figure 1.3(a), we first use the trajectory of the predicted object in the last few time steps as a query input to retrieve similar trajectories from the historical trajectory data set. Following that, these retrieved similar trajectories will be used to construct a prediction model (a generalized Hidden Markov Model, i.e. HMM) to make a probabilistic path prediction.
- For time series prediction, as shown in Figure 1.3(b), we use the time series of a sensor in the last few time steps as the input request, which is submitted

to a Graphics Processing Unit (GPU) to retrieve a set of k-Nearest Neighbor (kNN) time series. The kNN results are then input into the Gaussian Process (GP) model to predict the future value (with mean and variance) of the sensor.

- For itinerary recommendation, as shown in Figure 1.3(c), we use the user’s preferred Points of Interest (POIs) as the input request to select top-k best itineraries for the user from an inverted index of itineraries, which is built on top of the Hadoop Distributed File System (HDFS). Instead of returning the search results directly, we employ an initialization-adjustment model to refine the searched itineraries, followed by returning the adjusted near-optimal itineraries emphasizing the user’s preference.

The following three sections briefly describe the contribution of our three works respectively.

1.4.1 Trajectory prediction

Trajectory prediction, also called path prediction (hereafter, to be consistent with the idiomatic expression, we use the phrases “trajectory prediction” and “path prediction” interchangeably.), is useful in a wide range of applications. Most of the existing solutions, however, are based on eager learning methods where models and patterns are extracted from historical trajectories and then used for future prediction. Since such approaches are committed to a set of statistically significant models or patterns, problems can arise in dynamic environments where the underlying models change quickly or where the regions are not covered with statistically significant models or patterns.

We propose a “semi-lazy” approach to trajectory prediction that builds prediction models on the fly using dynamically selected reference trajectories. Such an approach has several advantages. First, the target trajectories to be predicted are known before the models are built, which allows us to construct models that

are deemed relevant to the target trajectories. Second, unlike the lazy learning approach, we use sophisticated learning algorithms to derive accurate prediction models with acceptable delay, based on a small number of selected reference trajectories. Finally, our approach can be continuously self-correcting, since we can dynamically re-construct new models if the predicted movements do not match the actual ones.

Our prediction model can construct a probabilistic path whose probability of occurrence is larger than a threshold, and which is furthest ahead in terms of time. Users can control the confidence of the path prediction by setting a probability threshold.

Lastly, we build a demonstration prototype incorporating all the techniques proposed above. In the demonstration system, we showcase the above key aspects of our approach, using several real-life trajectory datasets. The system provides a visual interface that shows moving objects and their predicted trajectories. The demonstration system also allows users to play with various parameter settings. An online demo of our system is available at:

<http://db128gb-b.ddns.comp.nus.edu.sg/jzhou/R2-D2/>.

1.4.2 Time series prediction

Time series prediction of sensors has many applications. Growth in computing capability has motivated the use of machine learning solutions for this purpose, and these solutions fall into two categories: eager and lazy learning. Eager learning methods pre-construct statistical models from historical data and then use the models for prediction. In contrast, lazy learning methods keep the historical data unprocessed until performing prediction and then find a subset of historical data with similar behavior (called k -Nearest Neighbors, k NNs) to make prediction by applying some simple computation.

We propose a novel semi-lazy learning approach for time series prediction. Like lazy learning, the semi-lazy learning approach maintains historical data un-

processed until prediction time, but it applies expensive eager learning model construction methods on the k NNs to predict more accurately. While such an approach results in higher computation cost, we argue that advances in computation hardware like GPUs will make such a “just-in-time” model construction feasible for real-time applications like sensor values prediction.

To illustrate our point, we present SMiLer, a time series prediction system for sensors that adopts the semi-lazy learning approach. To make our system feasible, two challenging problems are tackled which are: (I) a fast k NN search method using the popular Dynamic Time Warping (DTW) distance on the GPU and (II) an effective semi-lazy time series prediction model based on Gaussian Processes.

1.4.3 Itinerary recommendation

We design an itinerary recommendation service which can dynamically recommend multi-day itineraries for users. Creating an efficient and economic trip plan is the most annoying job for a backpacking traveler. We propose a novel semi-lazy learning approach to dynamically recommend personalized itineraries for specific users. Our design philosophy is to generate itineraries on the fly, utilizing historical trajectories and users’ preferences, via the semi-lazy learning approach.

Most existing works on itinerary recommendation are based on the eager learning approach, which takes a two-step scheme. They first adopt data mining algorithms to discover users’ traveling patterns from their published itineraries. Based on the relationships of the historical data, new itineraries are generated and recommended to the users. However, these pre-defined itineraries are not tailored for each specific customer. There are some previous efforts to address the dynamic itinerary recommendation problem by providing a dynamic planning service, which organizes the Points-of-Interests (POIs) into a customized itinerary. Because the search space of all possible itineraries is too costly to fully explore, to simplify the complexity, most work assumes that each user’s trip is limited to some

important POIs and will be completed within one day.

We designed a more general itinerary recommendation service, which generates multi-day itineraries for users. We iterate all candidate single-day itineraries using a parallel processing framework – MapReduce. The results are maintained in the DFS (Distributed File System), and an inverted index is built for efficient itinerary retrieval. Given a request, the system provides interfaces for the user to select preferred POIs explicitly. Then we selectively combine the single itineraries to recommend a multi-day itinerary. We designed an approximate algorithm to recommend near-optimal itineraries. The approximate algorithm adopts an initialization-adjustment scheme and a theoretic bound is guaranteed for the approximate result.

1.5 Thesis Outline

The rest of the thesis is structured as follows. In Chapter 2, we first introduce some preliminary knowledge blocks for spatio-temporal data analysis. We also give a thorough literature review for existing works related to the general semi-lazy learning approach, trajectory prediction, time series prediction and itinerary recommendation in this chapter.

In Chapter 3, Chapter 4 and Chapter 5, we show our three pieces of work on dynamic spatio-temporal data analysis, which are trajectory prediction, time series prediction and itinerary recommendation respectively. All of them are based on our proposed semi-lazy learning approach.

In Chapter 6, we conclude this thesis, followed by a discussion about limitations and future research directions for this thesis.

Chapter 2

Preliminaries and Related work

In this chapter, we first overview the existing works closely related to the semi-lazy learning approach in the data mining community. Then we talk briefly about distance function (in Section 2.1) which is a preliminary knowledge point of spatio-temporal data analysis. Lastly, we investigate the literatures related to our specific research problems which are trajectory prediction (in Section 2.2), time series prediction (in Section 2.3) and itinerary recommendation (in Section 2.4).

In this thesis, we propose a new machine learning approach to spatio-temporal data analysis. The general idea is that, instead of estimating a global model which is tolerable for all possible target prediction requests on average, we first find similar neighbors for the target request and then apply a sophisticated machine learning model on the search result to construct a specific model for the target. To our best knowledge, there is no existing work applying this approach to spatio-temporal data analysis. However, it is still desirable to discuss existing works with similar ideas about this approach from the whole scope of the data mining community.

There are indeed several works [45; 63; 103; 118] that also tried to combine the eager learning approach and the lazy learning approach. In particular, the authors of [45] also named their method “semi-lazy”. However, the essential difference is that they usually try to partition the whole training data into several parts, fol-

lowed by building machine learning models on each data part in the pre-processing stage. Then a similarity search algorithm is used to select the best local model for prediction. Therefore, these methods are essentially still eager learning, and hence suffer the drawbacks of the eager learning approach. In addition, another challenging problem is how to properly partition the training data set.

As far as we know, there are only two works [134; 15] which seem similar to our “semi-lazy learning” idea, i.e. retrieving kNN and then building heavy models on the kNN results. Nevertheless, both of these existing works focus on the image classification problem, whereas our study is the first work aimed towards the dynamic spatio-temporal data analysis problem, exploiting the semi-lazy learning approach.

Apart from the data application domain, our semi-lazy learning approach is still distinguished from existing works in two ways. First, we also study the online update problem for the “semi-lazy” learning approach, which is not considered by [134; 15]. Second, existing methods, including the works of not only these two image classification papers [134; 15], but also the former ones [45; 63; 103; 118] (mentioned in second paragraph of this section), did not make much effort to tackle the similarity search problem; whereas we devote an extensive study to efficient similarity search under different spatio-temporal data types, which is another important research contribution of this study.

2.1 Distance Function

The distance function is an essential building block for spatio-temporal data analysis. Given two time sequences T_1 and T_2 , a distance function $dist(\cdot, \cdot)$ calculates the similarity between them, denoted by $dist(T_1, T_2)$. In the past decades, a lot of distance functions have been proposed, such as Euclidean distance, DTW [13], LCSS [116], ERP [27], EDR [28] and SpADe [29], etc. They are summarized in Figure 2.1. We can refer to distance measures that compare the i -th point of one

time sequence to the i -th point of another as lock-step measures (e.g., Euclidean distance and the other L_p norms), and distance measures that allow the comparison of one-to-many points (e.g., DTW) and one-to-many/one-to-none points (e.g., LCSS) as elastic measures [49]. There is no last word on which distance function is more effective; a variety of distance functions have been used under different application domains. In the sequel, we introduce several important functions.

- **Lock-step Measure**
 - L_p -norms
 - L_1 -norm (Manhattan Distance)
 - L_2 -norm (Euclidean Distance)
 - L_{inf} -norm
 - DISSIM
 - **Elastic Measure**
 - Dynamic Time Warping (DTW)
 - Edit distance based measure
 - Longest Common SubSequence (LCSS)
 - Edit Sequence on Real Sequence (EDR)
 - Swale
 - Edit Distance with Real Penalty (ERP)
 - **Threshold-based Measure**
 - Threshold query based similarity search (TQuEST)
 - **Pattern-based Measure**
 - Spatial Assembling Distance (SpADe)

Figure 2.1: A Summary of Similarity Measures ¹

2.1.1 Euclidean distance

An example of Euclidean distance is shown in Figure 2.2. Given two time sequences $Q = \{q_1, q_2, \dots, q_n\}$ and $C = \{c_1, c_2, \dots, c_n\}$, the Euclidean distance between two time sequences is $Eu(Q, C) = \sqrt{\sum_{i=1}^n dist^2(q_i - c_i)}$.

Besides being relatively straightforward and intuitive, Euclidean distance and its variants have several other advantages, which are easy to implement, indexable and parameter-free. Furthermore, the Euclidean distance is surprisingly competitive with other more complex approaches, especially if the size of the database is

¹The figure is adopted from [49].

$$Q = \{q_1, q_2, \dots, q_n\}$$

$$C = \{c_1, c_2, \dots, c_n\}$$


$$Eu(Q, C) = \sqrt{\sum_{i=1}^n dist^2(q_i - c_i)}$$


Figure 2.2: Euclidean Distance

relatively large [49]. However, since the mapping between the points of two time sequences is fixed, these distance measures are sensitive to noise and misalignments due to the local time shifting problem.

Local time shifting refers to a condition wherein similar segments are out of phase. Figure 2.3 shows an example of local time shifting. Note that D is similar to Q at the semantic level, as there is a hump followed by an ascending trend in both of them. However, the lag of an ascending trend to the hump in Q (measured as $d - c$) is different from that (measured as $d' - c'$) in D. This time shifting problem can be overcome by another distance function— Dynamic Time Warping (DTW), which will be introduced in next section.

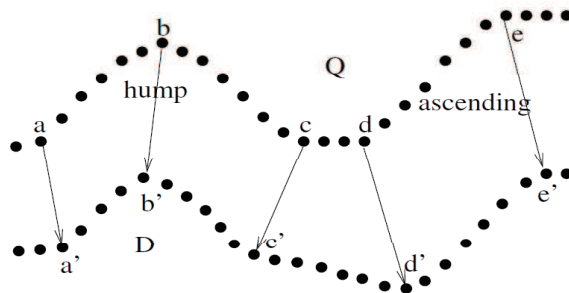


Figure 2.3: An illustration of time shifting: $d - c$ in Q is quite different from $d - c$ in D. ¹

¹The figure is adopted from [29].

2.1.2 Dynamic Time Warping

In this section, we will give a brief review of Dynamic Time Warping (DTW). Inspired by the need to handle time shifting in similarity computation, Berndt and Clifford [13] introduced DTW, a classic speech recognition tool, to the data mining community. The DTW distance allows a time sequence to be “stretched” or “compressed” to provide a better match with other time sequences. The DTW distance is recursively defined as follows:

$$DTW^2(Q, C) = \begin{cases} 0 & |Q| = |C| = 0; \\ \infty & |Q| = 0 \text{ or } |C| = 0; \\ dist(q_1, c_1) + \\ \min\{DTW^2(Rest(Q), Rest(C)), \\ DTW^2(Rest(Q), C), DTW^2(Q, Rest(C))\} & \text{otherwise.} \end{cases}$$

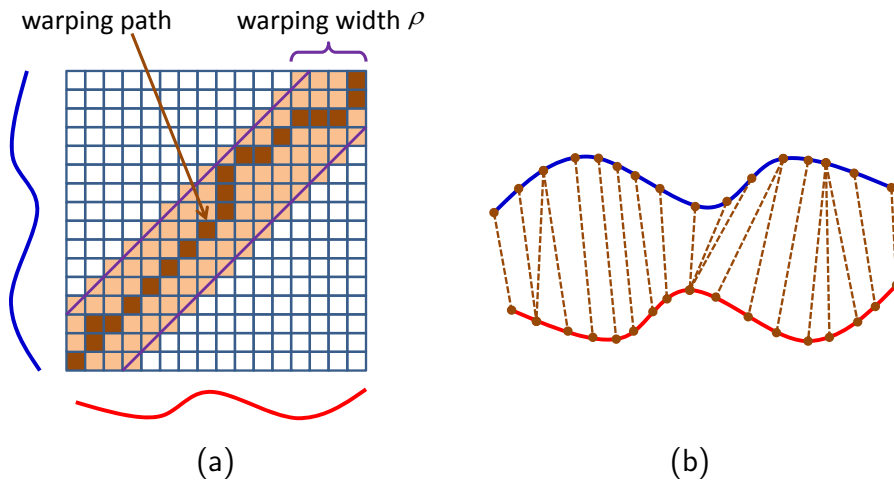


Figure 2.4: (a) An illustration for warping matrix with warping width and warping path; (b) Result alignment according to warping path. ¹

The DTW distance can be computed by dynamic programming with a matrix as shown in Figure 2.4 . We construct a matrix, of which each element (i, j) represents an alignment between q_i and c_j . The warping path W is a continuous

¹The figure is reproduced and modified from [72].

set of matrix elements which represent the optimal alignment between two time sequences. The i -th element of W is defined as $w_i = (w_i^q, w_i^c)$, i.e.:

$$W = \{w_1, \dots, w_i, \dots, w_k\} = \{(w_1^q, w_1^c), \dots, (w_i^q, w_i^c), \dots, (w_k^q, w_k^c)\},$$

$$\max(m, n) \leq k \leq m + n - 1$$

The constraints for W are :

- **Monotonic:** $w_i^q - w_{i-1}^q \geq 0$ and $w_i^c - w_{i-1}^c \geq 0$
- **Continuity:** $w_i^q - w_{i-1}^q \leq 1$ and $w_i^c - w_{i-1}^c \leq 1$

Let denote $|w_i|$ as the cost of w_i , which is given by:

$$|w_i| = \text{dist}^2(q_{w_i^q} - c_{w_i^c})$$

There are many warping paths in the matrix, the goal of DTW is to find the path that minimizes the cost:

$$DTW(Q, C) = \min \sqrt{\sum_{i=1}^k |w_i|}$$

Global constraints on the warping path are always used to reduce the time complexity of DTW [72; 142]. In Figure 2.4 (a), the Sakoe-Chiba band constraint [101] is applied on the warping path which is restricted to not more than ρ cells from the diagonal [72; 94; 7] where ρ is called the warping width. In other words, with the Sakoe-Chiba constraint, the (i, j) element in the warping matrix is ∞ if $|i - j| > \rho$. This band constraint not only reduces the computation cost of DTW, but also avoids the degenerated matchings (e.g. most of elements of a time series are matched to several elements of the other) [7]. In this thesis, if without specification, we only consider the DTW with Sakoe-Chiba band constraint.

2.1.3 More distance functions

Here we give a brief discussion about several other important distance functions. Longest Common Subsequences (LCSS)[116] can handle local time shifting and noise. However, it is not a metric measure. Hence, it is not easy to index LCSS, and its computation cost is high. A lower-bounding measure and indexing technique for LCSS are introduced in [115].

Edit Distance on Real sequence (EDR) [28] can handle the time shifting and noise problem, but it is also not a metric measure. In addition, the authors in [28] proposed three methods to reduce the computation cost.

SpADe [29] is a pattern-based similarity measure for time series. The advantage of SpADe is that it can handle the time shifting, time scaling, amplitude shifting and amplitude scaling, whereas the disadvantage of SpADe lies in the difficulty of tuning the parameters to handle those factors.

There are more than ten distance measures for similarities on spatio-temporal data, we refer interested readers to [49].

2.2 Trajectory Prediction

In this section, we will review several previous trajectory prediction works. Since a trajectory is just a sequence of locations, we also refer to the trajectory prediction as “*location prediction*” or “*path prediction*” hereafter, for convenience. In different application domains, the prediction techniques can be divided as pattern-based prediction and descriptive model-based prediction.

2.2.1 Pattern-based prediction

We future categorize pattern-based prediction methods into two classes: personal pattern-based prediction and general pattern-based prediction.

Personal pattern-based prediction methods consider movement of each object

to be independent. In [127], Yavas et al. propose an algorithm for predicting the next location of mobile users on the basis of mobility pattern mined from the user's trajectories. Jeung et al. [68] propose a hybrid prediction model which combines the motion function and the movement pattern of users. In [100], Sadilek et al. propose a model to support long-term mobility prediction, which leverages Fourier analysis and principal component analysis.

One shortcoming of personal pattern-based prediction methods is that it requires enough personal historical trajectory data (for the object whose path needs to be predicted) being available for pattern mining. This renders such methods not applicable for moving objects that do not have enough personal historical trajectories.

General pattern-based prediction methods use the common mobility patterns to predict the future location of users. In [85; 86], Morzy proposes methods to extract association rules from the moving object database and uses the association rules for prediction. In [128], Ying et al. propose a novel prediction model utilizing both geographic and semantic features of trajectories. Mobility patterns are also used for destination prediction, such as WhereNext [83] and SubSyn [124], which predicts moving objects' destinations without concerning paths of reaching the destinations.

Pattern-based prediction methods do not work well in dynamic environments. The main problem is that patterns mining is an expensive (in terms of time) process, therefore, it is difficult to mine patterns on the fly. As a result, such methods cannot capture new patterns just emerge in the dynamic environment. Furthermore, the movement of certain moving objects may not match with any pattern, which makes the pattern-based methods do not be able to make predictions [128].

2.2.2 Descriptive model-based prediction

Descriptive model-based prediction methods use mathematical models to describe the movement of moving objects. The models can be parameterized, e.g. mo-

tion function models [117; 91; 109; 110; 89], or nonparameterized, e.g. Markov model[67].

Motion function models estimate objects' future locations by their own motion functions. For example, one simple motion function is $l(t_h) = l_0 + v_0 \times (t_h - t_0)$, where l_0 is a starting point, v_0 is the velocity, t_0 is the current time and t_h is the predicted time [117; 91]. Recursive motion function [109] is the most accurate one, which uses the least squared error method to find the best parameters. Motion function does not take environment constraints into account, such as obstacles and other moving objects, and do not model the environmental changes, such as traffic jam and traffic signal.

Markov model-based methods are suitable for estimating future locations of moving objects in a discrete location space. In [67], Ishikawa et al. derive the transition probabilities from one or multiple locations to another. Then the object's next location can be deduced by the Markov model. In [88], Musolesi et al. provide a comprehensive survey of mobility models. The common problem of those models is that they are only able to predict the short-term path of a moving object.

Some stochastic process models, such as Gaussian process[111; 51] and Dirichlet process[70], have also been used to model the movement of objects. However, such models focus on moving objects' aggregate behavior and do not address the location prediction problem.

There are also some path prediction methods that utilize road networks and mobility models [71; 69]. Both [71] and [69] have a pre-processing phase to build mobility models for finding objects' aggregate behavior in a road network. These methods can only be applied in network-constraint spaces.

2.3 Time Series Prediction

Existing time series prediction methods can be separated into two categories, i.e. statistical regression methods and machine learning methods.

2.3.1 Statistical regression method

Statistical regression methods focus on finding parameterized functions that can describe and predict the behavior of time series. Linear statistical methods, such as Autoregressive Integrated Moving Average (ARIMA) models [20], robust regression [121] and their variants [133; 73; 99], have been studied for a long time. However, it becomes increasingly clear that linear models are not adaptable to some practical applications [47]. Several powerful yet rather complex nonlinear time series models, such as bilinear models [93], exponential smoothing [122; 64] and (General) AutoRegressive Conditional Heteroskedasticity ((G)ARCH) models [52; 16], were proposed. We refer interested readers to [20] for more details.

While many fascinating mathematical theorems have been established, these methods usually impose rigid assumptions on the time series data such as bounded, autoregressive or Markovian processes. Another potential drawback of these approaches is that additional user insight of statistical expertise and domain knowledge are required to guarantee the quality of the results.

2.3.2 Machine learning method

Machine learning models can be seen as the extension of statistical regression models which utilize historical data to learn the stochastic dependency between the past and the future of one time series. Machine learning methods can be further grouped into two classes: the eager learning approach and the lazy learning approach.

2.3.2.1 Eager learning approach

The eager learning approach usually has a pre-processing stage to mine models or patterns for time series prediction. This approach has been actively conducted in the last two decades [17]. Several machine learning models have been successfully employed for time series prediction such as Artificial Neural Networks (ANNs) [1; 119; 80], Support Vector Machines (SVMs) [126; 99; 87], Hidden Markov Models (HMMs) [55; 31] and Gaussian Process (GPs) [57; 90; 21; 125].

However, several difficulties exist for machine learning models with the eager learning approach. First, training process of machine learning models usually requires high computational cost (e.g. $O(n^3)$ for SVMs and GPs). Although some approximate methods, such as Sparse Gaussian Process [40] and Core Vector Machine [113], have been proposed, they may fall into overfitting or underfitting traps. To be more specific, the essential idea of these methods, sometimes called low-rank approximation, is to use a small number of well selected examples to approximate the whole dataset. Therefore, these approximate models are more influenced by the global distribution of the whole dataset rather than by the local behaviour of unknown prediction targets. It leads to a potentially high level of overfitting or underfitting [103]. Besides, machine learning methods can also suffer from the problem of concept drift. The hidden generating processes of the time series may change over time in unforeseen ways and the quality of prediction results may gradually degenerate. However the models cannot be easily updated due to the high computational cost.

2.3.2.2 Lazy learning approach

In contrast to the eager learning approach, the lazy learning approach simply stores the entire dataset and diverts all effort to the prediction phase. Prediction for future values of time series is done by finding a set of k similar time series of the sensor and doing some simple computation (e.g. average) over these query

results [82; 24; 14; 9].

Lazy learning methods however lack powerful predictive functions. For example, the k NN regression cannot estimate the predictive uncertainty (i.e. a confidence interval) of the predicted results in contrast to most of the eager learning methods. Moreover, the accuracy of the lazy learning approach still needs to be further improved.

There are several benefits for the lazy learning approach. First, using many local models to form an implicit global approximation, lazy learning approach can commit to a much richer hypothesis [132]. Second, the concept drifting problem can be effortlessly sidestepped since we only need to update new data into the historical dataset. Third, from a theoretical point of view, the performance of k NN regression tends to be Bayes optimal [38] and stable [19] as the size of the historical dataset tends to be infinity.

2.4 Itinerary Recommendation

Most existing works on itinerary recommendation take an eager learning scheme which has two steps. They first adopt the data mining algorithms to discover the users' traveling patterns from their published images, geo-locations and events [97; 39; 34]. Based on the the relationships of those historical data, new itineraries are generated and recommended to the users [106; 32; 129]. This scheme leverages the user data to retrieve Points of Interest (POIs) and organize the POIs into itinerary. Several recommender systems, such as [136; 137], are also developed to recommend the unvisited POIs to users based on Collaborative Filtering (CF) model [58]. The review of the eager learning recommender system is necessarily brief here; we refer the interested reader to [138] for a more detailed treatment. However, none of these methods can help the traveling agency provide the customized itinerary service, where all details of POIs are known and each user prefers different itinerary instead of adopting the most popular ones.

In fact, searching for the optimal single-day itinerary has been well studied. It can be transformed into the Traveling Salesman Problem (TSP) [36], which is a well-known NP-complete problem. For example, in [62], given a set of POIs, the system will generate a shortest itinerary to access all the POIs. If the distance measure is a metric and symmetric, the TSP has the polynomial approximate solution [33], but the approximate solution incurs high overhead for a large POI graph [76]. Therefore, some heuristic approaches [50] are adopted to simplify the computation.

Some interactive search algorithms [11; 77] are proposed recent years. These algorithms still focus on optimal single-day itinerary planning. To reduce the computation overhead and improve the quality of generated itineraries, users' feedbacks are integrated into the search algorithm. The search algorithm works iteratively. It proposes new itineraries for users based on their previous feedbacks and the users can adjust the weights of POIs in the itinerary or select new POIs into the itinerary. In the next iteration, the algorithm will refine its results based on the collected information. Those work can be considered as variants of optimal single-day itinerary planning problems. Moreover, interactive algorithms pose requirements for the users, who may be reluctant to provide the feedbacks.

Chapter 3

Probabilistic Path Prediction in Dynamic Environments

3.1 Introduction

Trajectory prediction is presently a popular area of research [75; 85; 86; 68; 83; 69; 128; 79; 100]. Predicting future trajectories of moving objects has a broad range of applications, including navigation, traffic management, personal positioning, actionable advertising [83], epidemic prevention [74], event prediction [107], anomaly detection [23; 78] and even spatial query optimization [30]. To be consistent with the idiomatic expression, we use the terms “*trajectory prediction*” and “*path prediction*” interchangeably.

In this chapter, we study the trajectory prediction problem in dynamic environments. An environment is considered to be dynamic if the movement of objects in the environment changes with some uncertain, aperiodic and irregular factors. Some real-life examples of dynamic environments include: (1) urban space where the movement of objects (e.g., cars) is affected by traffic signals, traffic jams and weather conditions; (2) massively multiplayer online games where the movement of game units varies depending on the placement of monsters and resources; (3) shopping malls where the movement of customers changes when certain shops are

on promotions.

Existing path prediction methods mostly adopt the eager learning approach [85; 68; 86; 128; 100], i.e., models or patterns are extracted from historical data and used to predict the future movements of objects. In such methods, historical trajectories are abandoned once the models or patterns are extracted. This results in a loss of information since the models or patterns are usually not fully representative of the data. Furthermore, since the models or patterns are generated without knowing the objects to be predicted, problems arise when the target objects are moving through regions that are not covered with statistically significant models. In extreme cases where the environment is so dynamic that completely new situations can arise (e.g. once in 50 years flash flood or new game settings), the models or patterns may become invalid.

To overcome these problems, we propose a “semi-lazy” learning trajectory prediction model, called **R2-D2**¹, to *pRobabilistic path pReDiction in Dynamic environments*. In R2-D2, historical trajectories are kept and indexed. To perform prediction for a target object, we match its past trajectory against historical trajectories and extract a small set of reference trajectories. In order to retrieve the reference trajectories, we devise an index and a matched-based search method to select the reference trajectories which have similar behavior of the target object. Sophisticated machine learning techniques are then applied on the reference trajectories to construct a local model, which can predict the future movement of the target object.

Figure 3.1 shows an application example of R2-D2 in vehicle path prediction. R2-D2 continuously collects streaming trajectories of a large number of moving objects (the cars in Figure 3.1). In Figure 3.2, the solid line is the trajectory of object O^p whose future path is to be predicted; while the dash lines are the historical trajectories which are selected as reference trajectories for predicting the path of O^p .

¹R2-D2 is a smart robot in the Star Wars.

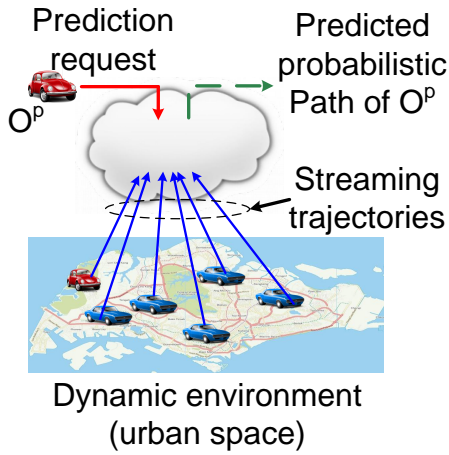


Figure 3.1: An application example of R2-D2 in vehicle path prediction

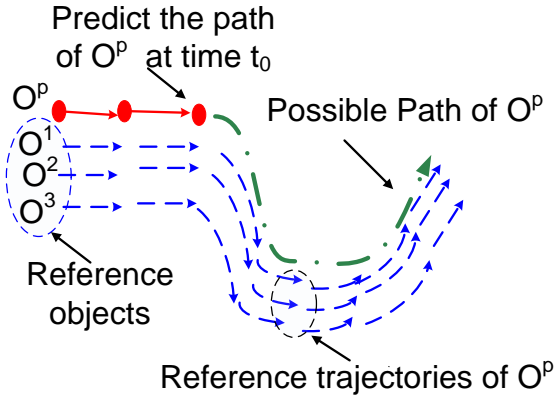


Figure 3.2: Path prediction based on reference trajectories.

R2-D2 has several advantages. First, the target trajectory is known before the model is derived from a set of similar reference trajectories. This ensures that the model is highly relevant. Second, since the learning is performed on a small set of reference trajectories, we can afford to use slightly more complex learning algorithms. Given the power of modern hardware, the time taken to derive such local model is typically acceptable. Finally, since the actual movement of the target object can be compared against the predicted movement subsequently, we can dynamically derive new models if the actual movement and the predicted movement do not match. This leads to a self-correcting continuous prediction approach.

On top of these advantages, R2-D2 also supports probabilistic path prediction. Since there is always a trade-off between the prediction accuracy and the length of the predicted path, R2-D2 chooses to predict the longest path (in terms of time) with probability (confidence) higher than a given threshold. The user can thus use R2-D2 to predict paths of different lengths by setting different confidence thresholds, making it flexible enough to be used in a broad range of applications that have different requirements for prediction confidence and predicted path length.

The rest of the chapter is organized as follows. Section 3.2 formalizes the prob-

lem and gives an overview of R2-D2. The prediction process of R2-D2 has three sub-processes: “Update”, “Lookup” and “Construction”, which are discussed in Section 3.3, Section 3.4 and Section 3.5, respectively. We evaluate R2-D2 in Section 3.6 and conclude the chapter in Section 3.8.

3.2 Overview and preliminaries

3.2.1 Overview

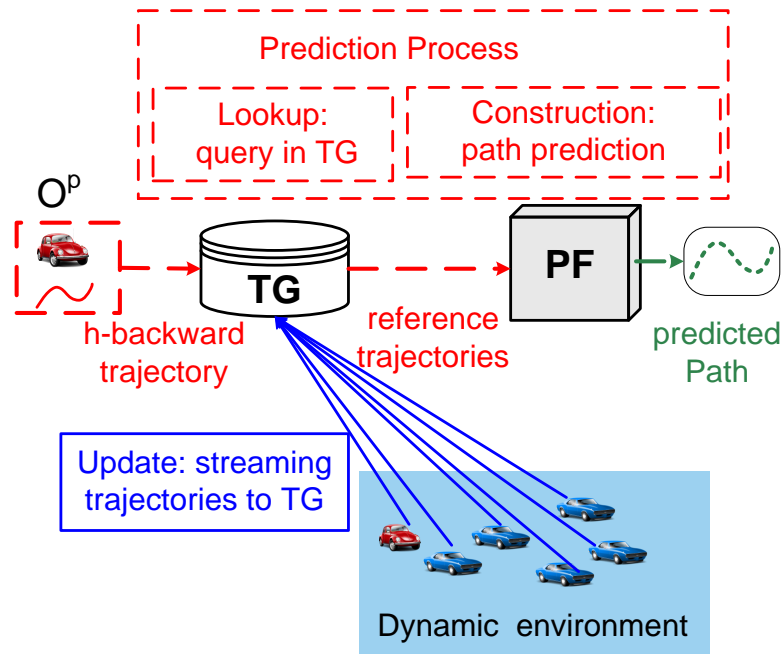


Figure 3.3: The detailed architecture of R2-D2. It has an “Update” process (solid lines) and a “Prediction” process (dash lines). “Prediction” process has two sub-process: “Lookup” process and “Construction” process.

Figure 3.3 illustrates the detailed architecture of R2-D2. It has two components: the Trajectory Grid (TG) and the Prediction Filter (PF). TG is a grid based indexing structure for storing historical trajectories. PF performs probabilistic path prediction. Figure 3.4 shows that how R2-D2 fits in the framework of our semi-lazy learning approach.

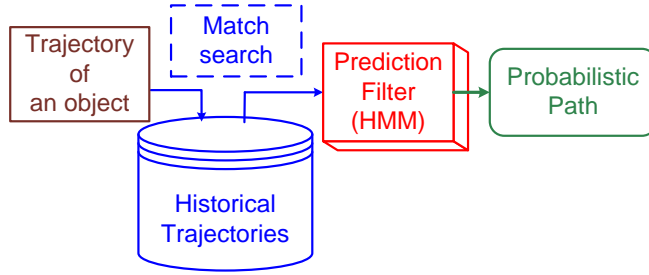


Figure 3.4: The framework of the semi-lazy learning approach to R2-D2.

There are also two separate processes: “Update” and “Prediction”. In Figure 3.3, we denote the “Update” process by **solid lines** and denote the “Prediction” process by **dash lines**. The “Update” process continuously collects streaming trajectories from the dynamic environment and stores them in TG (see **Section 3.3**).

The “Prediction” process makes path prediction. This process has two sub-processes: “Lookup” and “Construction”. In Figure 3.3, we want to predict the path of O^p (the red car). In the “Lookup” process, we use the trajectory of O^p in the last few time steps as query trajectory to retrieve reference trajectories from TG (see **Section 3.4**). R2-D2 will only make a prediction if the number of reference trajectories is greater than a predefined threshold. In the “Construction” process, the reference trajectories are used to construct a model for making path prediction (see **Section 3.5**).

3.2.2 Preliminaries

Table 3.1 lists notations used throughout this chapter. We use O^i to denote a moving object. A trajectory of O^i is a sequence of timestamped locations $T^i = \langle O_1^i, \dots, O_t^i, \dots \rangle$, where $O_t^i = ((x, y), t)$ denoting that O^i locates at location (x, y) at time t .

We assume that all trajectories have synchronised timestamps. When this assumption is not valid, we interpolate the trajectories with a linear algorithm which will be specified later. It is worth noting that the interpolation does not affect the result too much since previous study on the interpolation of trajectories [112]

Table 3.1: Table of Notations for Chapter 3

O^i	moving object	O^p	object to be predicted
O_t^i	location of O^i at time t	RO	reference objects of O^p
t_0	current time	C_t^i	credit of O^i at time t
T^i	trajectory of O^i	W_h	h -backward trajectory
ζ^i	a set of reference points	ζ_l^i	ζ^i in level l of cluster tree
s_k	state of O^p at $t_0 + k\Delta t$	S_k	state space at $t_0 + k\Delta t$
R	macro cluster radius	d_{mic}	micro cluster radius
θ	confidence threshold	ξ_m	support for ‘‘Construction’’
RO_k	reference points of O^p at time $t_0 + k\Delta t$		
$RO_{1:k}$	all reference points of O^p from $t_0 + \Delta t$ to $t_0 + k\Delta t$		
$SS_{1:k}$	a path of O^p from $t_0 + \Delta t$ to $t_0 + k\Delta t$		
$s_{k,l}^i$	the i -th state in level l of state space tree at time $t_0 + k\Delta t$		
CR_t	query range of O^p at time t for reference objects		
H	TG buffer interval threshold for dropping old trajectories		

shows that the accuracy of interpolated locations was always within the accuracy of the tracking method used. In other words, the errors of interpolated locations were always smaller than the distance that the moving objects are potentially able to travel during the average time elapsed between recorded locations [112].

We refer to the trajectory of O^p in the last h time steps as **h-backward trajectory** and denote it by $W_h = \langle O_{t_0-(h-1)\Delta t}^p, O_{t_0-(h-2)\Delta t}^p, \dots, O_{t_0}^p \rangle$ where t_0 is the current time.

The **reference objects** of O^p , denoted as RO , is a set of objects which have certain sub-trajectories matched with W_h . These objects have a similar tendency as O^p . We give a formal definition of the match function in Section 3.4.

For each object $O^i \in RO$, we denote $O_{t_v}^i$ as the timestamped location of O^i that is nearest to $O_{t_0}^p$. In other words, t_v is the timestamp when object O^i 's location is closest to $O_{t_0}^p$. The **reference points** of O^p at t_0 are defined as $RO_0 = \{O_{t_v}^i | O^i \in RO\}$. The **reference points** of O^p at $t_0 + k\Delta t$, namely RO_k , are defined as $RO_k = \{O_{t_v+k\Delta t}^i | O^i \in RO\}$. Note that the definition of RO_k is based on RO_0 , because the value of t_v for each $O^i \in RO$ is determined at time t_0 . Moreover, we use $RO_{1:k}$ to denote all reference points of O^p from $t_0 + \Delta t$ to $t_0 + k\Delta t$, i.e.,

$RO_{1:k} = \bigcup_{i=1}^k RO_i$. We also call $RO_{1:k}$ as **reference trajectories** of O^p .

Let $\odot(\textit{centroid}, \textit{radius})$ denote a circle. We call a vector $s_k = (\odot(\textit{centroid}, \textit{radius}), k)$ a **state** of O^p , which means O^p may be within the circle at time $t_0 + k\Delta t$. Since there could be several possible states for O^p at time $t_0 + k\Delta t$, we use s_k^i to denote a possible state whose identifier is i . We define the set of all possible states of O^p at time $t_0 + k\Delta t$ as a **state space**, i.e., $S_k = \bigcup s_k^i$. We denote a sequence of states of O^p from $t_0 + \Delta t$ to $t_0 + k\Delta t$ as $SS_{1:k} = \langle s_1, s_2, \dots, s_k \rangle$. We call $SS_{1:k}$ a **path** of O^p .

Given $RO_{1:k}$, we denote the probability that O^p is in state s_k as $p(s_k|RO_{1:k})$. Similarly, given $RO_{1:k}$, we denote the probability that O^p would appear in every state in $SS_{1:k}$ as $p(SS_{1:k}|RO_{1:k})$.

Based on the above definitions, the probabilistic path prediction problem can be formally defined as:

Definition 3.2.1 (Probabilistic Path Prediction). *Given a moving object O^p and a probability threshold θ at time t_0 , probabilistic path prediction returns a path $SS_{1:k}$ of length k time steps, which satisfies:*

(1) $p(SS_{1:k}|RO_{1:k}) \geq \theta$;

(2) for any path with length $k + 1$ time steps we have:

(a) $p(SS_{1:k+1}|RO_{1:k+1}) < \theta$, **OR** (b) $k + 1 > k_{max}$ where k_{max} is a predefined maximum length. (The additional constraint k_{max} is to avoid the length of the predicted path being infinite.)

Example 3.2.1. *In Figure 3.5, there are five moving objects: $\{O^p, O^1, O^2, O^3, O^4\}$.*

TG stores all their trajectories (see Section 3.3). At time $t_0 = 11$, we want to predict the future path of O^p . First, we use the 2-backward trajectory of O^p , i.e. $W_2 = \langle O_{10}^p, O_{11}^p \rangle$, to retrieve reference objects from TG (see Section 3.4), which are $RO = \{O^2, O^3, O^4\}$.

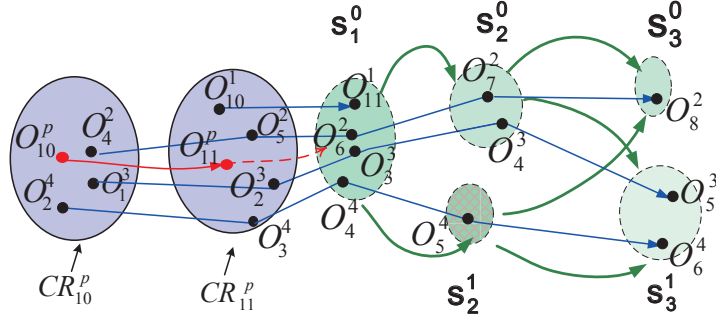


Figure 3.5: An example of using R2-D2 to predict O^p 's path.

Then, PF estimates a future path of O^p in two steps. First, at each future time $t = t_0 + k\Delta t$, we generate the state space S_k of O^p (see Section 3.5.2). For example, when $k = 2$ we have $S_2 = \{s_2^0, s_2^1\}$. A sequence of states is a possible path. Next, PF selects the longest path whose probability is larger than the probability threshold θ . If there are multiple such paths, the one with the highest probability is selected (see Section 3.5.1). In this example, if the user sets $\theta = 0.4$, the predicted path is $SS_{1:2} = \langle s_1^0, s_2^0 \rangle$. If $\theta = 0.2$, the predicted path is $SS_{1:3} = \langle s_1^0, s_2^0, s_3^0 \rangle$.

3.3 The Trajectory Grid and the Update Process

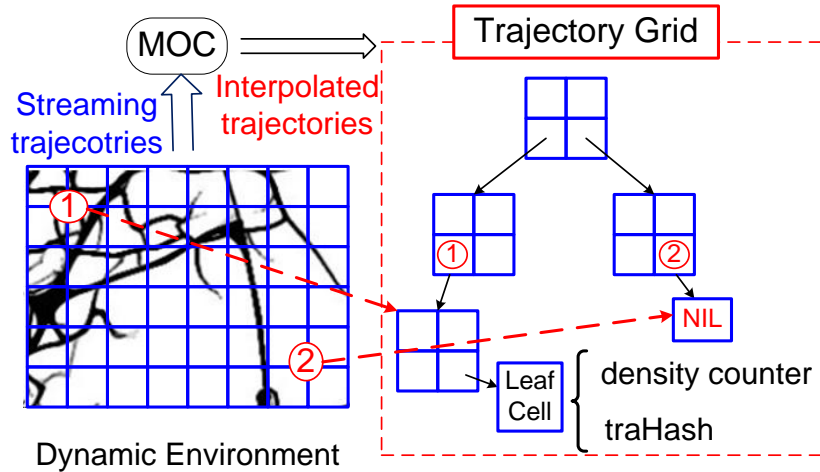


Figure 3.6: Overall structure of the Trajectory Grid.

The Trajectory Grid (TG) is a multi-level grid structure, which dynamically

stores new (recent) trajectories as they emerge. We use a *grid* to divide the area of interest into a set of rectangular regions with fixed width. We refer to a region as a cell. Each cell is uniquely identified by a discrete coordinate (x, y) describing the position of the cell in the grid. Hereafter, we use $cell(x, y)$ to denote the cell. Figure 3.6 shows the overall structure of TG. Each leaf node in TG corresponds to one cell. In the experiment part, we will discuss how to set the width of cells. If the trajectories do not have synchronised timestamps, we use a cache structure, called Moving Object Cache (MOC), to interpolate the trajectories. In the following sections, we will discuss each component of the Trajectory Grid in details.

3.3.1 Multi-level grid

TG is organized as a multi-level grid which can handle the skew distribution of data. In real-life datasets, there may be many empty areas where no moving object visits. Using multi-level grid, such area can be compressed with a high level empty grid cell. In Figure 3.6, we show an example of thus multi-level grid. There is a lot of moving object trajectories in region ①. Therefore, region ① will be divided into small cells to leaf cell of the multi-grid. Whereas, the region ② is almost empty(visited by few of moving objects), in this case, we do not sub-divide the region ②, and let it point to a *NIL*.

3.3.2 The Moving Object Cache

The Moving Object Cache is used to interpolate the streaming trajectories. As we can see from Figure 3.6, the streaming trajectories are input into MOC first, and then the interpolated trajectories are output from MOC to TG. We assume all trajectories have the same sample rate, which is not a real limitation since the user can easily re-interpolate the data if the sample rate is changed. Thus, we design a device to handle the interpolation for streaming trajectories before they are inserting into TG.

The interpolation should not have much effect on the prediction result. In previous study [112], there is evidence showing that the accuracy of interpolated locations was always within the accuracy of the tracking method used. In other words, with the linear interpolation method, the errors of interpolated locations are always smaller than the distance that the moving objects are potentially able to travel during the average time elapsed between recorded locations [112].

MOC is a in-memory hash table. The entry of MOC is $\langle key, value \rangle$, while the *key* is the identifier of a moving object O^{id} , and the *value* is a tuple $\langle l_0, t_0 \rangle$ where t_0 is last update timestamp and $l_0 = \langle x_0, y_0 \rangle$ is the corresponding location of the moving object at time t_0 . We suppose the fixed update interval is Δt . When there is an update of a moving object at time t_1 , we first check actual update time interval, i.e. $t_{int} = t_1 - t_0$. If $t_{int} < \Delta t$, we just ignore this update. If $t_{int} \geq \Delta t$, we generate a virtual update point by linear interpolation:

$$\begin{aligned} loc_v &= \frac{l_1 - l_0}{t_1 - t_0} \cdot \Delta t + l_0 \\ t_v &= t_0 + \Delta t \end{aligned} \tag{3.1}$$

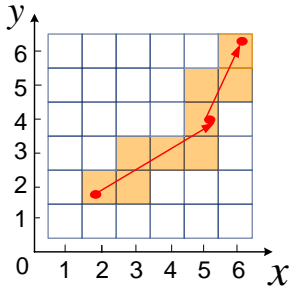
We loop the interpolation process until $t_v \geq t_1$. For each interpolation, we get a line segment $\langle l_0, t_0 \rangle, \langle l_v, t_v \rangle$. Then we transfer l_0 and l_v into their corresponding grid cells $cell(x_0, y_0)$ and $cell(x_v, y_v)$, followed by inserting them into corresponding leaf cells of TG which will be expounded in the following section (Section 3.3.3).

3.3.3 Structure of the leaf cell

In each cell, we record two pieces of information: density and trajectories passing the cell. The density of a cell indicates the popularity of the cell, i.e., how often the cell is visited. The density provides prior information for the ‘‘Prediction Filter’’ (see Equation (3.15)). Each cell in TG has a density counter. If a moving object visits the cell, we increase its density counter by 1. We also update the density of

all the cells along the interpolated line (see Figure 3.7).

Each cell uses a hash table, called **traHash**, to store the trajectories passing the cell. The key of traHash is a vector (O^i, t) , which means O^i passes this cell at time t . The value of traHash is a vector (x, y) , which is the coordinate of the *next* cell that O^i passes. In this way, we implicitly store moving objects' trajectories in the cells' hash tables. Knowing O^i in $cell(x, y)$ at time t enables us to retrieve its following trajectory after t (see Figure 3.8).



traHash of cell(2,2)	
key	value
...	...
(O^1, t)	(5,4)

traHash of cell(5,4)	
key	value
...	...
$(O^1, t+\Delta t)$	(6,6)

traHash of cell(6,6)	
key	value
...	...
$(O^1, t+2\Delta t)$	(9,10)

Actual trajectory of O^1 : $(1.8, 1.6, t) \rightarrow (5.4, 3.9, t+\Delta t) \rightarrow (6.3, 6.2, t+2\Delta t)$
 Approximate stored trajectory of O^1 : $(2, 2, t) \rightarrow (5, 4, t+\Delta t) \rightarrow (6, 6, t+2\Delta t)$

Figure 3.7: Density update.

Figure 3.8: Example of traHash and its stored trajectory.

Example 3.3.1. In Figure 3.7, O^1 is in cell(2, 2) at time t , then it moves to cell(5, 4) at time $t + \Delta t$. We insert a record into the traHash of cell(2, 2) with key (O^1, t) and value (5, 4). Similarly, we insert the records into cell(5, 4) and cell(6, 6). If we know O^1 is in cell(2, 2), we can recursively retrieve the approximate trajectory of O^1 after time t (see Figure 3.8).

TG always stores moving objects' trajectories in the most recent H time units. We denote H as TG buffer interval threshold. In the experiment part, we evaluate and discuss how to determine the value of H . When the moving objects report their new locations, TG updates the relevant cells' density and traHash, and at the same time, TG also checks the oldest element in the traHash (traHash is implemented as *linked hashtable*, therefore, the oldest element is always at the end of the linked list). TG then discards the expired elements of traHashes and updates density counters.

3.4 The Lookup process

We describe how to retrieve reference objects, from which we can easily derive reference trajectories. The general idea is that if O^i has a sub-trajectory that matches with the h -backward trajectory of O^p , we say O^i is a reference object of O^p .

The match function defines similarity between trajectories pairs by a boolean function. Given trajectory T^i and trajectory T^j , we have $match(T^i, T^j) = true$ if $dist(T^i, T^j) \leq \tau$, where $dist()$ is a distance function. Besides the high cost for building index to support such match function, it is also not easy to determine the threshold τ . We define a new match function which has clear semantic meanings and can support high performance query in TG. The definition of match function is as below ($dist()$ is the Euclidean distance function):

Definition 3.4.1 (Match Function). *Suppose $T_n^i = \langle O_1^i, \dots, O_n^i \rangle$ is a sub-trajectory of O^i and $W_h = \{O_1^p, O_2^p, \dots, O_h^p\}$ is the h -backward trajectory of O^p , then we have $match(T_n^i, W_h) = true$ if they satisfy:*

$$(c1) \quad dist(O_1^i, O_1^p) \leq \epsilon \text{ and } dist(O_n^i, O_h^p) \leq \epsilon;$$

$$(c2) \quad \forall O_u^p \in W_h \exists O_v^i \in T_n^i (dist(O_v^i, O_u^p) \leq \epsilon);$$

$$(c3) \quad \forall O_{u1}^p \in W_h \forall O_{u2}^p \in W_h (u1 < u2 \Rightarrow (\exists O_{v1}^i \in T_n^i \exists O_{v2}^i \in T_n^i (dist(O_{v1}^i, O_{u1}^p) \leq \epsilon \wedge dist(O_{v2}^i, O_{u2}^p) \leq \epsilon \wedge v1 < v2)))$$

Intuitions behind the definition are as follows: (c1) requires T_n^i and W_h to be close to each other; (c2) requires every point in W_h to be matched with a point in T_n^i ; (c3) requires O^i and O^p to have the same direction and tendency.

In TG, a query is processed in three steps: **(s1)** for each $O_u^p \in W_h$, we define a range $CR_u = \odot(O_u^p, \epsilon)$; **(s2)** we obtain objects that have visited any CR_u ; **(s3)** the objects visited all the CR_u by the time increasing order are reference objects.

One crucial problem is how to determine a proper value for ϵ . The rule we use is to multiply the moving objects' average velocity by half of the sampling time

interval of the trajectories. For example, if the average velocity of the moving objects is $11.1m/s$ and the sampling time interval is 30 seconds, then $\epsilon = 11.1 * 15 \approx 160(m)$.

Example 3.4.1. *In Figure 3.5, the reference objects of O^p are $RO = \{O^2, O^3, O^4\}$, which visit both CR_{10} and CR_{11} . The reference points of O^p at $k = 0$ are $RO_0 = \{O_5^2, O_2^3, O_3^4\}$.*

3.5 The Prediction Filter and the Construction process

The Prediction Filter (PF) is a model for path prediction. The input of PF is a set of reference points and the output of PF is a predicted path. The ‘‘Construction’’ process, which iteratively constructs state spaces and makes the path prediction, runs within PF. We first give an introduction of PF and the probabilistic path prediction in Section 3.5.1. Then, we introduce a hierarchical method to generate state spaces in Section 3.5.2. We then explain some important functions used in PF in Section 3.5.3. Finally, we discuss self-correcting continuous prediction in Section 3.5.4.

3.5.1 The Prediction Filter and probabilistic path prediction

PF is based on the Grid-based Filter model [6], which is a generalization of Hidden Markov Model. Recall that s_k is a state in state space S_k , we denote the probability distribution function of S_k by $p(s_k|RO_{1:k})$, where $RO_{1:k}$ are observations of state s_k . PF constructs $p(s_k|RO_{1:k})$ recursively, i.e., from $p(s_{k-1}|RO_{1:k-1})$ to $p(s_k|RO_{1:k})$, which is computed by the following function [6]:

$$p(s_k|RO_{1:k}) = \sum_i w_{k|k}^i \delta_i(s_k) \quad (3.2)$$

where

$$w_{k|k-1}^i \triangleq \sum_j w_{k-1|k-1}^j p(s_k^i | s_{k-1}^j) \quad (3.3)$$

$$w_{k|k}^i \triangleq \frac{w_{k|k-1}^i p(RO_k | s_k^i)}{\sum_j w_{k|k-1}^j p(RO_k | s_k^j)} \quad (3.4)$$

Note that $\delta(\cdot)$ is the Dirac measure function and $w_{0|0} = 1$. We will discuss functions $p(s_k^i | s_{k-1}^j)$ and $p(RO_k | s_k^i)$ in Section 3.5.3. How to generate a state space is discussed in Section 3.5.2.

To find the longest path whose probability is greater than the given threshold θ , we increase the length of the predicted path until its probability is smaller than θ . To realize this, we increase the value of k , and for each k value we find a path $SS_{1:k} = \langle s_1, \dots, s_k \rangle$ whose value of $p(SS_{1:k} | RO_{1:k})$ is maximized and then check whether its probability is still greater than θ .

Let us define a function $\eta_{k-1}(j)$ as follows:

$$\eta_{k-1}(j) = \max_{\langle s_1, \dots, s_{k-2} \rangle} p(\langle s_1, \dots, s_{k-2}, s_{k-1}^j \rangle | RO_{1:k-1})$$

$\eta_{k-1}(j)$ is the highest probability that the path ends with state s_{k-1}^j . By Bayesian inference in Grid-based Filter, we have:

$$\eta_k(i) = \max_j [\eta_{k-1}(j) p(s_k^i | s_{k-1}^j)] p(RO_k | s_k^i) \quad (3.5)$$

Equation (3.5) can be solved by dynamic programming algorithms, such as the Viterbi Algorithm [54]. Algorithm 3.1 lists the pseudo code for the probabilistic path prediction. The result is a predicted path, whose probability (confidence) is larger than θ and whose length (in terms of time) is longest.

Example 3.5.1. *In Figure 3.5, we have $\eta_2(0) = 0.53$, $\eta_2(1) = 0.48$; and $\eta_3(0) = 0.26$, $\eta_3(1) = 0.22$, $\eta_3(2) = 0.17$. Therefore, if $\theta = 0.4$, then $SS_{1:2} = \langle s_1^0, s_2^0 \rangle$; if $\theta = 0.2$, then $SS_{1:3} = \langle s_1^0, s_2^0, s_3^0 \rangle$.*

Algorithm 3.1: Probabilistic Path Prediction Algorithm

input : probability confidence threshold: θ
output: probabilistically predicted path: SS

- 1 $\eta_0(0) = 1, \psi_0(0) = 0, \theta^* = 1, k = 0$
// "Lookup" process, see Section 3.4
- 2 get RO and RO_0 from TG by W_h
- 3 **If** $|RO| < \xi_m$ **Then return** NIL
// "Construction" process, see Section 3.5
- 4 **while** $\theta^* \geq \theta$ **do**
- 5 $k=k+1$
- 6 get RO_k from TG based on RO_{k-1}
- 7 generate state space S_k // Section 3.5.2
- 8 **for** $i=1$ to $|S_k|$ **do**
- 9 $\eta_k(i) = \max_j [\eta_{k-1}(j)p(s_k^i | s_{k-1}^j)]p(RO_k | s_k^i)$
- 10 $\psi_k(i) = \arg \max_j [\eta_{k-1}(j)p(s_k^i | s_{k-1}^j)]$
- 11 $\theta^* = \max_i [\eta_k(i)]$
- 12 Backtrack SS from matrix ψ and return SS

3.5.2 Generate states by a hierarchical method

In this section, we discuss how to generate states of O^p at time $t = t_0 + k\Delta t$. In a nutshell, we cluster the reference points RO_k and then convert each reference points cluster to a state.

We do not use traditional clustering methods (such as K-means or DBSCAN) because it is hard to determine their parameters in our problem. Furthermore, we would like to generate states that cover small areas (i.e., the radius is small) but have high probabilities (i.e., $p(s_k | RO_{1:k})$ is high). Traditional clustering methods are unable to find a compromise between these two contradictory criteria.

We propose a hierarchical method to generate states (see Figure 3.9). Moreover, we define a score function to find local optimal that balances the area size against the probability of the state.

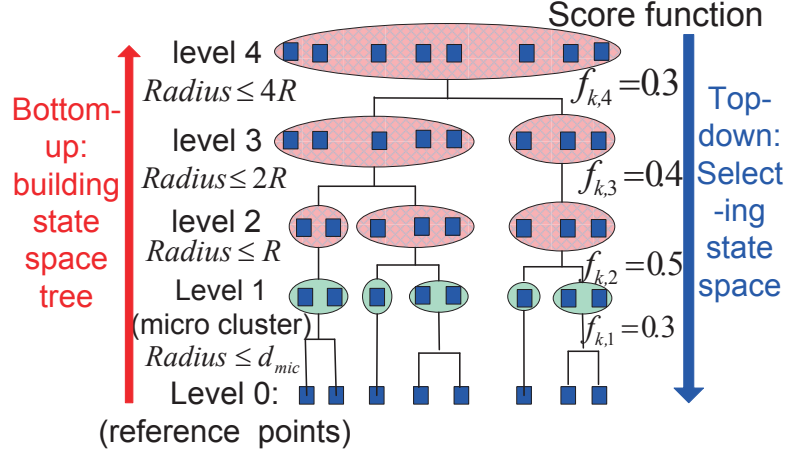


Figure 3.9: State generation at $t = t_0 + k\Delta t$

3.5.2.1 Bottom-up: building state space tree

Now we discuss how to build a state space tree. We use a modified agglomerative hierarchical clustering algorithm to merge the reference points. Then all clusters in each level of the cluster tree are converted to one candidate state space of O^p (which is a set of states, one cluster for one state).

Suppose ζ is a set of reference points in RO_k , i.e., $\zeta \subseteq RO_k$, we can define $Centroid(\zeta)$ and $Radius(\zeta)$ as follows:

$$\begin{aligned}
 Centroid(\zeta) &= \frac{\sum_{O_{t_v}^i \in \zeta} O_{t_v}^i}{|\zeta|} \\
 Radius(\zeta) &= \frac{\sum_{O_{t_v}^i \in \zeta} \|O_{t_v}^i - Centroid(\zeta)\|}{|\zeta|}
 \end{aligned} \tag{3.6}$$

The mean distance between two clusters is defined as :

$$dist_{mean}(\zeta^i, \zeta^j) = \|Centroid(\zeta^i) - Centroid(\zeta^j)\| \tag{3.7}$$

The bottom-up arrow in Figure 3.9 shows the agglomerative clustering process. At level 0, we treat each reference point as a cluster. Then, we continuously merge

a cluster with its nearest cluster by the mean distance, until its radius is larger than a threshold. Then we double the radius threshold, and merge the clusters again. The process repeats until there is only one root cluster. Initially, we set the radius threshold $R = \epsilon$ (ϵ is described in Section 3.4) since ϵ defines a proper size of the area in the environment.

Then, we generate the state space tree. For a cluster ζ_l^i in level l of the cluster tree, we construct a state $s_{k,l}^i = (\odot(\text{Centroid}(\zeta_l^i), \text{Radius}(\zeta_l^i)), k)$. The state space at level l is $S_{k,l} = \bigcup s_{k,l}^i$.

Time complexity. We index reference points by a k-d tree [12]. Suppose the total number of reference points is n . The time cost for building k-d tree is $O(n \log n)$. The time cost for nearest neighbor search in k-d tree is $O(n^{\frac{1}{2}})$. Our agglomerative clustering algorithm is based on [46], and every point needs to merge with others only $O(1)$ times. To sum up, the total time complexity is $O(n^{\frac{3}{2}})$.

3.5.2.2 Top-down: selecting state space

We define a score function to select the state space with the highest score from the state space tree by a top-down manner. All states in one level of the *state space tree* can be considered as a state space; we denote the state space at l^{th} level as $S_{k,l}$.

The score function is to measure the quality of a state space. For a state, we hope it has a high probability ($P(s_k|RO_{1:k})$) and a small radius. However, the states at a higher level of the state space tree have larger probabilities, but also have larger radii; and vice versa. Our objective is to find an optimal compromise between the probability and the radius.

Suppose the state with the largest probability in state space $S_{k,l}$ is $s_{k,l}^*$ (i.e., $\forall s_{k,l}^i \in S_{k,l}, p(s_{k,l}^*|RO_{1:k}) \geq p(s_{k,l}^i|RO_{1:k})$), and $r_{k,l}^*$ is the radius of $s_{k,l}^*$. Our score function is:

$$f_{k,l} = \frac{p(s_{k,l}^*|RO_{1:k})}{(r_{k,l}^*)^\alpha} \quad (3.8)$$

α in Equation (3.8) controls the compromise between the probability and the radius. We will evaluate how to set α . We choose the state space with maximum $f_{k,l}$ as our state space at time $t_0 + k\Delta t$, e.g. in Figure 3.9 we choose the states at level 2 as the state space.

The benefit of the top-down strategy is that we can prune nodes at low levels of the space tree. At one level, if all the states' probabilities are smaller than θ , we stop moving down to lower levels.

3.5.2.3 Improving efficiency by reusing micro cluster

We improve the efficiency of state generation by reusing micro clusters. The basic idea is that objects' locations are changing gradually [108]. By reusing previous clusters at time $t_0 + (k - 1)\Delta t$, we can reduce the time cost for state generation at time $t_0 + k\Delta t$.

A set of reference points in RO_k is defined as a micro cluster mic_k if $Radius(mic_k) \leq d_{mic}$. The set of all micro clusters is denoted as $MIC_k = \bigcup_i mic_k^i$. We try to generate MIC_k based on MIC_{k-1} . The whole process is as follows.

First, we get all reference points RO_k , and divide them into different reference point sets. The reference objects in the same micro cluster of mic_{k-1}^j are also in the same set ζ^j .

Second, for every reference point $O_{t_v}^i \in \zeta^j$, we check whether it is within the circle $\odot(Centroid(\zeta^j), d_{mic})$. If $O_{t_v}^i$ is outside of the circle, it will be split out as a new micro cluster [108]. All the reference points left in ζ^j also form a new micro cluster. We denote all the micro clusters generated in this step as MIC'_k .

Third, we use the bottom-up method to merge micro clusters in MIC'_k , which is the same with the method in Section 3.5.2.1.

Reusing micro clusters reduces the time complexity of the bottom-up process. The time complexity of the first and the second step is $O(n)$. For the third step, we need to perform a nearest neighbor query for each micro cluster in MIC'_k which is indexed by k-d tree. The time complexity of each query is $O(m'^{\frac{1}{2}})$. (m' is the

number of micro clusters in MIC'_k .) As such, the time complexity of the third step is $O(m'^{\frac{3}{2}})$. To sum up, the time complexity of reusing micro clusters is $O(n+m'^{\frac{3}{2}})$. Note that $m' \ll n$.

3.5.3 Functions in the Prediction Filter

3.5.3.1 State transition function

The state transition function $p(s_k^i | s_{k-1}^j)$ represents the probability that O^p will go to state s_k^i at time $t_0 + k\Delta t$ given that O^p is in state s_{k-1}^j at time $t_0 + (k-1)\Delta t$. We model the state transition function by considering two factors: spatial factor and connection factor.

The spatial factor is the extent of spatial overlap between state s_{k-1}^j and s_k^i , and is defined as follows:

$$J(s_k^i, s_{k-1}^j) = \frac{|s_k^i \cap s_{k-1}^j|}{|s_k^i \cup s_{k-1}^j|} \quad (3.9)$$

$|s_k^i \cap s_{k-1}^j|$ denotes the intersection area size of s_k^i and s_{k-1}^j , and $|s_k^i \cup s_{k-1}^j|$ denotes the union area size of s_k^i and s_{k-1}^j .

Connection factor is based on the common reference objects between s_{k-1}^j and s_k^i . We use $RO(s_k^i)$ to denote the reference objects within state s_k^i at $t_0 + k\Delta t$. For example, in Figure 3.5, we have $RO(s_2^0) = \{O^2, O^3\}$. The connection factor is defined as

$$C(s_k^i, s_{k-1}^j) = \frac{|RO(s_k^i) \cap RO(s_{k-1}^j)|}{|RO(s_k^i) \cup RO(s_{k-1}^j)|} \quad (3.10)$$

We combine Equation (3.9) and Equation (3.10) into a linear function:

$$f(s_k^i, s_{k-1}^j) = \lambda J(s_k^i, s_{k-1}^j) + (1 - \lambda)C(s_k^i, s_{k-1}^j) \quad (3.11)$$

where λ is used to adjust the weight of these two factors. In our experiment, we set $\lambda = 0.5$. Initially, we have $J(s_1^i, s_0^0) = 0$ and $C(s_1^i, s_0^0) = \frac{|RO(s_1^i)|}{|RO|}$. We can see

that a larger value of $f(s_k^i, s_{k-1}^j)$ leads to a larger state transition probability. To sum up, we have:

$$p(s_k^i | s_{k-1}^j) \propto f(s_k^i, s_{k-1}^j) \quad (3.12)$$

3.5.3.2 Likelihood function

The likelihood function represents the probability that RO_k is observed given O^p is in state s_k^i at time $t_0 + k\Delta t$. By Bayes' theorem, we have:

$$p(RO_k | s_k^i) = \frac{p(s_k^i | RO_k)p(RO_k)}{p'(s_k^i)} \quad (3.13)$$

$p(s_k^i | RO_k)$ can be computed according to the distribution of RO_k in different states. The more reference objects $O^i \in RO$ are in state s_k^i , the higher value of $p(s_k^i | RO_k)$ is. Then we have:

$$p(s_k^i | RO_k) = \frac{|RO(s_k^i)|}{|RO|} \quad (3.14)$$

$p'(s_k^i)$ is the prior probability that O will be in s_k^i . We assume that O^p has a higher prior probability to enter a state with a higher density in TG. We use $\rho(s_k^i)$ to denote the average density of s_k^i , which is the sum of density of all cells in s_k^i divided by the number of cells covered by s_k^i . Then the prior probability that O^p will be in s_k^i can be expressed as:

$$p'(s_k^i) = \frac{\rho(s_k^i)}{\sum_j \rho(s_k^j)} \quad (3.15)$$

Since $p(RO_k)$ is constant for every state, we have:

$$p(RO_k | s_k^i) \propto \frac{p(s_k^i | RO_k)}{p'(s_k^i)} \quad (3.16)$$

3.5.4 Self-correcting continuous prediction

In real-life applications, we may need to predict the path of a moving object continuously. Continuous prediction gives us an opportunity to improve the prediction result. During the prediction, the actual movement of the target object (O^p) can be compared against the predicted movement. With this information, we can incrementally refine the prediction model. The basic idea is to give more weight to the reference objects which help us make the correct prediction. Let us consider the following example.

Example 3.5.2. *In Figure 3.5, at time $t_0 = 11$, reference objects of O^p are $RO = \{O^2, O^3, O^4\}$. Suppose after 1 time unit, i.e., $t_0 = 12$, O^p goes to the state s_1^0 . At $t_0 = 12$, we have $RO' = \{O^1, O^2, O^3, O^4\}$. Since $\{O^2, O^3, O^4\}$ have helped us make the correct prediction, it is reasonable that we can trust them more than O^1 when we make future prediction.*

Now we introduce a new attribute, called *credit*, for the moving object. Let RO_t denotes the reference objects of O^p at time t . We denote C_t^i ($C_t^i \in \mathbb{N}^0$) as the credit of $O^i \in RO_t$.

We use a *linear growth and exponential decay* method to update moving objects' credits. From t to $t + \Delta t$, the credits are computed as follows: **(s1)** Suppose O^p is in state s_1^j at time $t + \Delta t$. For each $O^i \in RO_t$, if O^i contributes to generate s_1^j , we have $C_t^i = C_t^i + 1$ (*linear growth*); otherwise, $C_t^i = \lfloor \frac{1}{2} C_t^i \rfloor$ (*exponential decay*). **(s2)** Retrieve $RO_{t+\Delta t}$ from TG at time $t + \Delta t$, and initialize the credits of the reference objects as 1. **(s3)** For each $O^i \in RO_t$, if $C_t^i > 0$, put O^i into $RO_{t+\Delta t}$; if O^i is already in $RO_{t+\Delta t}$, sum its credits.

To integrate this self-correcting method into our prediction model, we need two modifications. First, during the state generation (see Section 3.5.2), we use the weighted center of cluster $Centroid'(\zeta_k)$ to replace Equation (3.6):

$$Centroid'(\zeta) = \frac{\sum_{O_{t_v}^i \in \zeta} C_t^i O_{t_v}^i}{\sum_{O_{t_v}^i \in \zeta} C_t^i} \quad (3.17)$$

Second, we integrate credits (as weighted coefficients) into Equation (3.10) and Equation (3.14). For example, we use Equation (3.18) to replace Equation (3.14). Equation (3.10) is modified in the same way.

$$p'(s_k^i | RO_k) = \frac{\sum_{O^u \in RO(s_k^i)} C_t^u}{\sum_j C_t^j} \quad (3.18)$$

3.6 Experiments

We evaluate R2-D2’s performance on real-world and synthetic data sets. We compare R2-D2 with two state-of-the-art prediction methods: RMF [109] and TraP-attern [86]; and study R2-D2’ distinct features, such as the confidence threshold θ and the self-correcting continuous prediction. Efficiency issue is also discussed.

3.6.1 Experiment setup and measurement

Table 3.2: Data sets of Chapter 3

data set name	ST	HT	BT
number of objects	13,200	9,800	25K,50K,100K,200K
unit of time	30sec	1 sec	1 step
width of grid cell	20m-80m(20m)	8 pixels	10 units
buffer interval H	0.5h-3h (1h)	10 min	200 steps

Data sets: Two real-world data sets and four synthetic data sets are used in our experiment, Table 3.2 summaries their details:

[**ST**] is a collection of trajectories of 13,200 taxies in Singapore over one week [123]. Each taxi continuously reports its locations every 20-80 seconds. We interpolate the trajectories over fixed intervals with 30 seconds spacing. Totally, the dataset contains 268 million points.

[**HT**] is a collection of human trajectories tracked from 30 minutes surveillance video in a lobby of a train station [139]. The video is 24 fps with a resolution

Table 3.3: Parameter settings for experiment of Chapter 3

Parameter	Description	range(default value)
θ	confidence threshold	0.2-0.8 (0.2)
d_{mic}	radius of micro cluster (cell width)	1-11 (7)
α	score function	1/32-2 (1/16)
h	backward steps	2-6 (3)

480×720 . Since high sample rate is not necessary, we downsample the timestamp interval of trajectories to 1 second. There are 9,800 trajectories and 159K points.

[BT] We use Brinkhoff generator [22] to generate four collections of synthetic trajectories on the road map of Oldenburg (see Table 3.2). Different from ST and HT, BT has 10 different types of moving objects. The speed of moving objects may change at each time unit. We also put 400 moving obstacles in the space to simulate the changes of the environment. We generate the trajectories for 400 time steps, and each moving object generates one point at each time step. We use the default time units and distance units of the generator. Note that the system performance is mainly affected by the total number of moving objects; therefore, these large synthetic datasets are also used to test the scalability of our model.

Settings: The experiments are conducted on a PC with Intel Q9550 Core Quad CPU 2.83GHz and 3.00GB RAM running Windows XP. All programs are implemented in Java with JDK 1.7. Table 3.3 lists all parameters used throughout the experiments. Parameters are set to default values (**bold font**) unless explicitly specified.

Competitors: R2-D2 is compared with two state-of-the-art algorithms: (1) Recursive Motion Function (RMF) [109] that is a descriptive model-based path prediction method and is the most accurate motion function in literature [109; 68]; and (2) Frequent Trajectory Pattern (TraPattern) [86] that is a general (i.e., not personalized) pattern-based path prediction method. Configurations of RMF and TraPattern are set for their best performance in terms of accuracy by their performance studies. In order to mine patterns for TraPattern, we use data be-

tween 19:00-20:00 from Monday to Friday of ST, use all trajectories of HT and BT.

Measurement: In order to assess the quality of the prediction result, we define four metrics. We use the centers of states as the predicted locations. All errors are measured by Euclidean distance. For a moving object, the **prediction distance error** at a predicted time is the *spatial distance* between the predicted location and the real location of the object. **Maximal distance error** is defined as the maximal *prediction distance error* during the whole predicted time frame. **Prediction length** is the length (time duration) of a predicted path. **Prediction rate** is the fraction of query trajectories for which the model outputs prediction (i.e., the number of predictable query trajectories over the total number of query trajectories). These measurements have been used to assess the quality of prediction models in [83; 68; 69].

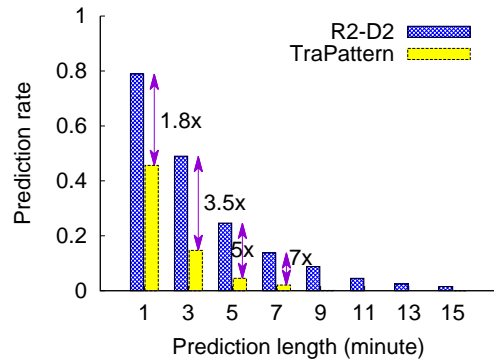
Methodology: For ST data set, we randomly select 200 trajectories between 19:50-20:00 on Tuesday as trajectories for prediction. We warm up TG with the Tuesday trajectories from 17:00-19:50. For HT data set, we randomly select 50 query trajectories within the [16,20] minute interval. We warm up TG with the trajectories within the [0,16] minute interval. For BT data set, we randomly select 1000 query trajectories within the [380,400] interval. We warm up TG with trajectories within the [0-380] interval.

3.6.2 Comparison with competitors

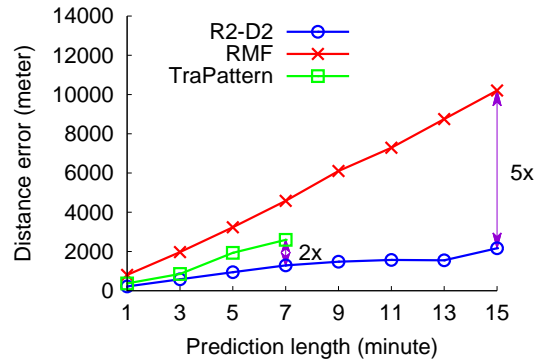
We find that R2-D2 outperforms the competitors in terms of prediction rate and prediction distance error by 2 to 5-fold.

3.6.2.1 Prediction rate

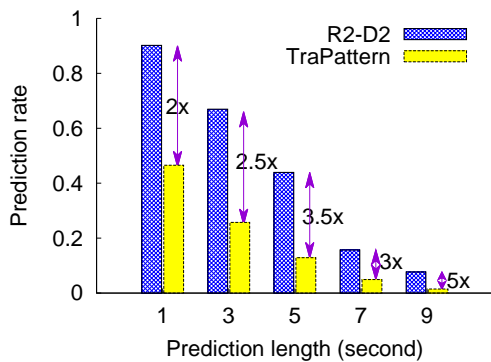
We show the result on prediction rate in Figure 3.10 (a), (c), (e). As the prediction length gets longer, the prediction rate gets lower. However, the prediction rate of TraPattern is much lower than R2-D2 in all prediction lengths. For example, in



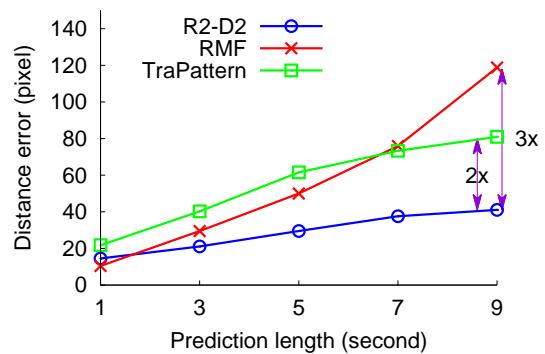
(a) Prediction rate on ST



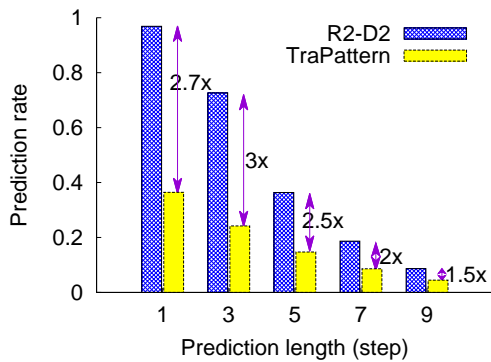
(b) Average prediction distance error on ST



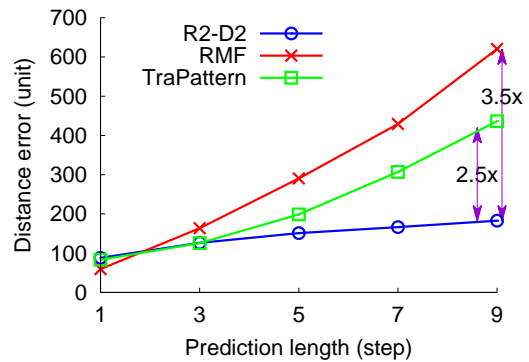
(c) Prediction rate on HT



(d) Average prediction distance error on HT



(e) Prediction rate on BT-200K



(f) Average prediction distance error on BT-200K

Figure 3.10: Comparison with competitors.

Figure 3.10(a), more than half of the predicted paths on ST can be longer than three minutes, while 20% trajectories have predicted paths with seven minutes. Whereas TraPattern cannot give prediction results for more than half of the prediction requests even when the predicted path length is short (e.g. 1 minute for ST). We do not show the prediction rate of RMF in the figures since its prediction

rate is always 100%, but the prediction error may be extremely large as shown in Figure 3.10(b).

3.6.2.2 Average prediction distance error

Figure 3.10 (b), (d), (f) show that R2-D2 has not only higher prediction rate, but also much lower average prediction distance error than those of RMF and TraPattern. In terms of average prediction distance error, R2-D2 outperforms RMF by 5 times on ST, 3 times on HT and 3.5 times on BT-200K; and R2-D2 outperforms TraPattern by 2 times on all data sets. Note that the longest pattern mined by TraPattern is about 8 minutes, therefore, in Figure 3.10(b) the TraPattern cannot predict any path longer than 8 minutes.

3.6.3 Study of R2-D2's distinct features

First, we can see the confidence threshold θ enables users to control the tradeoff between the prediction accuracy and the prediction length. Second, we show self-correcting continuous prediction can reduce the maximal distance error by 50%.

3.6.3.1 Effect of confidence threshold

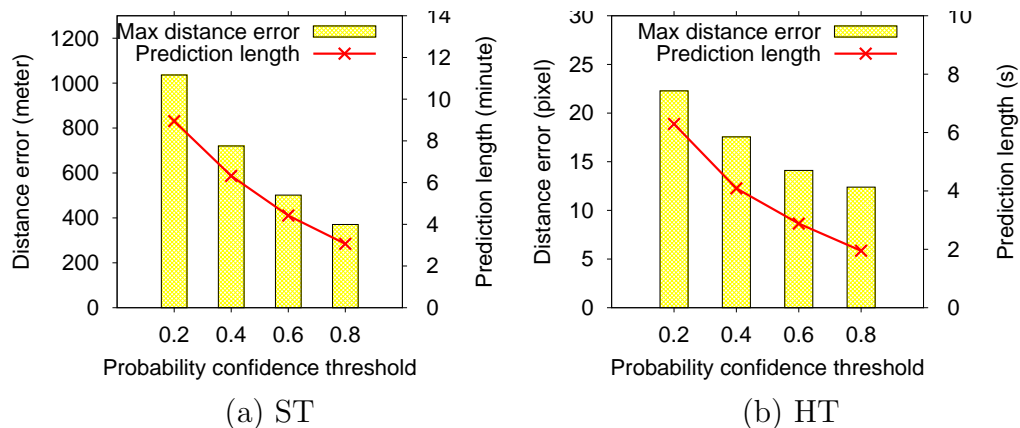


Figure 3.11: Effect of confidence threshold θ

We study the effect of the confidence threshold θ , in particular on the maximal

distance error and the prediction time length. It is desirable to have a low maximal distance error and a long prediction path. However, a longer prediction path typically comes with a larger prediction error. Fortunately, in R2-D2 users can use the confidence threshold θ to control the trade-off between them.

In Figure 3.11, we show the result on ST and HT data sets. We can see that the maximal distance error and the prediction length are statistically correlated with the confidence threshold. When we increase θ , both maximal prediction distance error and prediction length decrease.

3.6.3.2 Self-correcting continuous prediction

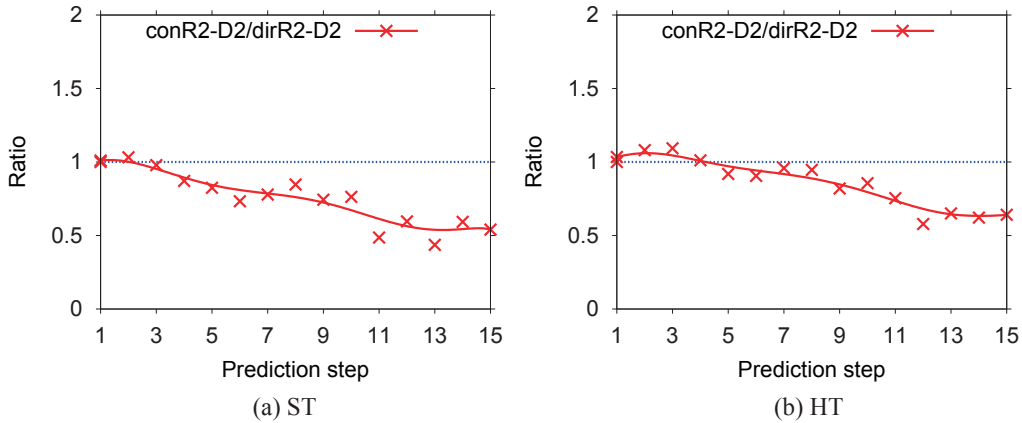


Figure 3.12: Self-correcting continuous prediction: $ratio < 1$ indicates R2-D2 with self-correcting is better than R2-D2 without self-correcting.

We evaluate the effect of self-correcting continuous prediction of R2-D2. For each query trajectory, we perform prediction with a fixed prediction length (next 15 time steps). For each prediction, we do prediction with two different methods: R2-D2 with and without self-correcting continuous prediction (denoted as *conR2-D2* and *dirR2-D2*, respectively). We set the current location to be the location of the object at next one time unit from the previous time, and repeat the same prediction process. For each prediction, we sum the maximal distance errors of *conR2-D2* and *dirR2-D2*, and compute the ratio of them, which is $ratio =$

$\frac{conR2-D2}{dirR2-D2}$. $ratio < 1$ indicates $conR2-D2$ performs better than $dirR2-D2$; and vice versa.

The curves in Figure 3.12 show a clear trend that the performance of $conR2-D2$ improves gradually with the continuous prediction. Since the ratios are noised over time, we use Bezier Curve to fit them over all the prediction steps.

3.6.4 Response time and scalability

We show that R2-D2 makes path prediction in real time. In Table 3.4, we can see the average response time of R2-D2 is only several milliseconds on HT and ST. Even for the largest dataset BT-200K, the response time is still acceptable.

Table 3.4: Response time of R2-D2 with confidence threshold $\theta = 0.2$

data set	ST	HT	BT			
			25K	50K	100K	200K
avg. time (ms)	15.05	8.98	34.89	78.28	112.10	572.50

Figure 3.13 shows the time cost of the Prediction process of R2-D2 on different stages. In Figure 3.13, the “State generation” denotes the running time for generating the states in the Prediction Filter (see Section 3.5.2). Then the “Path generation” denotes the running time for inference the path from the generated states by dynamic programming (see Section 3.5.1 and Algorithm 3.1). From Figure 3.13 we can see that, the Prediction Process (instead of the Lookup Process) takes the major time cost of R2-D2.

We do not show the response time of RMF and TraPattern. Since they only need to compute a math function or match patterns, their response time is faster than R2-D2. Note that we use a prefix tree to compress and index the patterns for TraPattern. Without such index, the time for matching patterns is unacceptable.

Effectiveness of reusing micro clusters: Table 3.5 shows that reusing micro clusters halves the response time. The value of d_{min} means the times of cell width. We can see that when $d_{mic} = 7 \times cell_width$, average response time is

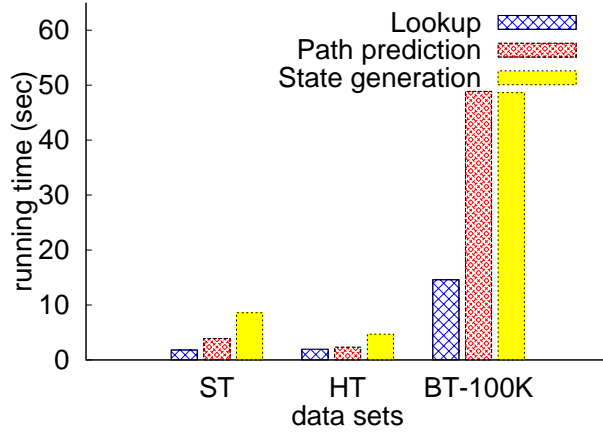


Figure 3.13: Time profiling of R2-D2 with confidence threshold $\theta = 0.2$.

about a half of that when $d_{mic} = 1 \times cell_width$. Note that $d_{mic} = 1 \times cell_width$ means reusing micro clusters is disabled since only points in one cell form a micro cluster.

Table 3.5: Response time VS. d_{mic} on ST data set

d_{mic} (\times cell width)	1	3	5	7	9	11
avg. time (ms)	24.86	19.04	15.75	15.01	13.70	12.78

3.6.5 Effect of parameters

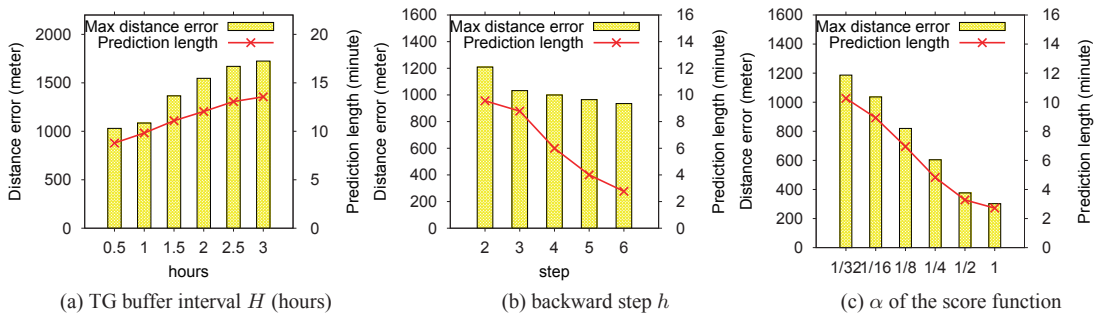


Figure 3.14: Effect of parameters.

We present the effect of parameters in R2-D2 on ST dataset. In Figure 3.14, we use two y-axes: the left one is the maximal distance error and the right one

Table 3.6: Size of TG on ST data set

cell width(m)	size(MB)	max dist err(m)
80	53.6	1693
40	116.9	1280
20	310.0	1020

is the prediction length. It is worth noting that for all datasets we set the same default value of h and α , and they work well.

Cell width of TG: Table 3.6 shows the memory needed by TG and the maximal distance error with different cell width. We can see that using larger cell size can reduce the memory cost, but also lead to larger maximal prediction distance error. The reason is that smaller cell size can better approximate the true distribution of moving objects.

TG buffer time interval H : H determines the length (time steps) of trajectories indexed in TG. From Figure 3.14(a), we can see that both the maximal distance error and the prediction length increase with the increasing of H , but the maximal distance error increases a little faster. When H is larger, TG contains older trajectories, some of which may be misleading when being used as reference trajectories. To balance the maximal distance error against the prediction length, we set H to 1 hour.

Backward steps h : Figure 3.14(b) shows that as h increases, the maximal distance error reduces slightly but the prediction length reduces dramatically. When the number of backward steps is larger, the trajectories of the selected reference objects are more similar to that of the target object, therefore, the maximal distance error is reduced. However, at the same time fewer trajectories are used for prediction, therefore, the prediction time length reduces dramatically. To balance them, we set $h=3$.

α of the score function: We investigate the effect of α of the score function (Equation (3.8)), shown in Figure 3.14(c). A larger α leads to shorter prediction time length and smaller maximal distance error. The reason is that when α is

large the score function always gives a high score to the state space with the small radius; then the maximal distance error is reduced. However, since the state radius is small, the probability of this state is also small. It leads to the fact that the path probability decreases dramatically over time. To balance distance error against prediction time length, we set α to $\frac{1}{16}$.

We also study the effect of other parameters in R2-D2, and find that they do not affect R2-D2’s performance much. For example, if minimum support ξ_m is not set too small (e.g., 3) or too large (e.g., 50), it has little effect on R2-D2’s performance. For all data sets, we set $\xi_m = 10$, and it works well.

3.7 System demonstration

We have implemented a web based demonstration system. In this demonstration, we showcase the above key aspects of the “R2-D2” system using several real-life and synthetic datasets. The system provides a visual interface that shows moving objects and their predicted path. Users will be able to interact with our system by setting different application scenarios with regard to dataset and parameter settings. The online demo of our system is available at:

<http://db128gb-b.ddns.comp.nus.edu.sg/jzhou/R2-D2/>.

3.7.1 System setup

We will demonstrate our system with two real-life datasets and one synthetic data set. The two real-world datasets are (1) the Singapore Taxies (ST) dataset and (2) Human Tracking (HT) dataset. The synthetic dataset is a collection of synthetic trajectories on the road network of Oldenburg generated by Brinkhoff generator (BT). We compare with two existing path prediction methods: Recursive motion function [109] that is the most accurate motion function in literature, and TraPattern [86] that is a general pattern-based path prediction method. More details about the datasets and the competitors are explained in Section 3.6.1.

3.7.2 Demo interface

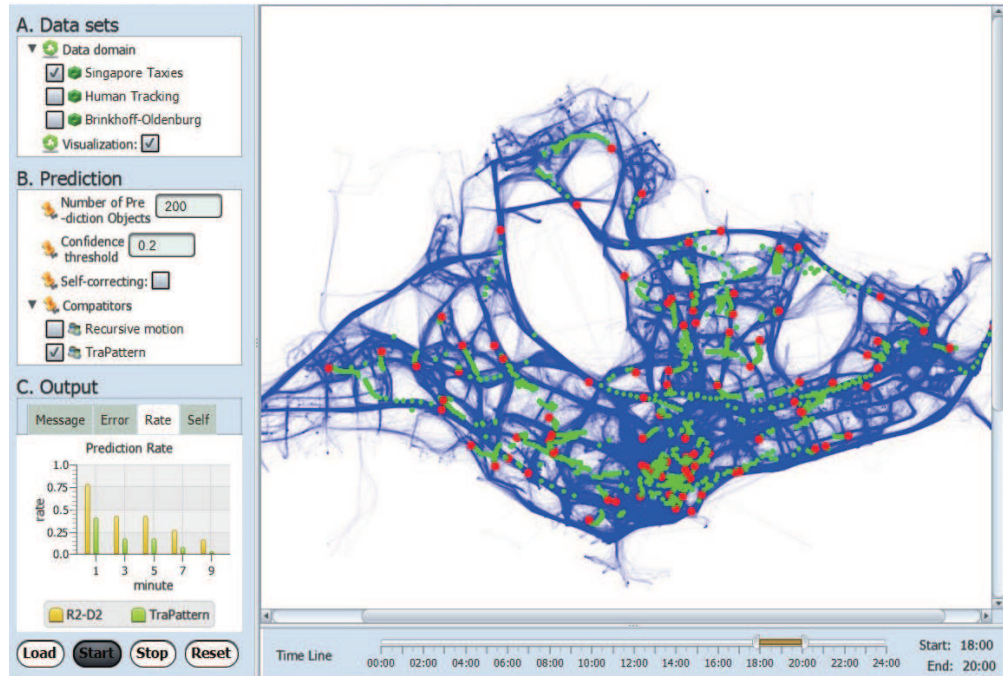


Figure 3.15: Screenshot of the main interface

Figure 3.15 is a screenshot of the demo interface. It consists of two parts: control panel (left part) and display panel (right part). The control panel is composed of four areas from top to bottom: (A) Data sets, (B) Prediction setting, (C) Output and (D) a group of control buttons. In the area (A), users can select the test data sets. In the area (B), the user can set various parameters. In the area (C), the users can see some system output information, such as the different statistics information of our prediction method compared against existing algorithms.

The display panel is composed of two parts from top to bottom: the canvas view and the timeline bar. In canvas view, the user can see the map of visualized trajectory, moving objects and predicted path. In Figure 3.15, the red points represent the target objects whose paths need to be predicted, and the sequences of green dots are the predicted paths. The background blue map is the visualization of trajectories of other objects. The timeline bar lets us set the time interval for selecting datasets.

3.7.3 Utility of the demo

The demonstration of our system provides interactive graphical user interface for users to play with our system, and includes dynamic visualization for users to observe the performance of our path prediction method. With the help of the demonstration, we can easily explain how our system works and show its performance on real world and synthetic datasets.

3.8 Conclusions

In this chapter, we propose a “semi-lazy” approach for performing probabilistic path prediction. Unlike previous approaches adopting eager learning, we propose to leverage on the growth of computing power by building prediction model on the fly, which utilizes historical trajectories that are dynamically selected. Our experiment shows that this self-adaptive “semi-lazy” approach can outperform existing eager learning methods in dynamic environments.

Chapter 4

Time Series Prediction for Sensors

4.1 Introduction

Sensor data is becoming prevalent. Time series prediction of sensors has a vast field of applications including events prediction, air pollution forecasting, manufacturing condition monitoring and medical diagnoses.

Much effort has been devoted to time series prediction by statistical regression analysis, which has a long and rich history [52; 16; 20; 31; 17], probably dating back to a pioneering work in 1927 [131]. Statistical regression methods, such as ARIMA [20], GRACH [16] and robust regression [121], have been well studied. However, it becomes clear over years that these mathematical formulae are not powerful enough to handle the large varieties of time series in all realistic settings [14].

In recent years, machine learning methods have drawn much attention and are becoming popular for time series prediction. This is due to their nonparametric, nonlinear properties and flexible modeling capability. Machine learning methods can be categorized into two classes: the eager learning approach and the lazy learning approach. The eager learning approach first computes statistical models, such as Support Vector Machines (SVMs) [87; 126; 98] and Gaussian Processes (GPs) [57; 90; 125], and then uses these models for prediction when the need arises.

The lazy learning approach, such as k -nearest neighbors (k NN) regression [82; 14], performs prediction during running time by conducting some simple computation (e.g. average) on the “nearest neighbour objects”. However these methods have several weaknesses when applied to sensor time series prediction.

First, there exists an “*information loss*” problem for models with the eager learning approach. The eager learning approach typically tries to train a global model from the historical time series data. To avoid the intractable training process, some low-rank approximation methods are usually adopted which can find approximate representation of the whole dataset. The consequence is that the constructed models are more influenced by the global distribution of the whole time series data and local behaviour is not captured.

Second, the eager learning approach for time series prediction may suffer from the problem of “*concept drift*”. In the case of sensors monitoring dynamic and evolving environments, the constructed statistical model might be outdated by the time when sufficient historical data is collected to build a global model. Furthermore, the underlying model that generates the data might gradually change over time. As such, paying high computational cost to construct a large, global model that fits the whole of the sensor time series may be a wasteful solution.

Third, the lazy learning approaches on the other hand are too simplistic and suffer from a lack of accuracy and statistical guarantee. For example, the k NN regression cannot estimate the predictive uncertainty of the prediction result, i.e. it cannot give a meaningful probability confidence interval. This predictive uncertainty is necessary in decision making. Finding k NNs continuously on a large number of sensor time series can also be challenging in term of efficiency.

In this chapter, we present **SMiLer**, a *SeMi-Lazy time series prediction system for sensors*. The core of SMiLer is our proposed semi-lazy learning approach to time series prediction, which is illustrated in Figure 4.1. We use the time series of a sensor in the last few time steps as the input request, which is used to retrieve a set of k -Nearest Neighbor (k NN) time series segments from historical data. The k NN

results are then used to construct a Gaussian Processes (GPs) model for predicting the future value (with mean and variance) of the sensor. In general, the proposed semi-lazy learning approach essentially follows the lazy learning paradigm until the last step, where more sophisticated machine learning models (e.g. GPs) are applied on the k NN results of the submitted prediction request.

Note that SMiLer makes prediction for individual sensors based on their own historical time series data. Hence, the individual sensors can be independent and heterogeneous in the SMiLer system. Making prediction using the correlation among the multiple homogeneous sensors is beyond the scope of this study.

Moreover, in Section 4.6, we will expound that why we devise a new method for time series prediction instead of using the trajectory prediction method. Briefly, different application purposes and data properties determine that we should use different models for time series prediction and trajectory prediction.

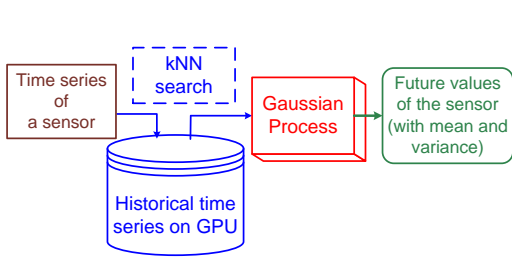


Figure 4.1: An illustration for the semi-lazy learning approach to time series prediction

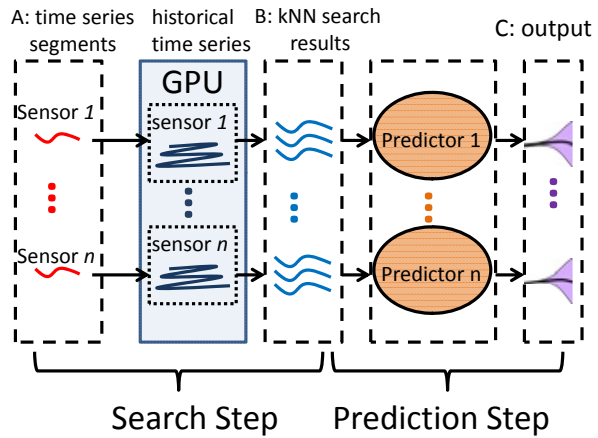


Figure 4.2: Overview framework of SMiLer

By adopting the semi-lazy learning approach, SMiLer has several appealing advantages over existing methods. First, since historical data is kept until prediction time, a very rich set of models are preserved as part of the data. When the query and its k NNs are given, more sophisticated machine learning method is then applied to construct a model. The constructed model caters to specifically making prediction for the submitted query without the need to cater for all parts of the

historical data which may result in over generalized model. Second, the concept drifting problem can be effortlessly eliminated since we only need to insert new time series data into the historical dataset to capture the changes of the environment. Third, the semi-lazy learning approach empowers the traditional lazy learning approach with higher prediction accuracy and advanced functions such as the estimation of predictive uncertainty (i.e. confidence interval of prediction results).

Two main technical obstacles must be overcome to make semi-lazy time series prediction possible: 1) a fast k NN search method for time series and 2) a proper time series prediction model. Figure 4.2 shows the overall framework of SMiLer comprising of two main steps:

- In the *Search Step*, we use recent observations of the sensor (in the last several time steps) as a query to retrieve a small set of k NN reference time series segments using the popular Dynamic Time Warping (DTW). We devise a novel index on the GPU and propose a new enhanced DTW lower bound which accelerates our search process by one order of magnitude over a straightforward GPU k NN search implementation.
- In the *Prediction Step*, we contrive a semi-lazy time series prediction method which utilizes ensemble prediction to make self-correcting, continuous prediction for sensor values with minimal requirement for parameter tuning by human users. The Gaussian Process model is embedded into the framework for time series prediction. By virtue of these methods, SMiLer can achieve higher prediction accuracy than the state-of-the-art competitors by 2-5 folds, and with better estimation of predictive uncertainty.

The rest of this chapter is organized as follows. Next, we discuss an overview of the framework of SMiLer in Section 4.2. Then we describe the design and the implementation of the two main components of SMiLer: the Search Step in Section 4.3 and the Prediction Step in Section 4.4. Finally, we evaluate our system

in Section 4.5, and conclude the chapter in Section 4.7.

4.2 Overview

In this section, we present a formal description of the semi-lazy learning time series prediction model. Table 4.1 lists the basic notations used throughout this chapter.

Table 4.1: Table of Basic Notations of Chapter 4

C^i	time series of sensor i	c_t^i	value of C^i at t
$x_{j,d}^i$	d -length segment of C^i	$X_{k,d}^i$	a set of $x_{j,d}^i$
$y_{j,h}^i$	h -step ahead value of $x_{j,d}^i$	Y_h^i	a set of $y_{j,h}^i$
EKV	Ensemble k NN Vector	k	number of k NNs
ELV	Ensemble Length Vector	d	length of segment

4.2.1 Preliminaries

A time series C^i is a collection of observations made sequentially in time from a sensor i (or more generally, an unknown system), i.e., $C^i = \{c_{t_1}^i, c_{t_2}^i, \dots, c_{t_j}^i, \dots\}$, where $c_{t_j}^i$ is the value of C^i at timestamp t_j . We assume the sample interval Δt of one sensor is always fixed¹, therefore, a time series is only a sequence of data points. $|C^i|$ denotes the length of C^i . A set of contiguous observations of C^i between two points c_t^i and $c_{t+d\Delta t}^i$ is called a *segment* and is denoted by $C_{t,d}^i$. We also call a segment with length d as a d -length segment.

At time t_0 , the h -step ahead prediction is to predict the value of the sensor at time $t_0 + h\Delta t$. Taking a d -length segment $x_{0,d}^i = C_{t_0-(d-1)\Delta t, d}^i = \{c_{t_0-(d-1)\Delta t}^i, \dots, c_{t_0}^i\}$ as model input and denoting the h -step ahead value of $x_{0,d}^i$ by $y_{0,h}^i = c_{t_0+h\Delta t}^i$, the h -step ahead prediction model is a mapping $f(\cdot)$ between $x_{0,d}^i$ and $y_{0,h}^i$, i.e. $y_{0,h}^i = f(x_{0,d}^i)$.

Since the time interval Δt is fixed, hereafter, we simply use $t + d$ (or $t + h$) to replace $t + d\Delta t$ (or $t + h\Delta t$). In this case, d (or h) is basically the number of fixed

¹This is not a real limitation since the user can easily re-interpolate data if the sample rate is changed.

time intervals (i.e. Δt) beyond t . Using $t + d$ can make notations simple and clear in the context to avoid confusing.

4.2.2 Semi-lazy time series prediction

In this section, we present our semi-lazy time series prediction model. Given a test input segment $x_{0,d}^i$, we first introduce an abstract semi-lazy predictor which can predict the probabilistic distribution of $y_{0,h}^i$. Then we propose an ensemble prediction method to exploit a group of predictors. Note that in SMiLer, we build independent semi-lazy time series prediction model for each sensor in a parallel manner giving linear scale-up.

4.2.2.1 Abstract semi-lazy predictor

Given a time series segment $x_{0,d}^i = \{c_{t_0-(d-1)}^i, \dots, c_{t_0}^i\}$ ending at time t_0 , we can retrieve k nearest neighbor segments with length d from time series C^i , of which the query result is $X_{k,d}^i = \{x_{j,d}^i\}_{j=1}^k = [x_{j,d}^i, \dots, x_{k,d}^i]$ (where $x_{j,d}^i$ is a segment of C^i ending at time t_j , i.e. $x_{j,d}^i = \{c_{t_j-(d-1)}^i, \dots, c_{t_j}^i\}$). For each segment $x_{j,d}^i$, its h -step ahead value is $y_{j,h}^i = c_{t_j+h}^i$. We denote the h -step ahead values of every $x_{j,d}^i$ in $X_{k,d}^i$ as a vector $Y_h^i = [y_{1,h}^i, y_{2,h}^i, \dots, y_{k,h}^i]^\top = [c_{t_1+h}^i, c_{t_2+h}^i, \dots, c_{t_k+h}^i]^\top$. Now, given $(X_{k,d}^i, Y_h^i)$, we formally define the abstract semi-lazy time series predictor.

Definition 4.2.1 (Semi-Lazy Time Series Predictor). *Given a d -length time series segment $x_{0,d}^i = \{c_{t_0-(d-1)}^i, \dots, c_{t_0}^i\}$ of C^i ending at time t_0 , the semi-lazy time series predictor is a model which can use the k NN data $(X_{k,d}^i, Y_h^i) = \{x_{j,d}^i, y_{j,h}^i\}_{j=1}^k$ and test input $x_{0,d}^i$ to obtain the posterior distribution of the h -step ahead observation $y_{0,h}^i$ (i.e. $c_{t_0+h}^i$):*

$$y_{0,h}^i = f(x_{0,d}^i, X_{k,d}^i, Y_h^i) \sim \mathcal{N}(\tilde{u}, \tilde{\sigma}^2) \quad (4.1)$$

where $f(\cdot)$ is an abstract predictor which can be instantiated with suitable probabilistic prediction model.

The semi-lazy time series prediction model is built independently for each sensor, but multiple sensors can be processed in the same way. Hereafter, unless otherwise stated, we focus on the k NN search and prediction for one sensor.

The superscript “ i ” (e.g. $y_{0,h}^i$ and $x_{j,d}^i$) indicates that the variables are from sensor i . Hereafter, when we focus on one sensor, we omit the superscript i for convenience.

4.2.2.2 Ensemble prediction

We further introduce a strategy to improve the prediction accuracy as well as to eliminate the parameters of the semi-lazy prediction model. In Definition 4.2.1, for each abstract predictor, there are two parameters: (1) k : the number of nearest neighbors and (2) d : the length of time series segment. For different sensors, semi-lazy prediction models may desire different settings for k and d . To avoid the trouble to specify the parameters as well as to improve the prediction accuracy, we propose an ensemble prediction method, which is a matrix of abstract predictors $f_{i,j}$ for a sensor with different k and d . The final predicted mean and variance are the mixture of all the $f_{i,j}$ predictors. To facilitate explanation, we define an ensemble matrix λ as below:

$$\lambda = \begin{bmatrix} (k_0, d_0) & \dots & (k_0, d_{n-1}) \\ \dots & (k_i, d_j) & \dots \\ (k_{m-1}, d_0) & \dots & (k_{m-1}, d_{n-1}) \end{bmatrix} \quad (4.2)$$

where k_i is the number of nearest neighbor and d_j is the length of time series segment for predictor $f_{i,j}$. We group the different numbers of nearest neighbors in the Ensemble k NN Vector denoted by $EKV = [k_0, \dots, k_{m-1}]$, and group the different lengths of query segments in the Ensemble Length Vector denoted by $ELV = [d_0, \dots, d_{n-1}]$.

In the ensemble matrix λ , each element $\lambda_{i,j}$ also indicates the weight of $f_{i,j}$ contributed to the final prediction result. Hence, the ensemble prediction model

for one sensor is formally defined as follows:

$$f_{em} = \frac{1}{\mathcal{C}_\lambda} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \lambda_{i,j} f_{i,j} \quad (4.3)$$

where \mathcal{C}_λ is the normalization constant by summing the weight of every element in λ , i.e. $\mathcal{C}_\lambda = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \lambda_{i,j}$.

If there is no prior information, we can set all the elements in λ with the same weight. We also propose a more intelligent and self-adaptive method to determine the ensemble matrix if we employ continuous prediction (see Section 4.4.1).

4.2.3 Objective of SMiLer

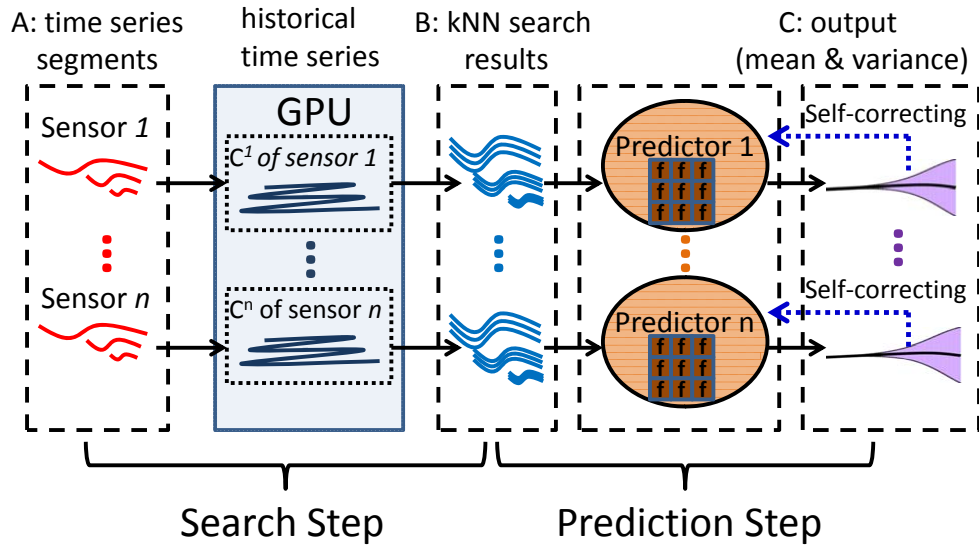


Figure 4.3: Overview framework of SMiLer, which has a Search Step (input A and output B) and a Prediction Step (input B and output C).

SMiLer is designed to make the semi-lazy time series prediction model feasible. To sum up Section 4.2.2, there are two requirements for the semi-lazy time series prediction: (I) k NN search of time series segment with different k and d (see Equation (4.2)); and (II) proper model selection for predictor $f(\cdot)$ with the ensemble prediction. Figure 4.3 shows an overview framework of SMiLer to satisfy

the above objectives.

In the first step, called Search Step, we use time series segment of the target sensor in the last few time steps (Rectangle A in Figure 4.3) as the queries, which are submitted to a Graphics Processing Unit (GPU). Following closely, a set of k NN reference time series segments (Rectangle B in Figure 4.3) are retrieved from the historical dataset of the target sensor. For each sensor, we invoke multiple queries with multiple k and d . The queries of one sensor are parallel processed on the data of itself. Furthermore, we also propose a novel index and an enhanced lower bound of DTW to accelerate the k NN time series search. More details about this part will be discussed in Section 4.3.

Next, in the second step, namely the Prediction Step, the k NN results (Rectangle B in Figure 4.3) are input into the semi-lazy time series prediction models to predict future value (with mean and variance) of each sensor (Rectangle C in Figure 4.3). The ensemble prediction and continuous self-correcting prediction are both investigated to improve the prediction performance as well as to minimize users' assistance for setting parameters. More details about this part will be discussed in Section 4.4.

4.3 DTW k NN search with the GPU

We use DTW distance to find k NN segments from historical time series with the help of the GPU. As we discussed in Section 2.1 of Chapter 2, there have been more than ten similarity measures for spatio-temporal data, such as Euclidean distance [53], DTW [13], LCSS [116], ERP [27], EDR [28] and SpADe [29]. Euclidean distance is simple but sensitive to shifting and scaling problem which usually appears in time series data. Among these measures, DTW is a simple but effective one which is robust to the shifting and scaling problem. Other distance measures can also handle time series similarity search, but they usually need a sophisticated index structure, which cannot be easily implemented on the GPU. Besides, there

are some evidences showing that DTW is the best measures for time series data mining problems [49; 102; 94]. Since DTW requires high computation cost, it is useful to resort to the help of the GPU to accelerate the computation.

4.3.1 Problem formulation

The SMiLer Index is designed to solve the Multiple k NN Search problem derived from the ensemble and continuous prediction (see Section 4.2.2.2), where there are multiple time series segment queries with different d and k for a target sensor. The k NN search with different k in $EKV = [k_0, k_2, \dots, k_{m-1}]$ (see Equation (4.2)) can be solved by invoking the Nearest Neighbor search with maximum value k_n , and then selecting different subsets of the result according to order of the DTW distances.

But it is not trivial for k NN search with different query lengths. For a target sensor, we can invoke multiple k NN queries with query segment length d indicated in $ELV = [d_0, \dots, d_{n-1}]$ (see Equation (4.2)). $[x_{0,d_0}, \dots, x_{0,d_{n-1}}]$ denotes all the queries at time t_0 where $x_{0,d_i} = \{c_{t_0-(d_i-1)}, c_{t_0-(d_i-2)}, \dots, c_{t_0}\}$. We can see that, if $d_i < d_j$, x_{0,d_i} is a suffix of x_{0,d_j} since both of them end at time t_0 . An example of x_{0,d_0} and x_{0,d_1} is shown in Figure 4.5. Based on this suffix property, some computation cost can be reused during the k NN search.

We first define some simplified notations. For a target sensor, ‘‘Master Query’’ MQ denotes the longest query segment where $MQ = x_{d_{n-1}} = \{q_0, \dots, q_{d_{n-1}}\} = \{c_{t_0-(d_{n-1}-1)}, \dots, c_{t_0}\}$. Since every x_{d_i} is a suffix of $x_{0,d_{n-1}}$, we also denote each query segment as ‘‘Item Query’’ IQ where $IQ_i = x_{0,d_i} = \{q_{d_n-d_i}, \dots, q_{d_{n-1}}\} = \{c_{t_0-(d_i-1)}, \dots, c_{t_0}\}$. Figure 4.5(b) shows an example of MQ and its item queries IQ_0 and IQ_1 . The Multiple k NN Search problem can be formally defined as:

Definition 4.3.1 (Multiple k NN Search). *Given a master query MQ of a sensor with the Ensemble Length Vector $ELV = [d_0, \dots, d_{n-1}]$, we can generate a set of item queries $\{IQ_0, \dots, IQ_{n-1}\}$ where $|IQ_i| = d_i$. The objective of the Multiple k NN*

Search is to find k -nearest neighbor segments C_{t,d_i} for each item query IQ_i under DTW distance on time series C of the sensor.

In the following sections, we will exploit the SMiLer Index for the Multiple k NN Search on the GPU. Readers may refer to Section 2.1.2 of Chapter 2 for a review about DTW. Note that in this chapter we only consider the DTW under Sakoe-Chiba band constraint with warping width ρ .

4.3.2 Enhanced lower bound for DTW

We propose an enhanced lower bound for DTW, denoted by LB_{en} , which is derived from the existing lower bound LB_{keogh} [72]. We first define the “envelope” of time series.

Definition 4.3.2 (time series envelope). *Given a time series C and a warping width ρ , the envelope $E(C)$ contains two sequences: upper envelope $U(C)$ and lower envelope $L(C)$, whose i -th elements are defined as:*

$$U_i^c = \max_{-\rho \leq r \leq \rho} (c_{i+r}), \quad L_i^c = \min_{-\rho \leq r \leq \rho} (c_{i+r}) \quad (4.4)$$

LB_{keogh} is the distance between $E(C)$ and the query Q :

$$LB_{keogh}(E(C), Q) = \sum \begin{cases} dist(U_i^c, q_i) & q_i > U_i^c \\ dist(L_i^c, q_i) & q_i < L_i^c \\ 0 & otherwise \end{cases} \quad (4.5)$$

Depending on which envelope is used, for convenience, we simply denote $LB_{EQ}(Q, C) = LB_{keogh}(E(Q), C)$ and $LB_{EC}(Q, C) = LB_{keogh}(E(C), Q)$. Examples of envelope, $LB_{EQ}(Q, C)$ and $LB_{EC}(Q, C)$ are shown in Figure 4.5.

Based on LB_{keogh} , we propose an enhanced lower bound of DTW, denoted

by LB_{en} , which are defined as:

$$LB_{en}(Q, C) = \max\{LB_{EQ}(Q, C), LB_{EC}(Q, C)\}$$

Theorem 4.3.1. $LB_{en}(Q, C)$ is a lower bound of $DTW(Q, C)$.

Proof. We have $LB_{EC}(Q, C) \leq DTW(Q, C)$ and $LB_{EQ}(Q, C) \leq DTW(Q, C)$, therefore, $LB_{en}(Q, C) \leq DTW(Q, C)$ is true. \square

4.3.3 kNN search on the SMiLer Index

In this section, we present our SMiLer Index with its Multiple k NN Search method. Figure 4.4 shows an overview of the process for the Multiple k NN Search.

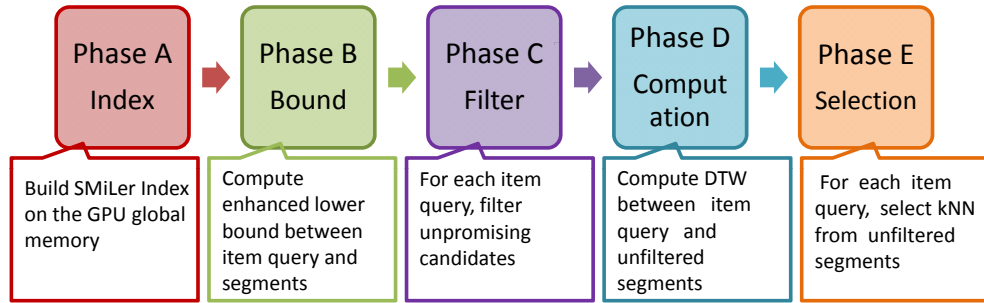


Figure 4.4: An overview of the Multiple k NN search on the SMiLer Index

4.3.3.1 Phase A: build SMiLer Index on the GPU

We build and store the SMiLer Index on the GPU’s global memory, which includes: (I) the disjoint windows and their envelope of time series C , (II) the sliding windows and their envelope of master queries MQ and (III) SD-Table (Sliding-Disjoint window lower bound Table).

Similar to the DualMatch framework [84; 61], we divide the time series C and its envelope $E(C)$ into disjoint windows DW ; and divide the master query MQ and its envelope $E(MQ)$ into sliding windows SW . The lengths of disjoint windows and sliding windows are equal, i.e. $\omega = |DW| = |SW|$. Figure 4.5 illustrates

examples for disjoint windows and sliding windows. Note that we divide the sliding windows in time-reserved order (from right to left in Figure 4.5(b)).

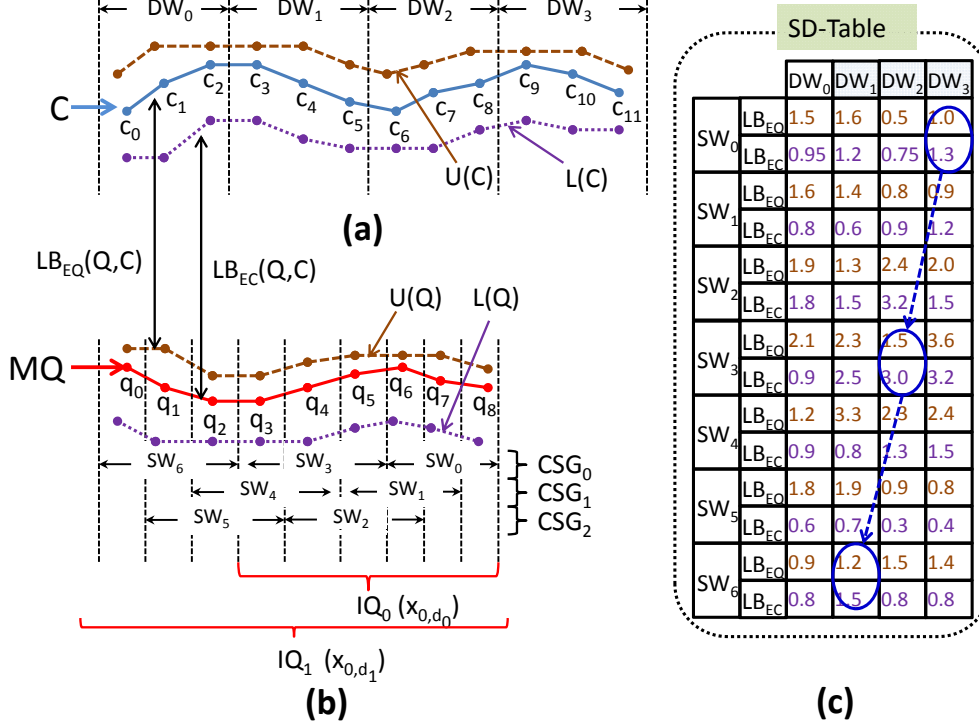


Figure 4.5: An illustration for SMiLer Index

We then construct the SD-Table whose entries are LB_{EQ} and LB_{EC} between all the possible pairs of sliding window and disjoint window. The SD-Table can be quickly computed by GPU with one block processes for each sliding window. An illustration of ST-Table is shown in Figure 4.5(c).

4.3.3.2 Phase B: compute enhanced lower bound

In Phase B, we compute the enhanced lower bound LB_{en} on the SMiLer Index. The idea is that partial sum of the sliding-disjoint window pairs in SD-Table can be the lower bound between an item query and a segment.

We first define the concept of *Catenated Sliding Window Group (CSG)* for a query (IQ_i or MQ) which is inspired by the concept of “equivalence class” in [61].

Definition 4.3.3 (Catenated Sliding Window Group). *A Catenated Sliding Window*

Group (CSG) of query Q contains maximum number of sliding windows without overlap.

$CSG_{i,s}$ denotes a Catenated Sliding Window Group (CSG) of item query IQ_i , where the subscript i is the identifier of IQ_i and the subscript s is the identifier of the first sliding window in the group. For a master query, we denote it as CSG_s . Note that $CSG_{s,i}$ of item query IQ_i is always the prefix of CSG_s of master query MQ .

Example 4.3.1. In Figure 4.5(b), master query MQ has three CSGs which are $CSG_0 = \{SW_0, SW_3, SW_6\}$, $CSG_1 = \{SW_1, SW_4\}$ and $CSG_2 = \{SW_2, SW_5\}$. The CSGs of IQ_0 are $CSG_{0,0} = \{SW_0, SW_3\}$, $CSG_{0,1} = \{SW_1\}$ and $CSG_{0,2} = \{SW_2\}$. $CSG_{0,s}$ of IQ_0 is always a prefix of CSG_s of MQ .

An important point to note is that the alignment between sliding windows in $CSG_{i,s}$ and disjoint windows of C defines an alignment between IQ_i and a segment C_{t,d_i} , where t is determined by the following lemma:

Lemma 4.3.1. Suppose that item query IQ_i has a $CSG_{i,s}$ whose sliding windows are aligned with a set of contiguous disjoint windows (from right to left) i.e. $\{DW_r, DW_{r-1}, \dots\}$. Then this alignment defines an alignment between IQ_i and a segment C_{t,d_i} where the starting position of C_{t,d_i} in C is:

$$t = (r - |CSG_{i,s}| + 1) * \omega - (d_i - s) \% \omega \quad (4.6)$$

where $|CSG_{i,s}|$ is the number of windows in $CSG_{i,s}$, ω is the length of window and d_i is the length of IQ_i .

Proof. We have two observations: (O1) For the item query IQ_i with $CSG_{i,s}$, the number of points in the right side of SW_s (exclusive) is s . Then, the number of points in the left side of SW_s (inclusive) is $d_i - s$. According to the definition of CSG, in the left side of SW_s , only $(d_i - s) \% \omega$ points are not included in $CSG_{i,s}$.

(O2) If the rightmost aligned disjoint window is DW_r , then the leftmost aligned disjoint window is $DW_{r-|CSG_{i,s}|+1}$. The starting position of the point of $DW_{r-|CSG_{i,s}|+1}$ is $(r - |CSG_{i,s}| + 1) * \omega$.

Then, based on (O1) and (O2), in order to match IQ_i with a segment, we only need $(d_i - s) \% \omega$ number of points from the starting position of $DW_{r-|CSG_{i,s}|+1}$ to left forward. Therefore, the starting position of the aligned segment C_{t,d_i} is $t = (r - |CSG_{i,s}| + 1) * \omega - (d_i - s) \% \omega$. \square

Corollary 4.3.1. *For each pair of IQ_i and C_{t,d_i} , there is one and only one alignment between $CSG_{i,s}$ and corresponding disjoint windows.*

Proof. Suppose that the disjoint windows $\{DW_r, DW_{r-1}, \dots, DW_{r-m+1}\}$ are covered by segment C_{t,d_i} where m is the number of the disjoint windows and r is the identifier of the rightmost disjoint window. We have $m = \lfloor \frac{(r+1)*\omega - t}{\omega} \rfloor$. We prove the corollary in two directions.

(I) There is one alignment. For the segment C_{t,d_i} , we can find a $CSG_{i,s}$ of IQ_i , where $s = t + d_i - (r + 1) * \omega$ (s is the number of points in the right side of SW_s). The number of sliding windows in $CSG_{i,s}$ is $\lfloor \frac{d_i - s}{\omega} \rfloor$. By replacing s with $s = t + d_i - (r + 1) * \omega$, it is computed that $m = \lfloor \frac{d_i - s}{\omega} \rfloor = \lfloor \frac{(r+1)*\omega - t}{\omega} \rfloor$. Therefore, $CSG_{i,s}$ can be aligned with the disjoint windows covered by C_{t,d_i} .

(II) There is only one alignment. We prove it by contradiction. Suppose that for a segment C_{t,d_i} there are at least two CSGs, denoted by $CSG_{i,s'}$ and $CSG_{i,s''}$, which are aligned with the disjoint windows $\{DW_r, DW_{r-1}, \dots, DW_{r-m+1}\}$. The number of points of C_{t,d_i} in the left side of DW_{r-m+1} is $s_l = (r - m + 1) * \omega - t$ (s_l is the number of points from t to the starting point of the disjoint window DW_{r-m+1}). Then the total length of C_{t,d_i} is $d'_i = s_l + m * \omega + s'$ and $d''_i = s_l + m * \omega + s''$. Since $s' \neq s''$, it is obvious that $d'_i \neq d''_i$. But C_{t,d_i} have only one length, i.e. $d_i = d'_i = d''_i$. There is a contradiction. Therefore, there is only one alignment between the CSG and the disjoint windows.

Based on (I) and (II), we prove the claim. \square

Then we can deduce a lower bound of DTW, called window enhanced lower bound LB_w , between IQ_i and C_{t,d_i} from the SD-Table.

Definition 4.3.4. *Given a $CSG_{i,s} = \{SW_s, SW_{s+w}, \dots\}$ of IQ_i and a disjoint window DW_r , we can compute the window enhanced lower bound LB_w between IQ_i and C_{t,d_i} as:*

$$LB_w(IQ_i, C_{t,d_i}) = \max \begin{cases} \sum_{j=0}^{m-1} LB_{EQ}(SW_{s+j*\omega}, DW_{r-j}) \\ \sum_{j=0}^{m-1} LB_{EC}(SW_{s+j*\omega}, DW_{r-j}) \end{cases} \quad (4.7)$$

where $m = |CSG_{i,s}|$ and $t = (r - m + 1) * \omega - (d_i - s) \% \omega$.

Finally, we have the following important theorem:

Theorem 4.3.2. *The following inequality always holds:*

$$LB_w(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$$

Proof. Suppose that the disjoint windows $\{DW_r, DW_{r-1}, \dots, DW_{r-m+1}\}$ are covered by segment C_{t,d_i} where r is the identifier of the rightmost disjoint window and m is the number of disjoint windows such that $m = \lfloor \frac{(r+1)*\omega - t}{\omega} \rfloor$. Then, the number of points of C_{t,d_i} in the left side of DW_{r-m+1} is $s_l = (r - m + 1) * \omega - t$ and the number of points of C_{t,d_i} in the right side of DW_r is $s_r = t + d_i - (r + 1) * \omega$ (refer to Proof of Corollary 4.4).

For simplicity, let the distance between the point q_j and the envelop of c_i be $LB(E(c_i), q_j)$, i.e.

$$LB(E(c_i), q_j) = \begin{cases} dist(U_i^c, q_j) & q_j > U_i^c \\ dist(L_i^c, q_j) & q_j < L_i^c \\ 0 & otherwise \end{cases}$$

For the lower bound $LB_{EC}(IQ_i, C_{t,d_i})$, we have:

$$\begin{aligned}
LB_{EC}(IQ_i, C_{t,d_i}) &= \sum_{j=0}^{d_i-1} LB(E(c_{t+j}), q_j) \\
&= \sum_{j=0}^{s_l-1} LB(E(c_{t+j}), q_j) + \sum_{j=d_i-s_r+1}^{d_i} LB(E(c_{t+j}), q_j) \\
&\quad + \sum_{j=s_l}^{d_i-s_l-s_r} LB(E(c_{t+j}), q_j) \\
&\geq \sum_{j=s_l}^{d_i-s_l-s_r} LB(E(c_{t+j}), q_j) = \sum_{a=0}^{m-1} LB_{EC}(SW_{s+a*\omega}, DW_{r-a})
\end{aligned}$$

In the same way, we can also get $LB_{EQ}(IQ_i, C_{t,d_i}) \geq \sum_{a=0}^{m-1} LB_{EQ}(SW_{s+a*\omega}, DW_{r-a})$. Combining these two inequalities, we have $LB_w(IQ_i, C_{t,d_i}) \leq LB_{en}(IQ_i, C_{t,d_i})$. According to Theorem 4.3.1, for the enhanced lower bound we also have $LB_{en}(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$. Finally, we get $LB_w(IQ_i, C_{t,d_i}) \leq LB_{en}(IQ_i, C_{t,d_i}) \leq DTW(IQ_i, C_{t,d_i})$

□

Based on the above lemmas and theorem, we can compute the lower bound between IQ_i and segments efficiently using GPU. The insight of Equation (4.7) is that the sum of the rows of SD-Table belonging to the same CSG with their aligned disjoint windows is the lower bound of IQ_i (see Example 4.3.2). Since the CSGs of item queries are always the prefix of that of master query, by summing the rows of SD-Table from top to down, we can obtain the lower bound for every item query during the process.

This computation can be parallel and efficiently processed by the GPU. The idea is to use one block of GPU to process one CSG (rows in SD-Table) and use each thread of the block to handle several disjoint windows (columns in SD-Table). In this way, the parallel processing capability of the GPU can be fully utilized.

Example 4.3.2. *In Figure 4.5, we show the SD-Table of master query MQ and time series C. For the GPU, we use Block 0 to handle rows of SD-Table in order*

of $SW_0 \rightarrow SW_3 \rightarrow SW_6$, and use Block 1 to do it in order of $SW_1 \rightarrow SW_4$, and use Block 2 to do it in order of $SW_2 \rightarrow SW_5$.

In Block 0, there is a thread (blue dashed arrow) visiting aligned windows of SD-Table in order of: $(SW_0, DW_3) \rightarrow (SW_3, DW_2) \rightarrow (SW_6, DW_1)$. To sum the first two elements, we have the lower bound between IQ_0 and $C_{6,6}$, e.g. $LB_{EC}(IQ_0, C_{6,6}) = LB_{EC}(SW_0, DW_3) + LB_{EC}(SW_3, DW_2)$. With the sum of all the three elements, we have the lower bound between IQ_1 and $C_{3,9}$, e.g. $LB_{EC}(IQ_1, C_{3,9}) = LB_{EC}(IQ_0, C_{6,6}) + LB_{EC}(SW_6, DW_1)$. Thus, by scanning the SD-Table from top to down, we get the lower bound of IQ_0 and IQ_1 between some segments.

Algorithm 4.1 shows the pseudo code for computing the lower bound from SD-Table of SMiLer Index. We use one thread to scan the columns of disjoint windows to compute the lower bound of item queries. There are two key points. The first one is to do shift sum (recall Equation (4.7)) to compute the lower bound LB_{EQ} and LB_{EC} in Line 7 and Line 8. The second one is to compute LB_w (recall Definition 4.3.4) for each item query in Line 11.

4.3.3.3 Phase C: filtering unpromising candidates

In Phase C, we filter the unpromising candidates. The method is to scan all pairs of IQ_i and C_{t,d_i} to discard candidates whose lower bound is larger than threshold τ_i .

There are two methods to determine threshold τ_i for item query IQ_i . The first one is to select the segment with the k -th smallest lower bound, and then set τ_i as the DTW distance between the segment and IQ_i . The second method is to re-use the k NN results during continuous prediction. Suppose that at time $t_0 - 1$ there is a query item IQ'_i . For continuous prediction, at time t_0 , the new query item IQ_i is formed by adding one point to the head of IQ'_i and removing the last point of IQ'_i . Since the IQ_i and IQ'_i are changing gradually, we can take the distance between the k -th NN segment of IQ'_i and the item query IQ_i as threshold τ_i , which is tighter than that of the first method. In SMiLer, we use the first method

Algorithm 4.1: Phase B: Compute enhanced lower bound

```
// One block of the GPU takes one CSW
1 for each  $CSG_s$  of master query  $MQ$  do
    // One thread of the block takes one disjoint window
2   for each disjoint window  $DW_r$  of  $C$  do
3      $wc \leftarrow 0$  // count window number
4      $qc \leftarrow 0$  // count item query number
5      $qd \leftarrow s + \omega$ 
    //  $n$  is number of item queries per  $MQ$ 
6     while  $qc < n$  do
7        $LB_q \leftarrow LB_{EQ}(SW_{s-wc*\omega}, DW_{r-wc})$ 
8        $LB_c \leftarrow LB_{EC}(SW_{s-wc*\omega}, DW_{r-wc})$ 
9       if  $qd + \omega > |IQ_{qc}|$  and  $qd \leq |IQ_{qc}|$  then
10         $t \leftarrow (r - wc) * \omega - (qd - s) \% \omega$ 
11         $LB_w(IQ_{qc}, C_{t,qd}) \leftarrow \max\{LB_q, LB_c\}$ 
12         $qc \leftarrow qc + 1$ 
13       $wc \leftarrow wc + 1$ 
14       $qd \leftarrow qd + \omega$ 
```

to determine the τ_i in initial queries, and then use the second method for the following continuous queries.

4.3.3.4 Phase D: compute real DTW distance

After filtering all unpromising candidates, we compute the real DTW distance between the un-filtered candidates and the item queries. In Algorithm 4.2, we show the pseudo code to compute the DTW distance (with Sakoe-Chiba band constraint) between Q and C .

The novelty of our computation method (i.e. Algorithm 4.2) lies in the use of a compressed warping matrix. The shared memory, which is much faster than the global memory, is an ideal place for the warping matrix of DTW which is frequently accessed. However, the shared memory is quite small (up to 64KB). Without careful design, the GPU may have to store part of the warping matrixes in global memory instead. We design a compressed warping matrix with size

$2 \times (2 * \rho + 2)$ where ρ is the warping width. The essential idea is to temporarily store the matrix elements along the warp path (2 rows and $2 * (\rho + 1)$ columns), while the modulus operation (%) is employed to reuse the memory space.

Algorithm 4.2: GPU_fast_DTW

input : a query Q ; a time series C
output: The DTW distance between Q and C
// This is pseudo-code for one thread.

```

1  $m = 2 * \rho + 2$ ; //  $\rho$  is warping width
2  $\gamma[m][2]$  //  $\gamma$  is allocated in shared memory
3 for  $i \leftarrow 1$  to  $m - 1$  do
4    $\gamma(i, 0) = \infty$ 
5    $\gamma(0, 1) = \infty$ 
6   for  $j \leftarrow 1$  to  $d$  do
7      $\gamma((j - \rho - 1)\%m, j\%2) = \infty$ 
8      $\gamma((j + \rho)\%m, (j - 1)\%2) = \infty$ 
9     for  $i \leftarrow (j - r)$  to  $j + r$  do
10       $\gamma(i\%m, j\%2) = dist(q_i, c_j) + \min \begin{cases} \gamma((i - 1)\%m, j\%2) \\ \gamma(i\%m, (j - 1)\%2) \\ \gamma((i - 1)\%m, (j - 1)\%2) \end{cases}$ 
11 return  $\gamma(d\%m, d\%m)$  //  $|Q| = |C| = d$ 

```

In Algorithm 4.2, we show the pseudo code to compute the DTW distance (with Sakoe-Chiba band constraint) between a query Q and a time series C . There are several technical issues that should be noticed. First, in order to reduce the number of accesses to the global memory, the query Q should reside in the shared memory; and furthermore, the query Q must be placed in the inner loop of Algorithm 4.2 (i.e. Q should be placed in line 9 instead of line 6). The second trick for reducing memory access latency is “coalesced access” [102; 37]. When several consecutive threads (i.e. with consecutive thread identifiers) access successive sliding windows for retrieving time series segments, all these accesses may be combined into one read request to the memory, resulting in improving the efficiency significantly.

4.3.3.5 Phase E: selection

In this phase, we use a fast k -selection algorithm on the GPU to select k candidates with the smallest DTW distance from all the unfiltered candidates. The main technique is distributive partitioning for k -selection on the GPU [2]. First, after defining a set of buckets, for each item query IQ_i , we partition all candidates, whose distance is smaller than τ_i , into different buckets according to their DTW distance. Then we identify the bucket that contains the k th-smallest candidate. Next, we focus only on the entries in this bucket, followed by projecting the entries in the bucket into a new set of buckets again. The iteration is repeated until we find the k th-smallest time series segment.

We make two improvements from the existing work [2]: (1) our implementation accepts multiple k -selections, with one block handling one k -selection for one query; (2) we return all the k smallest segments instead of only k -th one.

4.3.3.6 Reuse for continuous query

During continuous prediction, we can avoid building the SMiLer Index from scratch at every step. Suppose that at time $t_0 - 1$ there is a master query MQ' . Then at time t_0 , the new master query MQ is constructed by adding one point to the head of MQ' and removing the last point of MQ' . Consequently, we only need to add a new sliding window to MQ and remove the last sliding window of MQ' to avoid re-constructing the SMiLer Index.

Figure 4.6 illustrates how to update the SD-Table during the continuous prediction. For a new master query at time t_0 , we first clear the space of the last sliding windows SW_n of SD-Table, and then the new sliding window SW' is placed in the memory space of SW_n (see Figure 4.6(b)). The starting cursor (the red vertical arrow) of the SD-Table is also moved from SW_0 to SW' . Then at time $t_0 + 1$ (see Figure 4.6 (c)), the new sliding window SW'' replaces the memory space of SW_{n-1} and the starting cursor is moved to SW'' . In addition, after adding a

new point, the envelopes of previous ρ sliding windows are changed. Therefore, we need to re-calculate LB_{EQ} of these affected sliding windows. For example, if $\rho = 1$, LB_{EQ} of SW_0 is re-calculated in Figure 4.6 (b), and LB_{EQ} of SW' is re-calculated in Figure 4.6(c).

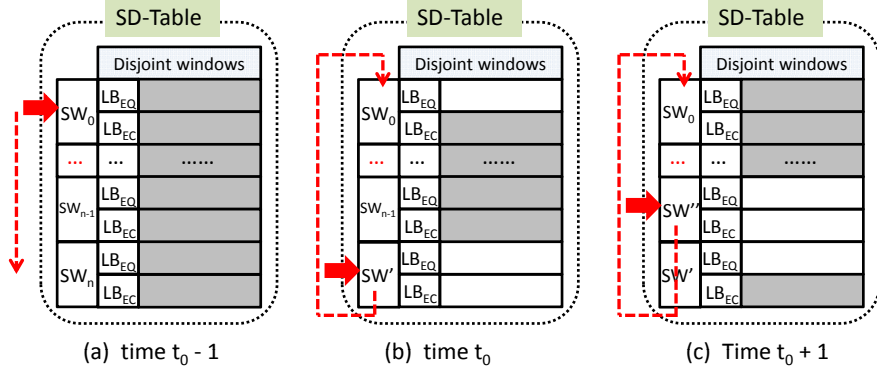


Figure 4.6: Reuse SD-Table of the SMiLer Index

4.3.4 Utility of the GPU

The SMiLer Index and its k NN search method are carefully tailored for the computation on the GPU. We summarize the utility of the GPU from several perspectives. First, the proposed lower bounds (LB_{en} and LB_w) are applicable to the computation of the GPU. Owing to the powerful parallel processing ability, we can obtain a tighter lower bound by computing both LB_{EC} and LB_{EQ} without increasing the response time.

Second, the SMiLer Index is devised to fully utilize the parallel processing ability of the GPU. On one hand, the search on the SMiLer Index is parallelized in as fine-grained manner as possible. For example, in Phase A we use one block to treat one sliding window to construct SD-Table; and in Phase B we use one block to treat one CSG to obtain the lower bound.

On the other hand, we try to ensure that processing in each thread block is as homogenous as possible. We use a two-phase scheme to filter and verify candidates (Phase C and Phase D) instead of having them in one phase. The

reason is that, due to the property of the SIMD architecture of the GPU, the GPU hardware serializes different execution paths. If we mixed Phase C and Phase D, threads doing different processing need to wait for each other before continuing their processing which sacrifices efficiency.

Third, the SMiLer Index can easily scale up with multiple sensors. To enable k NN search for multiple sensors, we only need to create multiple SMiLer Indexes and invoke more blocks.

4.4 Time series prediction via “semi-lazy” learning

Following by Section 4.2.2, we give a detailed discussion about our semi-lazy time series prediction model. We first present the self-correcting continuous prediction. Next, we introduce the instantiation of the *abstract predictor*, including a simple aggregate regression predictor and a sophisticated Gaussian Process predictor.

4.4.1 Self-correcting continuous prediction

The continuous prediction provides opportunities to learn and adjust the ensemble matrix λ to improve the performance of SMiLer. In real-life applications, we usually need to continuously predict the future value of sensors, as well as to monitor the reading value of sensors. The idea for self-correcting is to increase the weight of the predictor $f_{i,j}$ which makes good prediction after comparing the predicted value and the true value (see Section 4.4.1.1). Additionally, We also devise a sleep and recovery mechanism (see Section 4.4.1.2) which can make the predictors with small weight temporarily sleep to reduce computation cost.

4.4.1.1 Self-correcting with the ensemble matrix

During the continuous prediction, we can learn to adjust the weight of each predictor in the ensemble matrix. The trick is that, after acquiring the true value of the sensor, we can evaluate each predictor by comparing the true value with

the predicted one. Then we can increase the weight of predictors making good prediction.

Taking an abstract predictor $f_{i,j}$ as an example, we denote the true value of the sensor at time t as $y(t)$, and denote the predicted mean and variance as $\tilde{u}_{i,j}(t)$ and $\tilde{\sigma}_{i,j}^2(t)$. The likelihood function of $f_{i,j}$ after observing $y(t)$ is:

$$l_{i,j}(t) = l(y(t), \tilde{u}_{i,j}(t), \tilde{\sigma}_{i,j}^2(t)) \quad (4.8)$$

$$= \frac{1}{\sqrt{2\pi\tilde{\sigma}_{i,j}^2(t)}} \exp\left(-\frac{(y(t) - \tilde{u}_{i,j}(t))^2}{2\tilde{\sigma}_{i,j}^2(t)}\right) \quad (4.9)$$

It is clear that the larger the likelihood $l_{i,j}(t)$ is, the better the predictor is. Then the weight of $f_{i,j}$ in the ensemble matrix at time t is adjusted as follows:

$$\bar{\lambda}_{i,j}(t) = \lambda_{i,j}(t-1) + \frac{l_{i,j}(t)}{\sum_i \sum_j l_{i,j}(t)} \quad (4.10)$$

After Equation (4.10), we need to further re-normalized $\bar{\lambda}_{i,j}(t)$ to get the final weight of the predictor $f_{i,j}$, i.e.:

$$\lambda_{i,j}(t) = \frac{\bar{\lambda}_{i,j}(t)}{\sum_i \sum_j \bar{\lambda}_{i,j}(t)} \quad (4.11)$$

In fact, combining Equation (4.10) and Equation (4.11), $\lambda_{i,j}(t)$ is an effectively exponential smoothing average of the posterior probability of the predictor $f_{i,j}$ over time.

4.4.1.2 Sleep and recovery

We devise a mechanism to control the sleep and recovery of every predictor. If $\lambda_{i,j}(t)$ is smaller than a threshold, we can temporarily make $f_{i,j}$ sleep to reduce the computation cost. After several steps, the predictor would be recovered.

The mechanism is briefly presented here. In SMiLer, each predictor $f_{i,j}$ has a sleep counter $\varsigma_{i,j}$ specified how many steps it would sleep. If the weight $\lambda_{i,j}$

is smaller than threshold $\eta = \frac{1}{2 * n * m}$ ($n * m$ is the number of elements of the ensemble matrix), we make predictor $f_{i,j}$ sleep, who will recover when the number of subsequent prediction steps exceeds $\varsigma_{i,j}$. If there are κ predictors recovered, the new weight of every recovered predictor is $\eta / (1 - \kappa * \eta)$. After normalization, the weight of recovered predictors are equal to η .

Aiming to make the “weaker” predictor sleep longer, the sleep counter $\varsigma_{i,j}$ is self-adaptive during the continuous prediction. $\varsigma_{i,j}$ is first initialized as 1, which means the predictor would only sleep for one step. If after recovery the predictor $f_{i,j}$ goes to sleep immediately in next step, we will double the value of $\varsigma_{i,j}$. Otherwise, if the predictor successfully avoids the sleep trap, we would continuously halve the value of $\varsigma_{i,j}$ at every prediction step until $\varsigma_{i,j} = 1$.

4.4.2 Instantiation of the abstract predictor

We will discuss the possible instantiation of the abstract semi-lazy predictor (see Section 4.2.2). We first present a simple aggregation predictor, and then a sophisticated Gaussian Processes predictor.

4.4.2.1 A simple aggregation predictor

One simple predictor is a function which can aggregate all the k NN’s h -step ahead values of k NN data. We define an Aggregation Regression (AR) function with pseudo-mean \tilde{u}_0 and pseudo-variance $\tilde{\sigma}_0^2$:

$$\hat{y}(t_0 + h) = f(x_{0,d}, X_{k,d}, Y_h) \quad (4.12)$$

$$= AR(x_{0,d}, X_{k,d}, Y_h) \sim \mathcal{N}(\tilde{u}, \tilde{\sigma}^2) \quad (4.13)$$

$$\tilde{u} = \frac{\sum_{a=1}^k y_{a,h}}{k} \quad (4.14)$$

$$\tilde{\sigma}^2 = \frac{\sum_{a=1}^k (y_{a,h} - \tilde{u})^2}{k} \quad (4.15)$$

AR predictor is simple and can be effectively computed, but its drawback is that the true value of $y(t_0 + h)$ may not follow the normal distribution defined by \tilde{u}_0 and $\tilde{\sigma}_0^2$.

4.4.2.2 Gaussian Process predictor

In this section, we introduce the Gaussian Process (GP) predictor, which has better prediction accuracy and good ability to estimate the predictive uncertainty. In Section 4.4.2.2.1, we first briefly recall some fundamentals of the GP.

4.4.2.2.1 Review of the Gaussian Process We give a brief review of Gaussian Processes (GPs) here. Please refer to [95] for a comprehensive introduction. A Gaussian Process is a collection of random variables, any subset of which has a joint normal distribution. Suppose that a set of data pairs $(X, Y) = \{x_a, y_a\}_{a=1}^k$ are random variables, where x_a is a d -dimensional vector and y_a is the predicted value. We can assume that there is an underlying prediction function $f(\cdot)$ such that $\hat{y}_a = f(x_a)$ is based on the Gaussian Process, which is fully specified by the mean function $m(x)$ and the covariance function $c(x_a, x_b)$. We usually further assume that the mean function is set to be zero, i.e.:

$$[y_1, y_2, \dots, y_n]^\top \sim \mathcal{GP}(0, \Sigma) \quad (4.16)$$

where $\Sigma_{ab} = \text{cov}(y_a, y_b) = \text{cov}(f(x_a), f(x_b)) = c(x_a, x_b)$, which specifies the covariance between pairs of random variables. A widely-used covariance function is the squared exponential (SE) covariance function, i.e.,

$$\begin{aligned} \text{cov}(y_a, y_b) &= \text{cov}(f(x_a), f(x_b)) = c(x_a, x_b) \\ &= \theta_0^2 \exp\left(-\frac{1}{2} \frac{\|x_a - x_b\|^2}{\theta_1^2}\right) + \delta_{ij} \theta_2^2 \end{aligned}$$

where δ_{ab} is a Kronecker delta. $\delta_{ab} = 1$ if and only if $a = b$ and $\delta_{ab} = 0$ otherwise. The vector $\Theta = \{\theta_0, \theta_1, \theta_2\}$ is a set of hyperparameters. In particular, θ_1 is called

a *characteristic length-scale*, which determines how relevant an input is: if the length-scale has a very large value, the covariance becomes almost independent of that input, effectively removing it from the inference.

Now given data $(X, Y) = \{x_a, y_b\}_{a=1}^n$, and a test input vector x_0 , we want to estimate the predictive distribution of the value y_0 corresponding to the input x_0 . Given the Gaussian Process prior and Bayesian rules, the joint distribution of the observed values and the predicted value y_0 is given by

$$\begin{bmatrix} Y \\ y_0 \end{bmatrix} \sim \mathcal{GP} \left(0, \begin{bmatrix} C(X, X) & C(X, x_0) \\ C(x_0, X) & c(x_0, x_0) \end{bmatrix} \right) \quad (4.17)$$

where $C(X, x_0) = [c(x_1, x_0), \dots, c(x_n, x_0)]^\top$ is the $n \times 1$ vector of variances between the test vector and training vectors (similar for the other entries $C(X, X)$, $C(x_0, x_0)$ and $C(x_0, X)$). For simplicity, we use a compact notation as $C = C(X, X)$, $c_0 = C(X, x_0) = C^\top(x_0, X)$.

In the Gaussian Process model, for a test input x_0 , the predictive distributive is simply obtained through conditioning on the training data. The joint distribution of the variables being Gaussian, the posterior distribution for the input test data $p(\hat{y}_0|x_0, X, Y)$ is also a Gaussian distribution, with the following mean and variance:

$$\hat{y}_0 = f(x_0) \sim \mathcal{N}(u_0, \sigma_0^2) \quad (4.18)$$

$$u_0 = E(y_0) = c_0^\top C^{-1} Y \quad (4.19)$$

$$\sigma_0^2 = cov(y_0) = c(x_0, x_0) - c_0^\top C^{-1} c_0 \quad (4.20)$$

An illustration for GP prediction is shown in Figure 4.7.

4.4.2.2.2 Semi-lazy Gaussian Process Predictor For the GP predictor, given an input test segment $x_{0,d}$, the predictive distribution of $\hat{y}_{0,h}$ is obtained through conditioning on the k NN data $(X_{k,d}, Y_h)$ (recall Section 4.2.2.1). The predictive

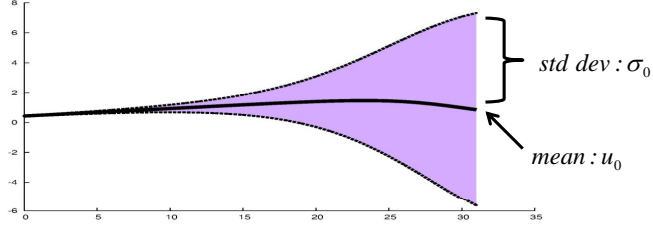


Figure 4.7: An illustration of a Gaussian Process with predicted mean and variance

distribution is also a Gaussian distribution with mean \tilde{u} and variance $\tilde{\sigma}^2$ as follows:

$$\hat{y}(t_0 + h) = f(x_{0,d}, X_{k,d}, Y_h) \quad (4.21)$$

$$= GP(x_{0,d}, X_{k,d}, Y_h) \sim \mathcal{N}(\tilde{u}, \tilde{\sigma}^2) \quad (4.22)$$

$$\tilde{u} = c_0^\top C^{-1} Y_h \quad (4.23)$$

$$\tilde{\sigma}^2 = c(x_{0,d}, x_{0,d}) - c_0^\top C^{-1} c_0 \quad (4.24)$$

where C , c and c_0 are specified by the covariance function:

$$c(x_a, x_b) = \theta_0^2 \exp\left(-\frac{1}{2} \frac{\|x_a - x_b\|^2}{\theta_1^2}\right) + \delta_{ab} \theta_2^2 \quad (4.25)$$

Before making prediction, an important point is that we must determine the hyperparameters $\Theta = \{\theta_0, \theta_1, \theta_2\}$.

4.4.2.2.3 Online training for model optimization We use an online training method to determine the hyperparameters $\Theta = \{\theta_0, \theta_1, \theta_2\}$. For the eager learning approach, a heavy training process is employed to learn the optimal hyperparameters of GP in a pre-processing stage. In contrast, with the semi-lazy learning approach, we can afford the time to invoke an online training process to determine the hyperparameters because there are only a small number of training points (i.e. k NN data $(X_{k,d}, Y_h)$). The advantage of this method is that the hyperparameters are specially trained for the test input $x_{0,d}$ (and its neighbors). In this way, we can avoid the underfitting or overfitting problems of the eager learning approach.

Now we explain how to train the GP predictor (to determine the hyperparameters) on the k NN dataset $(X_{k,d}, Y_h)$. For the GP, the predictive log probability when leaving out a training item $(x_{a,d}, y_a)$ is [95]:

$$\text{logp}(y_a|X_{k,d}, Y_{-a,h}, \Theta) = -\frac{1}{2}\text{log}\sigma_a^2 - \frac{(y_a - u_a)^2}{2\sigma_a^2} - \frac{1}{2}\text{log}2\pi \quad (4.26)$$

where the notation $Y_{-a,h}$ means all the h -step ahead values in Y_h except y_a . The u_a and σ_a^2 are computed according to Equation (4.23) and Equation (4.24) respectively, in which the training set is $(X_{-a,k,d}, Y_{-a,h})$. Thus, the leave-one-out (LOO) log likelihood function on the whole k NN data is:

$$L(X_{k,d}, Y_h, \Theta) = \sum_{a=1}^k \text{logp}(y_a|X_{k,d}, Y_{-a,h}, \Theta) \quad (4.27)$$

The objective is to determine Θ to maximize the LOO log likelihood function (i.e. Equation (4.27)). To achieve this goal, we can compute its partial derivatives w.r.t. the hyperparameters and use the Conjugate Gradient optimization.

It seems that we need to compute Equation (4.26) k times in order to optimize Equation (4.27). However, since the expressions in Equation (4.23) and Equation (4.24) are almost identical for different points (only one column and one row removed in turn), the computation cost can be significantly reduced by the inversion of the partitioned matrix. An efficient approach to such training process can be found in [105].

4.4.2.2.4 Online training in continuous prediction In the continuous prediction, we can use an online optimization method to train the GP model. The intuition for the online training is that the hidden model generating the time series should change gradually. Consequently, the fixed steps pursuit training method is enough to find near-optimal value of the hyperparameters. Based on this point, in SMiLer, we only use the fixed five-step gradient descent to update the hyperparameters for the subsequential predictions.

Formally, the hyperparameters can be updated by the following fixed steps pursuit gradient decent method. For a time series predictor GP at time t , $\theta_r(t)$ denotes a hyperparameter and $L(t)$ denotes the LOO log likelihood function on the k NN data $(X_{k,d}, Y_h)$. Then, a successive estimate of the hyperparameter at time $t + 1$ is computed with the following formula:

$$\theta_r(t + 1) = \theta_r(t) - \frac{\partial L(t + 1)}{\partial \theta_r(t)} \quad (4.28)$$

After deducing an initial approximation of $\theta_r(t + 1)$ from Equation (4.28), we can further employ the Conjugate Gradient (CG) optimization method (with fixed steps of descent) to obtain a near-local optimal value of θ_r . By this method, we can avoid using random seeds to train the model at every step. Moreover, the energy paid for the training process in previous steps is partially preserved.

4.5 Experiments

4.5.1 Settings

We aim to answer these questions in the experiment:

- Search Step: Can SMiLer support fast k NN search? We exhibit that it accelerates the process of k NN search by more than one order of magnitude over its baselines.
- Prediction Step: How is the prediction performance of SMiLer? We show that its prediction accuracy can outperform the state-of-the-art competitors by 2-5 folds with good estimation of predictive uncertainty. The ensemble and self-correcting prediction method does improve the prediction performance of SMiLer.
- Practicality: How well does SMiLer perform in real-life applications? We show that it can carry out prediction for sensors in real time and scale well.

We also claim that SMiLer is more practical than all its competitors.

Table 4.2: Default parameter for experiment of Chapter 4

Parameter	Description	value
ρ	warping width	8
ω	window length	16
ELV	Ensemble Length Vector	{32, 64, 96}
EKV	Ensemble k NN Vector	{8, 16, 32}

4.5.1.1 Environment and parameters

Experiments were conducted on a CPU-GPU platform. The GPU is a GeForce GTX TITAN with 6 GB memory. We implemented the GPU code using CUDA 6. The other program was implemented in C++ and was running on CentOS 6 with an Intel Core i7-3820 CPU server and 64 GB RAM.

Table 4.2 lists the default parameters in the experiment. For the DTW computation (SMiLer and its competitors), we set the warping width as $\rho = 8$, i.e. about 10% of the maximum number of dimensions which is suggested by a previous study [96]. The window size ω is set to 16. Unless otherwise stated, we set $ELV = \{32, 64, 96\}$ which means for each sensor we will do three k NN searches with different query length d . For all queries, we set $k = 32$ which is the maximum value in $EKV = \{8, 16, 32\}$. Unless otherwise stated, in SMiLer we used a 3×3 ensemble matrix for prediction (see Section 4.2.2.2) where k and d are indicated in EKV and ELV respectively. Note that SMiLer is not sensitive to the parameter settings since it can self-adaptively select the best parameters by applying the ensemble and self-correcting prediction method.

4.5.1.2 Datasets

We used three real-life time series datasets to evaluate our system. Two datasets are publicly available which can ensure the repeatability, and the remaining

dataset is provided by our collaborators. We used z-normalization to normalize the time series of each sensor.

[ROAD] This dataset [42] consists of times series of 963 road traffic sensors of San Francisco bay area freeways provided by the California Department of Transportation PEMS website [92]. Each sensor measured the occupancy rate of a road in a city for 15 months with a 10-minute sample interval. In total, there are 61.0 million data points. The data set is publicly available in the UCI Machine Learning Repository [8] and can be download freely [43].

[MALL] This dataset consists of time series of available car park lots in main shopping malls in Singapore. There are a total of 26 car parks with a record in every 10 minutes for 12 months. We duplicated every time series 40 times. In total, there are 1040 (26×40) sensor time series and 53.9 million data points (after duplication). The data is crawled from dataMall website [104] from September 2013 to September 2014.

[NET] This dataset is a time series of internet traffic data of a network backbone. It was collected for 3 months with a 5-minute sample interval. We duplicated this time series 1024 times. In total, there are 1024 (1×1024) sensor time series and 20.4 million data points (after duplication). The data set is publicly available in the DataMarket website and can be download freely [44].

4.5.2 Search Step: fast DTW k NN search

4.5.2.1 Competitors and experiment settings

We compared our DTW k NN search method of SMiLer, denoted as “SMiLer-GPU”, with three k NN search competitors, namely FastGPUScan, GPUScan [102] and FastCPUScan. FastGPUScan first computes the DTW distance between the queries and all segments with the Sakeo-Chiba constraint, and then uses the GPU fast selection method to obtain the k NNs. (The method is similar with SMiLer Index but has only Phase D and Phase E.) GPUScan [102] is similar to

FastGPUScan but without the Sakeo-Chiba constraint. FastCPUScan computes the DTW under the Sakeo-Chiba constraint with pruning criteria studied in [72; 94]. Although all the competitors are indeed scanning methods, it is widely recognized that the linear-scan family algorithms outperforms the complex index methods for k NN search in high dimensional space [94].

For all datasets, we randomly selected 100 master queries for each sensors and averaged the running time. Unless otherwise stated, for each sensor we did the Multiple k NN Search, i.e. invoking three item queries with length indicated in *ELV*. The running time shown in the experiment is the total time for all the sensors.

4.5.2.2 Evaluation with running time

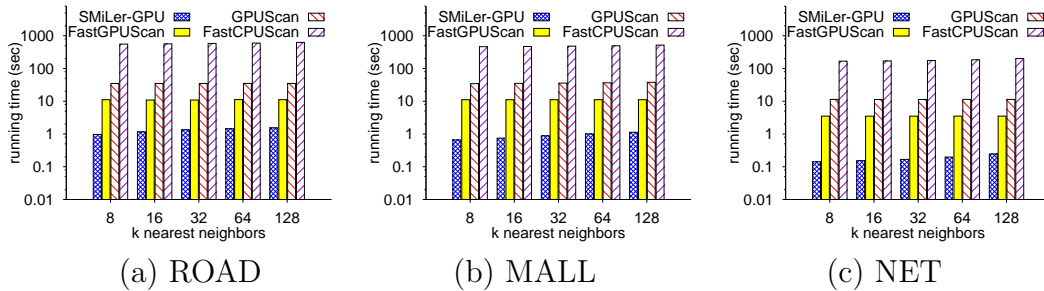


Figure 4.8: Time cost (log-scaled) of the Multiple k NN Search on all sensors with varying numbers of nearest neighbors k .

In Figure 4.8 (y-axis is log-scaled), we show the running time of the the Multiple k NN Search for different datasets with varying k . As we can see, SMiLer-GPU is still one order of magnitude faster for the Multiple k NN Search even when compared with the best competitor FastGPUScan. SMiLer-GPU only needs about 1 second to finish the search on all sensors, while FastGPUScan needs 10 seconds and FastCPUScan needs about 500 seconds. Besides, the time cost of SMiLer-GPU, FastGPUScan and GPUScan is quite stable with different numbers of nearest neighbors. The reason is that, as we stated in Section 4.3.3.5, we use a GPU-based distributive partitioning algorithm to select the best k candidates, which is

almost independent of the value of k .

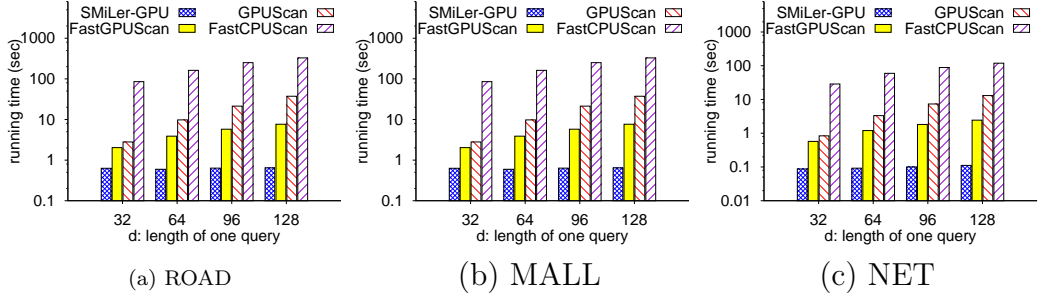


Figure 4.9: Time cost (log-scaled) of k NN search with varying query length d

Figure 4.9 shows the time cost for k NN search with varying query length d . In this case we used one item query per sensor with varying the query length d . Figure 4.9 shows that, as d increases, the time cost of SMiLer-GPU is relatively stable while the time cost of FastGPUScan, GPUScan and CPUScan becomes higher gradually. The reason is that longer item query requires more time to compute DTW; whereas the SMiLer Index can effectively filter many unpromising candidates to reduce the computation cost.

Table 4.3 exhibits the utility of the lower bound LB_{en} . On the ROAD and MALL datasets, the number of unfiltered candidates of LB_{en} is only about a half of that of LB_{EQ} and two thirds of that of LB_{EC} . The effect of LB_{en} on the NET dataset is not as significant as others because it has smaller variance. Thanks to the pruning power of LB_{en} , the computation time is reduced significantly.

Table 4.3: Effect of Enhanced Lower Bound. The “time” (in seconds) is the total time for all sensor, and the “number” is the number of unfiltered candidates per query per sensor.

data	ROAD		MALL		NET	
	time	number	time	number	time	number
LB_{EQ}	2.55	12558	1.38	6632	0.20	753
LB_{EC}	1.77	9206	1.17	5707	0.20	725
LB_{en}	1.35	6739	0.88	3677	0.17	516

4.5.3 Prediction Step: effectiveness of SMiLer

4.5.3.1 Competitors and experiment settings

SMiLer-AR (Section 4.4.2.1) and SMiLer-GP (Section 4.4.2.2) denote our semi-lazy prediction models with different instantiated predictors. We compared our method with three state-of-the-art prediction methods which are:

- HoltWinters, a popular statistical regression model for time series with periodical patterns [122; 64]. We used its implementation in “forecast” package of R [66]. To fully utilize the features of HoltWinters, after finishing prediction of one test input, we added this test data to training data to update the model. We set the period as one day, and parameters were determined by minimizing the squared error on the training data.
- Projected Sparse Gaussian Process (PSGP), a predictive analysis model with eager learning approach. PSGP is an approximation to the standard Gaussian Process [95] by projecting all information onto a set of “active points” [40]. We set the number of “active points” to 32, whose effect is also investigated in Section 4.5.4.2. We used an opensource project of PSGP [10].
- LazyKNN, a lazy learning prediction method for time series prediction [3], where the predicted value is an average of the k nearest neighbors weighted by the inverse of DTW distance. To make it comparable with other methods, we used the variance of the k NN results as the predicted variance.

We evaluate the prediction performance by two measures: mean absolute error (MAE), which is an average of the absolute errors between the predicted value and the true value; and mean negative log predictive density (MNLPD), which is an average of negative log of the density of the true value under the normal distribution predicted by the above methods. The MAE can evaluate the accuracy of the predicted result; while the MNLPD can assess the quality of the predictive uncertainty. In other words, the model with smaller MNLPD can not only predict

accurately, but also estimate a proper prediction confidence (standard deviation) which is very useful in decision making. For both measures, the smaller the value is, the better the method is.

For the ROAD dataset, we cut off a segment (i.e. leave-out testing) with 1000 points at the end of every time series. For the MALL and NET datasets, since there is duplication, we randomly cut off a segment (leave-out testing) with 1000 points from every time series. Then we made 200-step continuous prediction for each sensor along the segment. For PSGP, we only tested on 50 randomly selected time series, since the training time of PSGP for all sensors are too high to be acceptable (see Section 4.5.4.2 for more explanation).

4.5.3.2 Evaluation with MAE and MNLDP

Figure 4.10 shows the prediction performance of SMiLer (SMiLer-AR and SMiLer-GP) compared to HoltWinters, PSGP and LazyKNN with varying look-ahead step h . From Figure 4.10(a)(c)(e), we can see that SMiLer always has a smaller MAE compared with other methods. Taking the MAE on the ROAD dataset when $h = 15$ as an example, the MAE of HoltWinters is 5 times larger than that of SMiLer-GP; the MAE of PSGP is 3 times larger than that of SMiLer-GP; and the MAE of LazyKNN is 2 times larger than that of SMiLer-GP. Similar results can be obtained from the MALL and NET datasets.

An interesting point to note is that the MAE of SMiLer-AR is about 2 times larger than SMiLer-GP on the ROAD dataset (see Figure 4.10(a)), while their MAE is almost identical on the MALL and NET datasets (see Figure 4.10(c)(e)). This is due to the fact that the ROAD dataset contains more dynamic and irregular traffic information while the MALL and NET datasets have some seasonal patterns. Therefore, SMiLer-GP, which has a strong GP predictor, has innate advantage over SMiLer-AR on such complex time series data.

Figure 4.10(b)(d)(f) demonstrate that SMiLer-GP is significantly better than SMiLer-AR and LazyKNN under the MNLDP measure. From Figure 4.10(b)(d)(f),

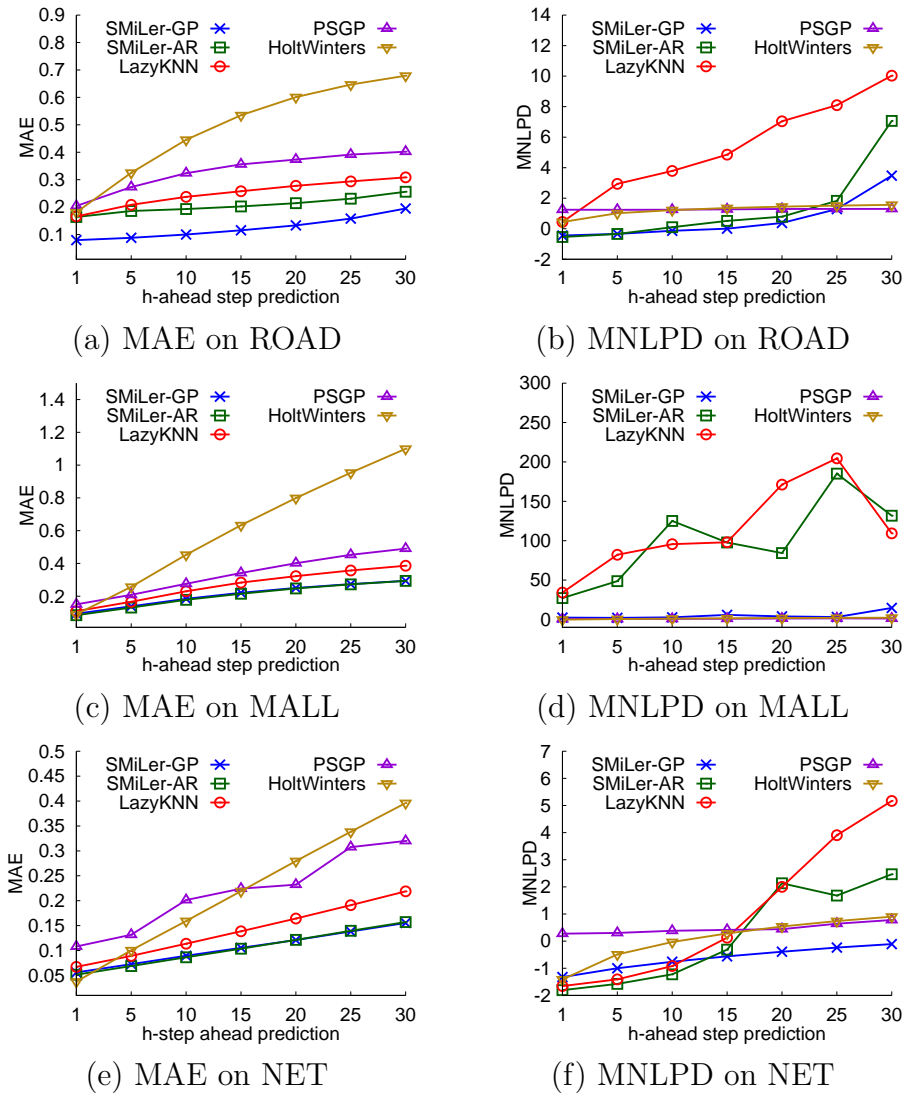


Figure 4.10: MAE and MNLDP with varying h -step ahead prediction

we can see that the MNLDP of LazyKNN and SMiLer-AR is much larger (and unstable in Figure 4.10 (d)) than the other methods. Indeed, LazyKNN and AR predictor innately lack the ability to estimate the predictive uncertainty, which is one of the major drawbacks of the lazy learning approach.

In Figure 4.10 (b)(d)(f), we can also see that SMiLer-GP is better than (at least is on par with) HoltWinters and PSGP under the MNLDP measure. The MNLDP of SMiLer-GP is always smaller than HoltWinters and PSGP except after step 30 on the ROAD and MALL datasets. We believe that the slight increasing of

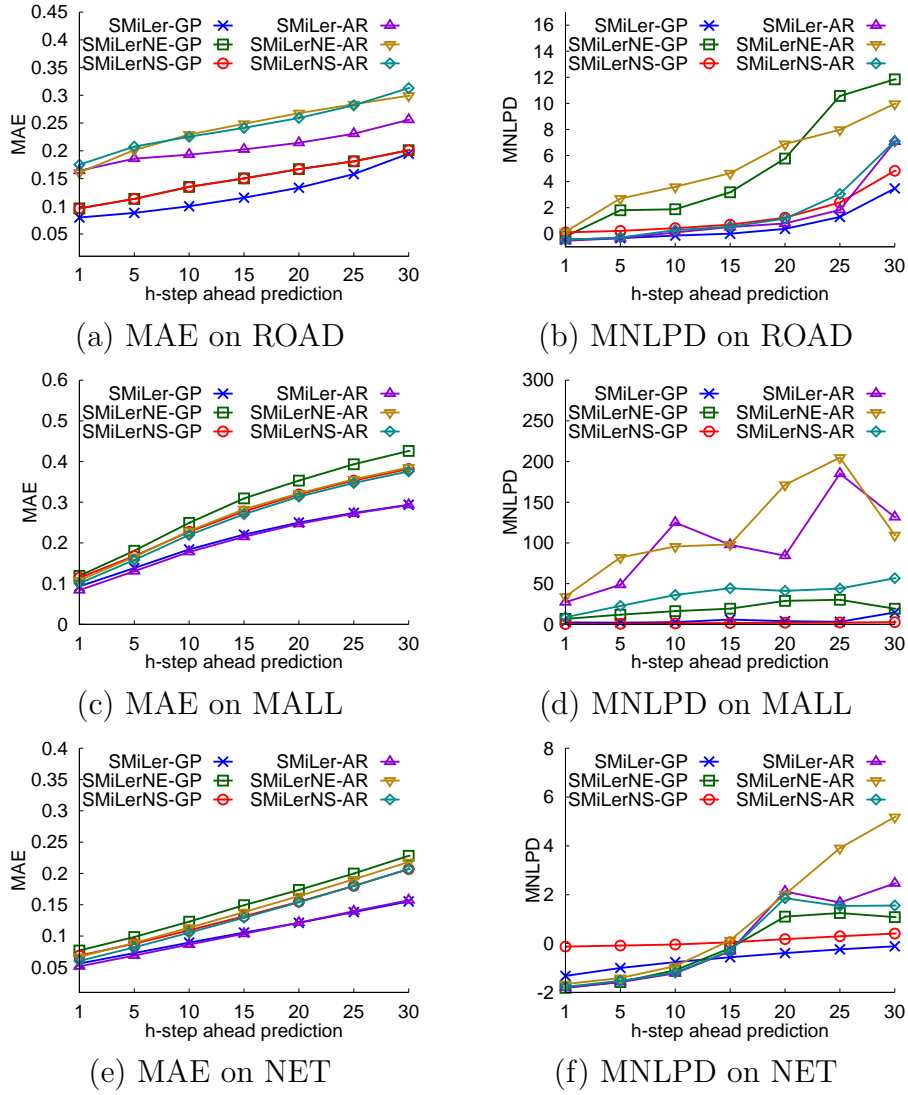


Figure 4.11: Effect of ensemble and self-correction prediction

the MNLPD of SMiLer-GP is due to the cumulative uncertainty for the long-step ahead prediction; PSGP and HoltWinters on the other hand can predict a large variance (which may not be interesting) in such cases based on the knowledge of the whole dataset. More comparison between SMiLer-GP and PSGP can be found in Section 4.5.4.2.

4.5.3.3 Effect of the ensemble and self-correcting method

Figure 4.11 reveals the effect of the ensemble and self-correcting prediction method of SMiLer. SMiLerNE-GP and SMiLerNE-AR denote the experimental result of SMiLer without the ensemble prediction (i.e. only one predictor instead of a matrix of predictors)². For SMiLerNE, we fixed the query segment length as $d = 64$ and the number of nearest neighbors as $k = 32$. SMiLerNW-GP and SMiLerNW-AR denote the experimental result of SMiLer with the ensemble prediction but without the self-correcting prediction. The other setting of SMiLerNW was the same as SMiLer’s, i.e. $ELV = \{32, 64, 96\}$ and $EKV = \{8, 16, 32\}$. As we can see from Figure 4.11, SMiLer-GP always has a better performance than SMiLerNW-GP and SMiLerNE-GP on all datasets under both measures (MAE and MNLPD). SMiLer-AR holds similar conclusion on the MAE measure. But under the MNLPD measure, since AR predictor lacks ability to estimate predictive uncertainty, there is no final conclusion.

4.5.4 Practicality of SMiLer

4.5.4.1 Running time of SMiLer

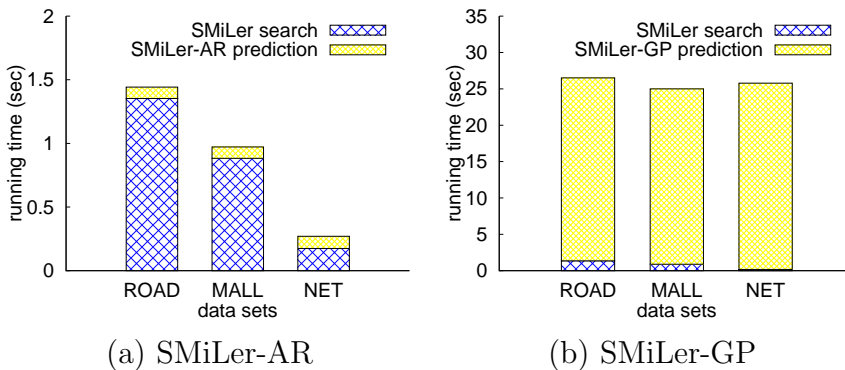


Figure 4.12: Total time cost of SMiLer (search and predict) on all sensors

We show the total running time of all sensors on SMiLer in Figure 4.12. We

²SMiLerNE-AR degenerates to LazyKNN in this case.

used a GPU-CPU platform to run SMiLer without multithreading on the CPU. As we can see, SMiLer-AR can make prediction within 1.5 seconds with a thousand sensors. Most of the time is spent on the SMiLer search step. SMiLer-GP can predict within 30 seconds on all the datasets and most of the time is consumed in prediction step. The higher time cost of SMiLer-GP than that of SMiLer-AR is due to the online training process of GP; while the paybacks are more accurate prediction results (lower MAE) and better estimation of predictive uncertainty (lower MNLPD). Note that, since the sample time interval is 5-10 minutes in the datasets, both SMiLer-AR and SMiLer-GP can process prediction requests for all sensors in real time.

SMiLer scales well in real-life applications. Since we build an independent prediction model for each sensor, SMiLer can scale linearly with the number of sensors. Besides, the running time of SMiLer-GP can be easily reduced by multithreading on multi-core architecture.

4.5.4.2 Comparison of PSGP and SMiLer

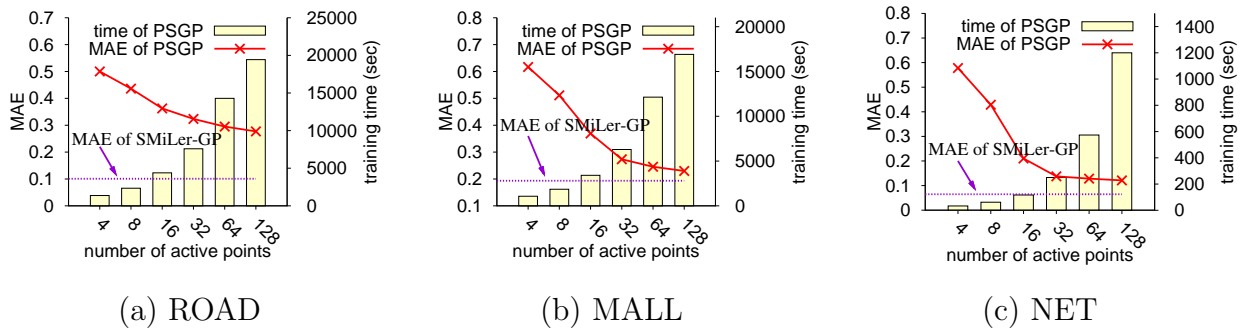


Figure 4.13: Comparison of PSGP and SMiLer-GP: average training time per sensor of PSGP and their MAE

We claim that SMiLer is more practical than all its competitors. Figure 4.10 shows that HoltWinters can estimate predictive uncertainty (low MNLPD) but its accuracy is bad (high MAE); LazyKNN may have a medium accuracy (relative low MAE) but it cannot estimate the predictive uncertainty (high MNLPD). It

seems that PSGP has potential to achieve better prediction performance since its accuracy can be improved by increasing the number of “active points”. Next, we reveal the impracticality of this solution.

Figure 4.13 indicates why SMiLer is more practical than PSGP in real-life applications. For each dataset, we randomly selected 50 sensors to make prediction using PSGP and averaged their training time and MAE. In Figure 4.13, when we vary the number of active points, the left y-axis shows the MAE of PSGP and the right y-axis shows the average training time of PSGP for each sensor. The MAE of SMiLer-GP averaged on such 50 sensors is also illustrated in Figure 4.13 with violet dash line. We can see that, after the number of active points become larger than a certain threshold (e.g. 32), the marginal improvement of MAE is small, but the increase of the computational time is exponential. For example, on the ROAD dataset, the total training time for its 963 sensors with 128 active points should be about 200 days ($18000 \times 963 = 17334000$ seconds). As we illustrated in Figure 4.13, SMiLer-GP still has lower MAE than PSGP on all the datasets even if we allow such an expensive training process for PSGP.

4.6 Comparison of R2-D2 and SMiLer

In this section, we compare the performance of our semi-lazy trajectory prediction method (denoted by **R2-D2**) with the semi-lazy time series prediction method (denoted by **SMiLer-GP**) on the sensor time series data. Then we discuss the difference of time series prediction and trajectory prediction for model selection.

4.6.1 Datasets preparation

We use three data sets in the experiment of our semi-lazy time series prediction method (i.e. SMiLer-GP), which are: ROAD (road traffic sensors data), MALL (available car park lots in shopping malls) and NET (internet traffic data of a network backbone). More details about these datasets can be found in Section

4.5.1.2. We convert the time series data into trajectory data. For every point of time series, we append one external dimension to the point with a constant value 0. In the sensor data, one sensor is equivalent to one dynamic environment. Therefore, we only test them on one sensor of the data sets. For R2-D2, there are many trajectories of different moving objects while there is only one long time series for a sensor. To overcome this problem, we divide the time series into different segments whose length is equal to one day (one day is the period of the sensor). To measure the prediction performance, we make 50 prediction queries on the data set and average their distance error and prediction rate.³ For R2-D2, we set the length of the backward window as 6.

4.6.2 Experiment results

Figure 4.14 and Figure 4.15 show the prediction error of R2-D2 compared with SMiLer-GP. In Figure 4.14, to make R2-D2 and SMiLer-GP comparable we set the probability confidence threshold as $\theta = 0$; whereas in Figure 4.15, we set the probability confidence threshold as $\theta = 0.05$. We can see that with the increasing of the predicted path length, the distance error of R2-D2 becomes larger and larger in both figures. The probability threshold θ can reduce the prediction error, especially for the MALL dataset. Note that if the length of the predicted path is larger than 5 ($h \geq 5$), the distance error of R2-D2 in Figure 4.15 becomes unstable. The reason is that many prediction queries cannot make predictions after step 5 since its probability confidence threshold is smaller than 0.05.

The error of the R2-D2 is determined by whether we can generate correct states (i.e. cluster) whose center is the predicted result. For time series data, the reading value of the sensor may be distributed in the whole domain without obvious clusters. Therefore, the prediction result of R2-D2 is not as good as SMiLer-GP. It is worth noting that R2-D2 has better performance on the MALL

³We use 100 prediction queries for the ROAD dataset since its prediction rate is too small (which is smaller than 0.5).

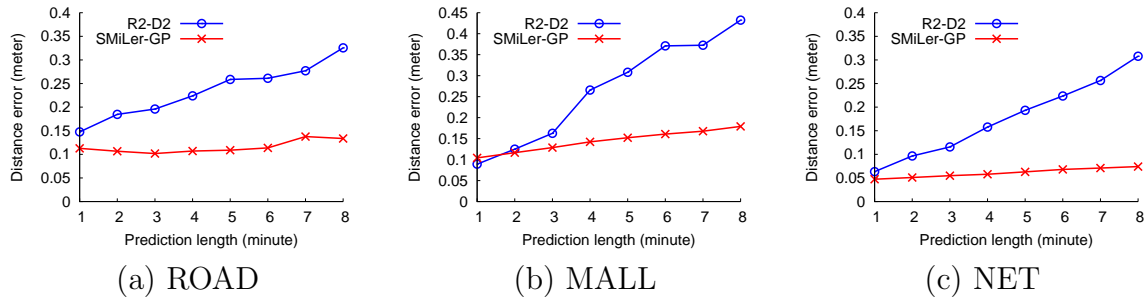


Figure 4.14: Distance error of R2-D2 with confidence threshold $\theta = 0$

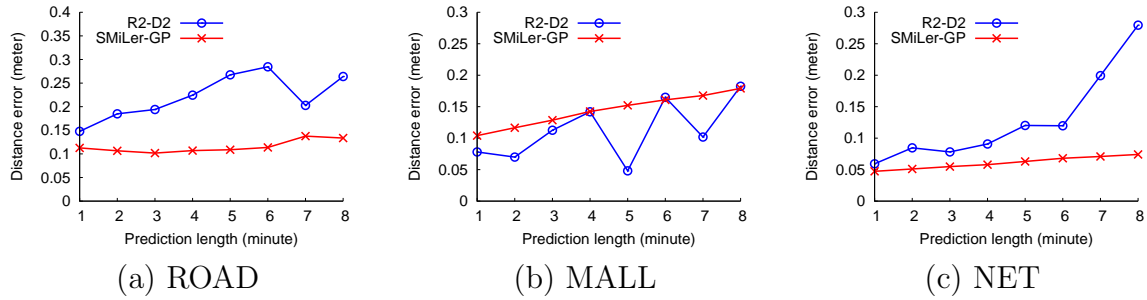


Figure 4.15: Distance error of R2-D2 with confidence threshold $\theta = 0.05$

dataset than on the ROAD and NET datasets. It is because, with the MALL data set having better seasonal patterns than the ROAD and NET datasets, the values of time series in MALL dataset may closely gather in a small interval which can be recognized by the cluster method of R2-D2.

Figure 4.16 shows the prediction rate of R2-D2 on the datasets. The prediction rate of R2-D2 falls quickly with the increase of the predicted path length. R2-D2 predicts a sequence of possible values, and the confidence probability measures the probability of the whole sequential values instead of one value. Therefore, the probability confidence descends dramatically with the increasing of the predicted path length.

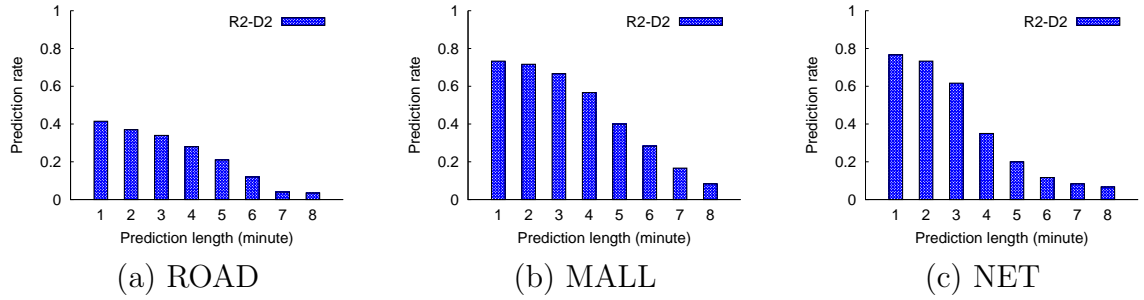


Figure 4.16: Prediction rate of R2-D2 with confidence threshold $\theta = 0.05$

4.6.3 Model selection for time series prediction and trajectory prediction

In this section, we discuss why we use different models to do predictive analysis for time series and trajectories. We explain the reasons from two perspectives: the application purpose and the data property.

4.6.3.1 Application purpose

The time series prediction and trajectory prediction have different prediction purposes based on their applications. As we will clarify in details later, the time series prediction concerns the variance of the predicted result; whereas the trajectory prediction more concerns the probability of the predicted path (a sequence of possible locations). Therefore, we use the Gaussian Process model for time series prediction to estimate the variance of the predicted results, and use the Hidden Markov Model for trajectory prediction to estimate the probability of the predicted path.

For time series prediction, this prediction variance is useful for decision making. With the Gaussian Process model, we can predict the value with its variance of the sensor at a certain time. For example, we predict that there are 10 available car park lots in a shopping mall after two hours. If the standard deviation of the predicted result is 1, it may be sure that there is available car park lots after two hours. However, if the standard deviation of the predicted result is 100, we may

alarm the coming users that there may not be enough car park lots for them. In this case, the Gaussian Process model is more suitable for time series prediction which can estimate the variance for the predicted value.

In contrast, for trajectory prediction the probability of the predicted path is useful for applications. With the Hidden Markov model, we can estimate the probability of a predicted path for a sequence of future time steps. For example, with setting a minimum probability confidence as 0.2, we can predict the path of a car which will pass a restaurant. If we reduce the probability confidence threshold as 0.1, we can predict a longer path of the car which will pass a restaurant and a gas station. With the estimated probability, the administrator can determine either to only send the message of the restaurant to drivers ahead of time, or to send all the messages of the restaurant and the gas station to drivers. Whereas the variance of the predicted locations is not as important as the one of time series prediction since the distance from the predicted location to the Point of Interest does not vary too much. In this case, the Hidden Markov Model is more suitable for trajectory prediction which can estimate the probability for the predicted path (a sequence of possible locations).

4.6.3.2 Data property

We also need to consider the data property of time series and trajectories for predictive model selection. The continuous data distribution of time series data determines that the Gaussian Process model is superior to the Hidden Markov Model for predictive analysis. The sensor always monitors an environment which should change gradually and continuously. For example, the temperature of an environment may change gradually and continuously from one value to another. Then the value of time series data may distribute the whole domain without a clear boundary. In this case, it is difficult and unnatural to discretize the values of time series into different clusters (or states). In this case, the Discrete Hidden Markov Models (HMMs) is not suitable to the time series prediction involving continuous

data. Whereas, the Gaussian Process can predict the value as continuous function in a probabilistic manner based on the correlations among all the individual points, where the problem of discretization is conveniently avoided.

In contrast, the discrete distribution of trajectory data determines that the Hidden Markov Model is superior to the Gaussian Process model for predictive analysis. The trajectories of moving objects usually follow some routes (such as roads in space). For example, the trajectories of cars in urban space are constraint by the road network. In this case, the future points of trajectories can be divided into different clusters to form the states of the Hidden Markov Model; while the Gaussian Process model may predict wrong results since the aggregation of all the points may be outside any of the clusters. In this case, the Hidden Markov Model can generate more meaningful prediction results.

4.7 Conclusions

In this chapter, we present SMiLer, which is a semi-lazy prediction system for sensors. The core idea of this system is to employ the “semi-lazy” learning approach to time series prediction. To make our system feasible, two challenging problems are solved, which are a fast k -nearest neighbors search under DTW and a semi-lazy time series prediction model. For the former problem, we depicted a GPU-based index for fast DTW k NN search. For the latter one, we devised a semi-lazy time series prediction model integrating the ensemble and self-correcting prediction. We also resorted to the Gaussian Process as an instantiated predictor of our model.

Extensive experiments on real datasets demonstrate that SMiLer does efficiently perform time series prediction with high accuracy and good predictive uncertainty. SMiLer can also scale well and carry out prediction in real time. A discussion about the model selection for trajectory prediction and time series prediction is also expounded in this chapter.

Chapter 5

Dynamic Itinerary Recommendation for Traveling Services

5.1 Introduction

Traveling market is divided into two parts. For casual customers, they will pick a package from local travel agents. The package, in fact, represents a pre-generated itinerary. The agency will help the customer book the hotels, arrange the transportations and pre-order the tickets of museums/parks. It prevents the customers from constructing their personalized itineraries, which is very time-consuming and inefficient. For instance, Figure 5.1 lists a four-day package to Hong Kong, provided by a Singapore agency. It covers the most popular POIs for a first-time traveler and the customers just need to follow the itinerary to schedule their trips.

Most existing works on itinerary recommendation employ an eager learning method, which takes a two-step scheme. They first adopt the data mining algorithms to discover the users' traveling patterns from their published images, geo-locations and events [97; 39; 34]. Based on the the relationships of those historical data, new itineraries are generated and recommended to users [106; 32; 129].

Although the travel agencies provide efficient and convenient services, for ex-

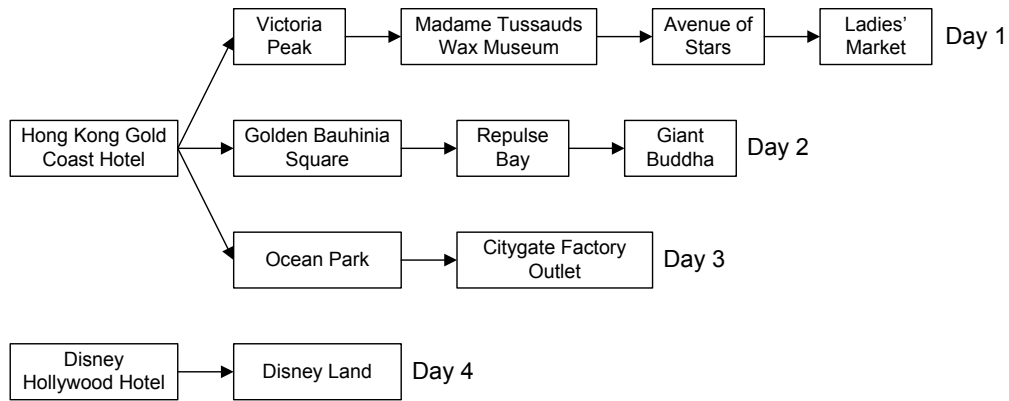


Figure 5.1: A 4-Day Trip to Hong Kong

perienced travelers, the itineraries provided by the travel agents lack of customization and cannot satisfy individual requirements. Some interested POIs are missing in the itineraries and the packages are too expensive for a backpacking traveler. Therefore, they have to plan their trips in every detail, such as selecting the hotels, picking POIs for visiting and contacting the car rental service.

Therefore, to attract more customers, travel agency should allow the users to customize their itineraries and still enjoy the same services as the pre-defined itineraries. However, it is impossible to list all possible itineraries for users. A practical solution is to provide a dynamic itinerary recommendation service. The user lists a set of interested POIs and specifies the time and money budget. The itinerary recommendation service returns top- k itineraries package satisfying the requirements. In the ideal case, the user selects one of the recommended k -day itineraries package as his plan and notifies the agent.

However, none of current itinerary planning algorithms (e.g., [50] and [11]) can recommend a ready-to-use itinerary plan, as they are based on various assumptions. First, current planning algorithms only consider a single day's trip, while in real cases, most users will schedule an n -day itinerary (e.g., the one shown in Figure 5.1). Generating an n -day itinerary is more complex than generating a single day one. It is not equal to constructing n single day itineraries and combining

them together, as a POI can only appear once. It is tricky to group POIs into different days. One possible solution is to exploit the geo-locations, e.g., nearby POIs are put in the same day’s itinerary. Alternatively, we can also rank POIs by their importance and use a priority queue to schedule the trip.

Second, the travel agents tend to favor the popular POIs. Even for a city with a large number of POIs, the travel agents always provide the same set of trip plans, composed with top POIs. However, those popular POIs may not be attractive for the users, who have visited the city for several times or have limited time budget. It is impossible for a user to get his personal trip plan. The travel agent’s service cannot cover the whole POI set, leading to few choices for the users. In our algorithm, we adopt a different approach by giving high priorities to the selected POIs and recommending a customized trip plan on the fly.

Third, suppose we have N available POIs and there are m POIs in each single day’s itinerary averagely. We will end up with $\frac{N!}{(N-m)!m!}$ candidate itineraries. It is costly to evaluate the benefit of every itinerary and select several sets of representative itineraries (i.e. trip plan) to recommend to different groups distinguished by ages, incomes, occupations, etc. Therefore, in [50] and [11], some heuristic approaches are adopted to simplify the computation. However, the heuristic approaches are based on some assumptions (e.g., popular POIs are selected with a higher probability). They only provide limited numbers of itineraries and are not optimized for the backpack traveler, who plans to have a unique journey with his own customized itinerary.

Last but not the least, handling new emerging POIs are tricky in previous approaches. The model needs to be rebuilt to evaluate the benefit of including the new POIs into the itinerary. For systems based on the users’ feedback [11], we need to collect the comments for the new POIs from the users, which is very time consuming.

To address the above problems, we propose a novel dynamic itinerary recommendation system based on our semi-lazy learning approach. Instead of generating

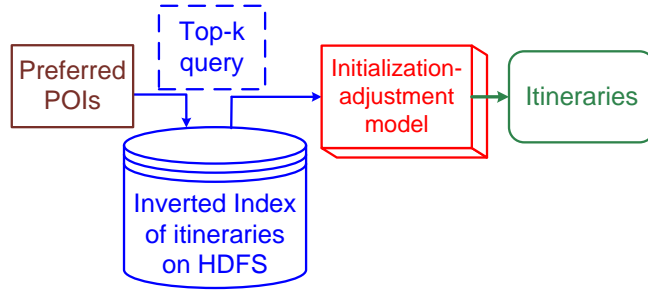


Figure 5.2: Framework of the semi-lazy learning approach to itinerary recommendation.

the ready-to-use itineraries in a pre-processing time like the eager learning method, the design philosophy of our approach is to dynamically recommend the itinerary based on users’ preference on the fly. Figure 5.2 shows how the itinerary recommendation system fits in the framework of our “semi-lazy” learning approach. We first generate all possible itineraries as a reference database for the itinerary recommendation. For a user’s requirement, we will select top- k best itineraries from the indexes. Instead of recommending the searched itineraries directly, we apply an initialization-adjustment model to refine the itineraries to emphasize the user’s selection.

In a nutshell, our semi-lazy learning approach can be seen as a transformation from the Team Orienting Problem (TOP) into the weighted set-packing problem which has efficient approximate algorithms. The initialization-adjustment model is indeed an approximate initialization-adjustment algorithm for the set-packing problem.

Figure 5.3 shows a detailed overall architecture of our itinerary recommendation system. We reduce the overhead of constructing a personalized itinerary for the traveler; and we provide a tool for the agents to customize their services. Specifically, our approach can be summarized as follows.

We first iterate all candidate single-day itineraries using a parallel processing framework, MapReduce [48]. The results are maintained in the DFS (Distributed File System) and an inverted index is built for efficient itinerary retrieval. To construct a multi-day itinerary, we need to selectively combine the single itineraries.

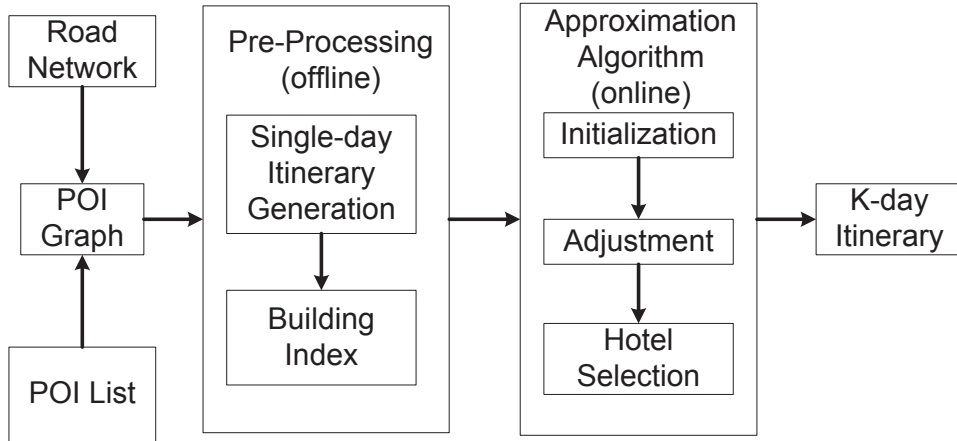


Figure 5.3: The Detailed architecture of dynamic itinerary recommendation system.

The preprocessing stage, in fact, transforms the team orienteering problem into a set-packing problem [36], which has well-known approximated algorithms.

In the online stage, we design an approximate algorithm to generate the optimal itineraries. The approximate algorithm adopts the *initialization-adjustment* model. After retrieving the k -best itineraries from the inverted index, the initialization-adjustment approximate model are used to refine the itineraries. A theoretic bound is guaranteed for the quality of the approximate result.

To evaluate the proposed approach, we use the real data from Yahoo Travel¹. The experiments show that our approach can efficiently return high quality customized itineraries. The rest of the chapter is organized as follows. In Section 5.2, we formalize the problem and give an overview of our approach. Then, Section 5.3 and Section 5.4 present the pre-processing stage and online stage of our approach, respectively. We evaluate our approach in Section 5.5. Finally, the chapter is concluded in Section 5.6.

¹<http://travel.yahoo.com/>

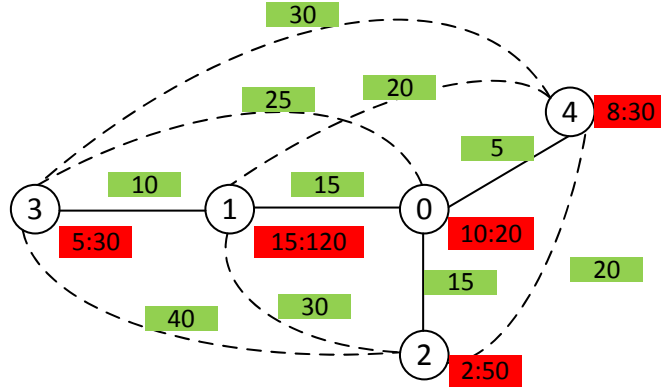


Figure 5.4: POI Graph

5.2 Overview

5.2.1 Problem statement

In the itinerary recommendation system, the user selects a set of interested POIs, S_p , and asks the system to recommend a k -day itinerary. We use (S_p, k) to denote a user's request. To model the recommendation problem, we organize the POIs into a *POI Graph*.

Definition 5.2.1. POI Graph

In the POI graph $G = (V, E)$, we generate a vertex for each POI and every pair of vertexes are connected via an undirected edge in E . In G , the vertex and edge have the following properties:

1. $\forall v_i \in V$, $w(v_i)$ denotes the weight (importance) of the POI and $t(v_i)$ is the average time that tourists will spend on the POI.
2. $\forall (e_x = v_i \rightsquigarrow v_j) \in E$, $t(e_x)$ is the cost of the edge, computed as the average traveling time from v_i to v_j .

Figure 5.4 shows a POI graph with 5 nodes. Each node denotes a POI and has two properties: the weight and travel time (shown in the red blocks). The nodes are connected via weighted edges. The edge's weight is set to the average

traveling time for the shortest path between the corresponding POIs in the map. In fact, there are two types of edges. The first type represents that the two nodes are directly connected in the map (no other POI exists in their shortest path, e.g., $0 \rightsquigarrow 1$). The second type contains multiple shortest pathes in the map (e.g., $0 \rightsquigarrow 3 = (0 \rightsquigarrow 1) \oplus (1 \rightsquigarrow 3)$). Transforming the POI graph into a complete graph reduces the processing cost of our itinerary algorithm.

The definition of POI graph assumes that the costs of edges are symmetric. Namely, the traveling time from v_i to v_j is equal to the time from v_j to v_i . In fact, as our approach does not rely on the assumption, it can be directly applied to the case of non-symmetric cost (e.g., traffics are different for $v_i \rightsquigarrow v_j$ and $v_j \rightsquigarrow v_i$).

Let $w(v_i)$ denote the weight (importance) of POI v_i . The initial weight of v_i is generated from the users' reviews (e.g. in Yahoo Travel, users can specific a score ranging from 0 to 5 for each POI. We accumulate the scores and use the average values as the initial weight).

Users can also select a set of preferred POIs, denoted as S_p . Given a request (S_p, k) , if v_i is selected by the request ($v_i \in S_p$), we intentionally increase its weight to $\alpha(w(v_i) + 1)$, where α can be set to an arbitrary integer. The intuition is that user-selected POIs are far more important than any other POIs.

For a request (S_p, k) , if $k = 1$, we just need to generate a single day itinerary. A single day itinerary is represented as $L = v_0 \rightsquigarrow \dots \rightsquigarrow v_n \rightsquigarrow h_j$, where h_j is a hotel POI. The elapsed time is estimated as

$$t(L) = \sum_{i=0}^n t(v_i) + \sum_{i=0}^{n-1} t(v_i \rightsquigarrow v_{i+1}) + t(v_n \rightsquigarrow h_j)$$

In the rest discussion, we remove the hotel part and focus on how to merge the POIs into itineraries. After all other POIs are fixed, we will solve the hotel selection problem.

Assume there are H available hours per day for traveling. The itinerary L must satisfy that $t(L) \leq H$. For a common traveling request, it always includes a

k -day ($k \geq 1$) trip, which is defined as:

Definition 5.2.2. k -day Itinerary

Given a POI graph G and time budget k , a valid k -day itinerary consists of k single-day itineraries, $\mathcal{L} = \{L_1, L_2, \dots, L_k\}$, which satisfies that

1. $\forall i \forall j, L_i$ and L_j do not share a POI.
2. $t(L_i) \leq H$ for all $1 \leq i \leq k$.

Based on the POIs included in the itinerary, the score of a k -day itinerary can be computed as:

$$w(\mathcal{T}) = \sum_{i=1}^k \sum_{v_j \in L_i} w(v_j) \tag{5.1}$$

The goal of our itinerary recommendation algorithm is to find the k -day itinerary with the highest score. However, we will show that finding the optimal itinerary is an NP-complete problem, which is equivalent to the team orienteering problem (TOP) [25]. Even approximate algorithm within constant factor does not exist. Existing work (e.g., [4]) solves the problem by employing heuristic algorithms, which may generate arbitrarily bad results.

5.2.2 System architecture

In our system, instead of trying to propose new algorithms for the TOP, we transform the optimal itinerary planning problem into a set-packing problem by an offline MapReduce process and an approximate algorithm is applied to solve the set-packing problem. If the maximal number of POIs in the single day itinerary is bounded by m , the optimal result can be approximated within factor of $\frac{2(m+1)}{3}$ (m is the maximal number of POIs in each single-day itinerary).

Figure 5.3 shows the architecture of our itinerary recommendation system. In the first step, POI graph is constructed via the road network and POI coordinates. The Google Map’s APIs are used to evaluate the distance between POIs. The

average elapsed time of a POI is estimated from users' blogs and travel agency's schedules.

After the POI graph is constructed, a set of MapReduce jobs are submitted to iterate all possible single-day itineraries in the pre-processing. The number of itineraries is exponential to the number of POIs. However, using parallel processing engine, such as MapReduce, we can efficiently generate all itineraries in an offline manner. To speed up the single-day itinerary retrieval, an inverted index is built. Given a POI, all single-day itineraries involving the POI can be efficiently retrieved.

For a user request (S_p, k) , POIs' weights are updated based on S_p and we compute the scores for each single-day itinerary. The problem of finding optimal k -day itinerary is transformed to select k single-day itineraries that maximize the total score. We show that the new problem can be reduced to the weighted set-packing problem, which has polynomial approximate algorithms. Therefore, we simulate the approximate algorithm for set-packing problem to generate the k -day itinerary. The algorithm uses a greedy strategy to create an initial solution, which is continuously refined in the adjustment phase. The adjustment phase scans the index to find a potentially better solution.

In the next two sections, we first present how we apply the MapReduce framework to generate and index the single-day itineraries. The parallel processing engine enables us to search the optimal solution in a brute-force manner. Next, we show that, after the preprocessing, the complexity of TOP is reduced and approximate algorithms are available.

5.3 Pre-processing

The preprocessing includes two steps. In the first step, a set of MapReduce jobs are submitted to produce all possible single-day itineraries. In the second step, the single-day itineraries are reorganized as an itinerary index, which supports

efficient itinerary search.

5.3.1 Intractability of optimal itinerary algorithm

Given a user request (S_p, k) , the goal of itinerary recommendation algorithm is to provide an itinerary, which ranks highest among all possible itineraries. The score of the itinerary is computed based on the POI weights. However, as shown in the following theorem, this is an NP-complete problem and no polynomial time algorithm exists.

Theorem 5.3.1. *Finding optimal k -day itinerary in a POI graph $G = (V, E)$ is an NP-complete problem.*

Proof. The optimal k -day itinerary can be reduced to the team orienteering problem (TOP) [25], which is a well known NP-complete problem. Consider a simple scenario where

1. k vehicles are created, which start from the same position.
2. Each vehicle has a time limit (1 day) for traveling the POIs.
3. Each vehicle collects the profit by visiting the POIs.
4. The POI accessed by a vehicle will not be considered by other vehicles.
5. The POI's profit is equal to its weight.

The TOP is to find the traveling plan that generates the most profits. The results of the TOP are also the best k -day itinerary. \square

Due to the complexity of TOP, it is impossible to find the exact solution. Instead, previous work focus on proposing heuristic algorithms. The basic idea is to generate an initial plan and then adjust it based on some heuristic rules. Those

algorithms have three drawbacks. First, the heuristic algorithms need many iterations to get a good enough result, which incur high computation cost [114]. Second, the adjusting rules are too complicated and the potential gains are unknown. Finally, there is no bound of the approximate result, which may be arbitrarily bad in some cases.

We reduce the complexity of the TOP by transforming it into a set-packing [60] problem. As the transformation is done in an offline manner, the performance of online query processing is not affected.

5.3.2 Single-day itinerary

Algorithm	5.1: map(Object <i>key</i>, Text <i>value</i>, Context <i>context</i>)
------------------	---

```

// we allow maximally m-round MapReduce jobs, i.e. the maximally
length of path is m
//value: existing path, each MapReduce job tries to add one more POI to
the path
1: Path P = parsePath(value)
2: for i = 0 to POIGraph.POINumber do
3:   if isConnected(P, i) and !P.contains(i) then
4:     Path newPath = P.append(i)
5:     cost = P.cost + POIGraph.getCost(P.endPOI, i)+POIGraph.getCost(i)
6:     weight = P.weight + POIGraph.getWeight(i)
7:     newPath.cost=cost
8:     newPath.weight=weight
9:     if newPath.cost ≤ H then
10:       Key newKey = parsePath(newPath).sort();
11:       context.collect(newKey, newPath)
12:     else
13:       DFS.write(resultFile, P)
14:     end if
15:   end if
16: end for

```

The basic idea of transformation is to iterate all possible single day itineraries. This is done by a set of MapReduce jobs. In the first job, we generate $|\mathcal{P}|$ initial itineraries for the POI set \mathcal{P} . Each initial itinerary only consists of one POI.

Algorithm 5.2: reduce(Key *key*, Iterable *values*, Context *context*)

```
1: bestCost = ∞
2: bestPath = NIL
3: for Path P: values do
4:   if P.cost < bestCost then
5:     bestPath = P
6:     bestCost = P.cost
7:   end if
8: end for
9: context.collect(key, bestPath)
```

Iteratively, the subsequent MapReduce job tries to add one more POI to the itineraries. If no more single day itineraries can be generated, the process terminates. In current implementation, we allow maximally m MapReduce jobs in the transformation process to reduce the overheads. Therefore, a single day itinerary contains at most m POIs. This strategy is based on the assumption that users cannot visit too many POIs in one day. In our crawled dataset from Yahoo travel, setting m to 10 is enough for Singapore data, which includes more than 400 POIs. Only a few single-day itineraries can contain more than 10 POIs.

Algorithm 5.1 and 5.2 show the pseudo codes of the MapReduce job. The *mappers* load the partial paths from the DFS, which are generated in the previous MapReduce jobs. we try to append new POI to existing itineraries. For each new path, we test whether it can be completed within one day. If not, we will discard the new path. If the old path cannot result in any new path, we will output the old path. For the last MapReduce job (the m th job), all the candidate itineraries are used as the results. The output key-value pair is using the sorted POIs in the itinerary as the key.

In the *mappers*, to compute the weight and cost of new itinerary, we load the POI graph table from the DFS. As the graph table is small, each *reducer* maintains

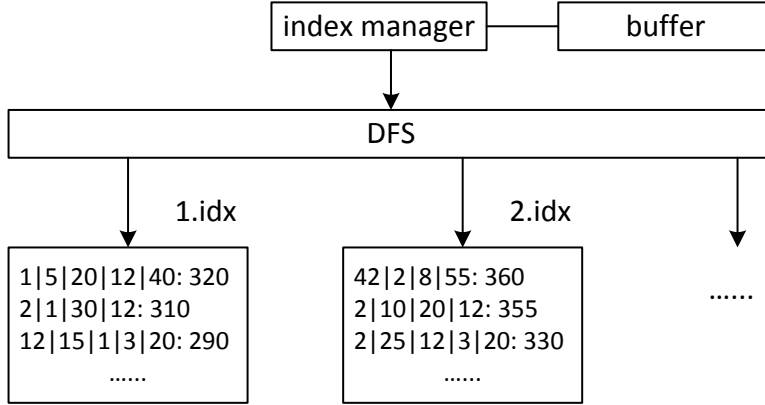


Figure 5.5: Itinerary Index

a copy in its memory. The table's schema follows

$$(S_POI, E_POI, S_weight, E_weight, S_cost, E_cost, cost)$$

where S_POI and E_POI denote the two POIs linked by a specific edge, $cost$ is the traveling cost from S_POI to E_POI, and S_POI is the primary key of the table.

In the *reducers* (Algorithm 5.2), we select the path with smallest cost of paths with the same POIs. In each reducer, all the paths have the same POIs. We only keep the path with smallest cost and output such path for the next round. Note that since all the paths have the same POIs, these paths have the same weight.

After all itineraries have been generated, a clean process is invoked to remove the duplication. For two itineraries ($L_0 = v_0 \rightsquigarrow \dots \rightsquigarrow v_n$ and $L_1 = v'_0 \rightsquigarrow \dots \rightsquigarrow v'_n$), L_0 contains L_1 , iff

$$\forall v'_j \in L_1 \rightarrow \exists v_i \in L_0 (v_i = v'_j)$$

Namely, all POIs in L_1 are also included by L_0 . If L_0 contains L_1 , we will only keep L_0 , as it provides more POIs for the users.

5.3.3 Itinerary index

To efficiently locate the single day itineraries, an inverted index is built. The key is the POI and the values are all itineraries involving the POI. By scanning the index, we can retrieve all the itineraries. Figure 5.5 illustrates the index structure. We create an index file for each POI in the DFS. The file includes all single itineraries involving the POI, which are sorted based on their weights. For example, in Figure 5.5, “1.idx” contains all itineraries for the first POI. The itinerary ”1|5|20|12|40” is the most important itinerary in the index file with weight 320.

The inverted index is constructed via a MapReduce job. Algorithm 5.3 and 5.4 show the process. The *mappers* load the single-day itinerary and generate key-value pairs for each involved POI. The *reducers* collect all itineraries for a specific POI and sort them based on the weights before creating the index file. In our system, the size of index file may vary a lot. Some POI may have an extremely large index file, due to its popularity and short visit time. In *reducers*, those POIs may result in the exception of memory overflow in the sorting process. To address this problem, in the *map* phase, instead of using the POI as the key, we generate the composite key by combining the POI and the itinerary weight.

Algorithm 5.3: `map(Object key, Text value, Context context)`

```
//value: single day itinerary
1: Itinerary it = parse(value)
2: for i = 0 to it.POISize() do
3:   int nextPOI = it.getNext(i)
4:   Key key = new CompositeKey(nextPOI, it.weight/bucketSize)
5:   context.collect(key, it)
6: end for
```

In particular, we partition the itineraries into n buckets. The bucket ID is used as a part of the composite key. In this way, we split the itineraries of a POI into n groups and each group can be efficiently sorted in the memory. Each group will result in an index file. However, it is not necessary to merge the files, as the files

Algorithm 5.4: reduce(Key *key*, Iterable *values*, Context *context*)

```
1: CompositeKey ck = key, Set s =  $\emptyset$ 
2: for Itinerary it: values do
3:   s.add(it)
4: end for
5: sort(s)
6: DFSFile f = new DFSFile(ck.first+”_”+ck.second)
7: f.write(s)
```

are partitioned based on the weights. By scanning all files from the n th bucket to the 1th bucket, we can get a sorted list for all itineraries involving a POI.

To simplify the index manipulation, an index manager is built in our query engine. The index manager only provides one interface $scan(POI)$, where POI denotes the owner of the index. The interface returns an iterator, which can be used to retrieve all itineraries of the POI. A memory buffer is established to cache the used itineraries and the LRU strategy is applied to maintain the buffer.

5.3.4 Discussion: why MapReduce

Although the input dataset (POI graph) is small in size, the partial results of the possible itineraries are extremely large (more than 100G or even 1T). The computation is also intensive, which can not be completed by a single machine. MapReduce is the solution to partition the partial results and generate the itineraries in parallel. Its advantages are two-fold:

1. Parallel computing effectively reduces the running time of preprocessing. The search space explodes when the number of POIs and traveling days increases. It is impractical to generate all possible itineraries. But by exploiting the power of MapReduce, we can share and balance the workload between multiple machines. The scalability is achieved by adding more nodes into the cluster. In our experiment, the running time of preprocessing is significantly reduced with the number of nodes (See Figure 5.13)

2. MapReduce algorithms can remove the duplicated itineraries in a simple way. In Algorithm 5.2, by leveraging the framework of MapReduce, we map all the itineraries with the same POIs into the same reducer and only keep one itinerary with the lowest cost. This approach can prune the low-benefit partial itineraries as early as possible and lead to less input for the next round of computation.

5.4 Initialization-Adjustment algorithm

After the itinerary indexes are constructed, the user request (S_p, k) can be processed by selecting k best itineraries from the indexes. Namely, the problem of generating optimal k -day itinerary is transformed into a weighted set-packing problem as shown in the following theorem.

Definition 5.4.1. Weighted Set-Packing Problem

In a universe \mathcal{U} , we assume that each element in \mathcal{U} has a weight and the weight of any subset of \mathcal{U} equals to the sum of the element weights in the subset. Given a family \mathcal{S} of \mathcal{U} 's subsets, the set-packing problem is to select a subfamily \mathcal{S}' from \mathcal{S} , where all subsets in \mathcal{S}' are disjoint and the weight of \mathcal{S}' is maximal among all possible selections.

Theorem 5.4.1. *Finding optimal k -day itinerary can be reduced to the weighted set-packing problem.*

Proof. By solving the set-packing problem, we can also get the optimal k -day itinerary, as

1. Each single day itinerary can be considered as a subset of the POI set \mathcal{P} .
2. The subsets selected by the set-packing problem are disjoint and hence in the k -day itinerary, we will not visit a POI twice.

1.idx	2.idx	3.idx	4.idx
1 2 4 X ₁	5 2 4 X ₁	3 7 4 X ₁	4 2 1 X ₁
1 2 4 X ₂	5 2 4 X ₂	3 7 4 X ₂	4 2 1 X ₂
5 1 6 X ₁	2 8 9 X ₁	7 5 3 X ₁	3 4 5 X ₁
5 1 6 X ₂	2 8 9 X ₂	7 5 3 X ₂	3 4 5 X ₂

Figure 5.6: Example of Set-Packing

3. Each subset is replicated $k - 1$ times and thus, we have k identical itineraries. For the i th itinerary, a virtual POI x_i is appended, denoting that the itinerary is designed for the i th day.
4. Apply the algorithm of set-packing to get the optimal solution. Let S_r be the result set. If $|S_r| > k$, there must be two itineraries for the same day and they are not disjoint. If $|S_r| < k$, we still have available days for traveling and new itineraries can be added. Therefore, $|S_r| = k$ and S_r can be considered as a k -day itinerary.

□

In step 3 of our proof, we replicate the itinerary $k - 1$ times. That is to guarantee that the solution of set-packing problem returns exactly k subsets. Figure 5.6 illustrates the idea. Suppose we have four index files and want to generate a 2-day itinerary. Without the replication, the set-packing algorithm may return a 3-day itinerary, such as “5|1|6”, “2|8|9” and “3|7|4”. By replicating the itineraries and adding the virtual elements X_1 and X_2 , the above selection cannot work, as two itineraries will share at least one virtual element. In this case, the set-packing algorithm will return another solution (e.g., “1|2|4|X₁” and “7|5|3|X₂”), which satisfies our time requirement.

Although set-packing is also an NP-complete problem, different from the TOP, in a special case, set-packing problem has approximate algorithms. As mentioned

in the pre-processing, we set the maximal number of MapReduce jobs in generating the single-day itineraries to m . Therefore, each itinerary can have at most m POIs. It was shown that when the size of subsets is bounded by a constant, the weighted set-packing problem can be solved by polynomial approximations [60; 5]. By following the above ideas, in this chapter, we design a variant of the approximate algorithm in [60], which provides a bound of $\frac{2(m+1)}{3}$ for the quality of the approximate answers. The algorithm includes an initialization phase and an adjustment phase.

5.4.1 Initialization

For the user request (S_p, k) , we adjust the weights of POIs in S_p to emphasize the user's selection. If $v_i \in S_p$, v_i 's weight is increased to $\alpha(w(v_i) + 1)$, where α is an integer larger than 0 and $w(v_i)$ is the original weight of POI v_i . Algorithm 5.5 shows how we generate the seed itineraries using the greedy strategy.

Theorem 5.4.2. *Given a list of POIs $L = \{v_0, v_1, \dots, v_n\}$ that can be accessed within one day, by scanning the index of v_i in L , we can get the itineraries that contain all POIs in L and the first candidate is the itinerary with maximal weight.*

Proof. Because L can be finished within one day, there must be some itineraries containing all the POIs in L . Let I_0 and I_1 be first and second candidate itineraries respectively. I_0 's weight is larger than I_1 's, since before weight adjustment (i.e. if v_i is in the user's selected POI set S_p , v_i 's weight is increased to $\alpha(w(v_i) + 1)$.) I_0 has a higher weight than I_1 according to the sorting order of the inverted index. Then after weight adjustment, both of them receive the same weight boost. To sum up, the first candidate is the itinerary with maximal weight. \square

To improve the weights of the obtained itineraries in the greedy algorithm, we adopt the adjustment phase.

Algorithm 5.5: Initialization(POIList L , Day k)

```
1: sortByWeight( $L$ )
2: int  $i=0$ , Set  $seed = \emptyset$ , Set  $rev = \emptyset$ 
3: while  $i < k$  and  $L.size() > 0$  do
4:   int  $poi = L.nextPOI()$ ;
5:   Set  $group = new Set()$ 
6:    $group.add(poi)$ 
7:   int  $lastpoi = poi$ 
8:   while not  $L.isEmpty()$  do
9:     int  $newpoi = getNearest(lastpoi, L)$ 
10:    int  $time = getTravelTime(group, newpoi)$ 
11:    if  $time \leq$  one day then
12:       $group.add(newpoi)$ 
13:       $L.remove(newpoi)$ 
14:       $lastpoi=newpoi$ 
15:    else
16:      break;
17:    end if
18:  end while
19:   $i++$ ,  $seed.add(group)$ 
20: end while
21: for  $i=0$  to  $seed.size()$  do
22:   Set  $group = seed.get(i)$ 
23:   IndexIterator  $iter = indexManager.scan(group.get(0))$ 
24:   while  $iter.hasMoreElements()$  do
25:     Itinerary  $I = iter.next()$ 
26:     if  $I.contains(group)$  then
27:       removeReplicatedPOI( $I, rev$ )
28:        $rev.add(I)$ 
29:     break
30:   end if
31: end while
32: end for
33: return  $rev$ 
```

5.4.2 Adjustment

In the adjustment phase, new solutions are searched and used to replace the greedy itineraries. The process repeats until no improvement can be obtained. In the following discussion, we discard the virtual POIs to simplify our representations.

Suppose $idx(v_j)$ returns the itineraries in the index of POI v_j . We define the neighborhood of an itinerary as

Definition 5.4.2. Neighborhood

Given an itinerary L_i , its neighborhood $ngb(L_i)$ is an itinerary set satisfying:

$$ngb(L_i) = \bigcup_{v_j \in L_i} idx(v_j)$$

For example, in Figure 5.6,

$$ngb(1|2|4) = \{5|1|6, 5|2|4, 2|8|9, 4|2|1, 3|4|5\} \quad (5.2)$$

The neighborhood of L_i represents the candidate itineraries that can replace L_i . However, some itineraries share the common POIs, which cannot coexist in the result. Therefore, we define the independent set as:

Definition 5.4.3. Independent Set

An independent set $IS(L_i)$ is a subset of $ngb(L_i)$. Any two itineraries in $IS(L_i)$ do not share a common POI. Namely, $\forall L_0, L_1 \in IS(L_i) \rightarrow (L_0 \text{ and } L_1 \text{ are disjoint})$.

Neighborhood of each itinerary can have multiple independent sets and each set denotes a different adjustment strategy. Let \mathcal{S} be the initial itinerary set returned by Algorithm 5.5. An alternative solution \mathcal{S}' can be constructed from \mathcal{S} by replacing the itineraries by their independent sets. More formally,

$$\mathcal{S}' = \mathcal{S} - f(\mathcal{S}, ngb(L_i)) + IS(L_i)$$

where $f(S_a, S_b)$ returns a subset of S_a , which shares at least one POI with itineraries in S_b .

For itinerary “1|2|4” in Figure 5.6, its independent set is $\{2|8|9, 3|4|5\}$. If $\mathcal{S} = \{1|2|4, 7|5|3\}$, after the adjustment, we will get $\mathcal{S}' = \{2|8|9, 3|4|5\}$. All itineraries are replaced by new ones. To avoid the case of cascading replacement,

the size of $IS(L_i)$ should be less than k , as only k single day itineraries are required. In our implementation, we limit the size of $IS(L_i)$ to $\frac{k}{2}$. Namely, at most half of the itineraries are replaced.

The benefit of itinerary adjustment is computed as

$$B = \text{weight}(\mathcal{S}') - \text{weight}(\mathcal{S})$$

If $B > 0$, we assume that the adjustment improves the quality of the results. Hence, a better itinerary can be produced by replacing the old itineraries with corresponding independent sets.

Algorithm 5.6: Adjustment(Set \mathcal{S} , double P , int $step$)

```

1: int  $j = 0$ ;
2: while  $j < step$  do
3:   Set  $cand = \emptyset$ , int  $max = -\infty$ , int  $idx = -1$ 
4:   for  $i = 0$  to  $\mathcal{S}.size()$  do
5:     Set  $ngb = \mathcal{S}.get(i).getNeighborhood()$ 
6:     Set  $ind =$ 
        $getIndependentSetWithMaximalWeight(ngb)$ 
7:     Set  $\mathcal{S}' = \mathcal{S} - f(\mathcal{S}, ngb) + ind$ 
8:     double  $B = \text{weight}(\mathcal{S}') - \text{weight}(\mathcal{S})$ 
9:      $cand.add(\mathcal{S}')$ 
10:    if  $B > max$  then
11:       $max = B$ ,  $idx = i$ 
12:    end if
13:  end for
14:  if  $max > 0$  then
15:     $\mathcal{S} = cand.get(idx)$ 
16:  else
17:    if  $\text{randProb}() > P$  then
18:       $\mathcal{S} = cand.get(idx)$ 
19:    end if
20:  end if
21:   $j++$ 
22: end while

```

Algorithm 5.6 summarizes the idea of adjustment process. We set a threshold

for the maximal number of adjustments. In each iteration, we find the independent sets for existing itineraries. If one itinerary has multiple independent sets, we will select the one with maximal weight (line 6). The new results are then computed by performing the replacement (line 7) and we record the benefit (line 8). After all possible replacement strategies have been checked, we will select the one with maximal benefit. If the benefit is larger than 0, the result itineraries are updated as the new ones (line 13). Otherwise, we will perform the updates, only with a small probability (line 15-16). The idea is to simulate the hill-climbing algorithm to avoid the sub-optimal solution. The algorithm guarantees the quality of the returned itinerary as shown in the below theorem.

Theorem 5.4.3. *Algorithm 5.6 returns a k -day itinerary, which approximate the optimal solution with the bound $\rho = \frac{2(m+1)}{3}$.*

Proof. We first prove that Algorithm 5.6 can return a k -day itinerary, then we prove that we have a bound $\rho = \frac{2(m+1)}{3}$ for the initialization-adjustment algorithm (m is the maximum number of POIs in the itineraries).

(I) *Algorithm 5.6 can return a k -day itinerary.* In Algorithm 5.6, we add a virtual POI to each itinerary to mark its traveling day. Therefore, the adjustment algorithm at most returns k disjoint itineraries. Otherwise, there are two itineraries sharing the same virtual POI. Namely, they are supposed to be traveled in the same day, which is not possible. If the algorithm returns less than k itineraries, we can still repeat the initialization and adjustment to fill in the left days. In this way, we guarantee that Algorithm 5.6 returns exactly a k -day itinerary.

(II) *The result of Algorithm 5.6 approximate the optimal solution with the bound $\rho = \frac{2(m+1)}{3}$.* Based on Theorem 5.4.1, the problem of selecting the k -day itinerary can be reduced to the weighted set-packing problem. Therefore, in Algorithm 5.6, we simulate the heuristic set-packing algorithm. The heuristic algorithm has been analyzed in [60]. Suppose there are X iterations in the algorithm. Let

I_i be the results of the $X - i - 1$ iteration. I_1 will be the final result. Let d_i be the payoff factor of each iteration. We have

$$(m + 1)w(I_1) \geq (2 - \frac{1}{d_1} + \frac{1}{2d_1^2})w(opt)$$

where $w(I_1)$ and $w(opt)$ represent the weights of the itinerary returned by the heuristic algorithm and the optimal itinerary respectively. The right side of the equation is minimized when $d_1 = 1$. In that case, we have

$$(m + 1)w(I_1) \geq (1 + \frac{1}{2})w(opt)$$

Therefore, we have a bound $\rho = \frac{2(m+1)}{3}$ for the heuristic approach, where m is the maximal number POIs in the itinerary (m is also the number of MapReduce jobs in our preprocessing). □

The most expensive operations in Algorithm 5.6 are retrieving the neighborhood sets. We need to scan the indexes of involved POIs to find all itineraries. We find that as Algorithm 5.6 only selects one independent set for each itinerary, we can save I/O costs by scanning a small portion of the index file. Therefore, in our implementation, we read the first n itineraries of an index file in batch and if independent sets are found, the process stops. Otherwise, we will continue to load the next n itineraries.

5.4.3 Hotel selection

In fact, hotels can be considered as a special type of POIs. It must appear as the last POI in the itinerary. We need to calculate the traveling time from other POIs to the hotel POIs. Hotel POIs do not incur access cost and their weights are set as users' rankings for the hotels. Based on the user's preference, we have two processing strategies.

5.4.3.1 Multiple hotels

If the user does not insist on staying in the same hotel (e.g., he can select k different hotels, one for each day), we can extend the preprocessing algorithm to handle the hotels. In the MapReduce jobs, when a new itinerary L_i is generated, we test every hotel POI and try to append it to the end of L_i . Given a hotel POI h_j , we use $L_i|h_j$ to represent the combined itinerary. $L_i|h_j$ is considered as a single day itinerary, if

1. The total traveling time of $L_i|h_j$ is less than H . H is the average traveling time per day.
2. For any other non-hotel POI \bar{v} which is not included by L_i , $L_i|\bar{v}|h_j$ cannot be completed within H time.

When we detect a new single day itinerary, we output it to the DFS for indexing.

Algorithm 5.7: HotelSelection(Set *hotels*, Set *itinerarySet*)

```
1: double max = 0, Set result =  $\emptyset$ 
2: for  $i=0$  to hotels.size() do
3:   Hotel  $h_i=hotels.get(i)$ 
4:   Set copy = itinerarySet
5:   for  $j=0$  to copy.size() do
6:     Set  $L_j = copy.get(j)$ 
7:     while getTravelTime( $L_j, h_i$ ) >  $H$  do
8:        $L_j.removeLast()$ 
9:     end while
10:     $L_j.append(h_i)$ 
11:  end for
12:  double weight=getTotalWeight(copy)
13:  if  $max < weight$  then
14:     $max = weight$ 
15:     $result = copy$ 
16:  end if
17: end for
18: return result
```

The itinerary generation algorithm is exactly the same, except that the hotel POI can appear in different itineraries. In Algorithm 5.6, we do not consider the

hotel POIs, when performing the disjoint test for itineraries. The output itinerary may contain multiple hotels (h_i represents the hotel POI):

$$2|5|10|h_1, 3|7|8|h_1, 9|10|0|h_2$$

5.4.3.2 Single hotel

If the user prefers to staying in the same hotel, the itinerary generation problem cannot be easily reduced to the set-packing problem. Instead, we adopt a best-effort solution. In particular, we still apply Algorithm 5.6 to find the candidate k -day itinerary without hotel POIs. After that, we invoke Algorithm 5.7 to append the hotel POI.

The idea is to discard a few POIs from the end of each itinerary and try to append the hotel POIs to the shortened itinerary. In line 7-8, the itinerary progressively removes the last POI, until it can include the hotel POI to form a single-day itinerary, e.g., the total traveling time is less than H . In line 10, we will get a new set of k itineraries, where all itineraries contain the same hotel POI. We will generate such a k -day itinerary for each hotel. After comparing weights of the itineraries, the one with maximal weight is returned as our final k -day itinerary.

5.5 Experiments

5.5.1 Dataset description

To evaluate the performance of our proposed approaches, we crawl the traveling information from Yahoo Travel (<http://travel.yahoo.com>). In particular, we focus on the Singapore POIs. Figure 5.7 illustrates our crawling strategy. Yahoo classifies the POIs into *hotels*, *things to do* and *cities*. We use the first two types in our experiments, as the last one is the geo-locations for the city. *things to do* contains 254 POIs of Singapore and *hotels* contains 276 hotels from unranked to five stars.

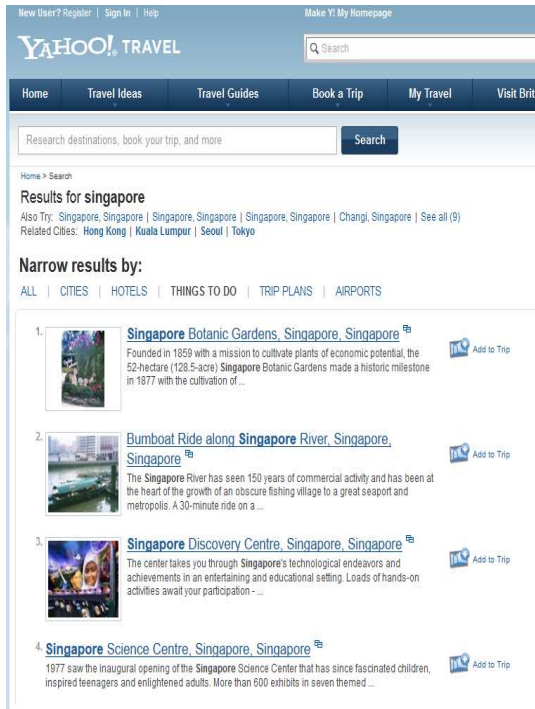


Figure 5.7: Yahoo POIs

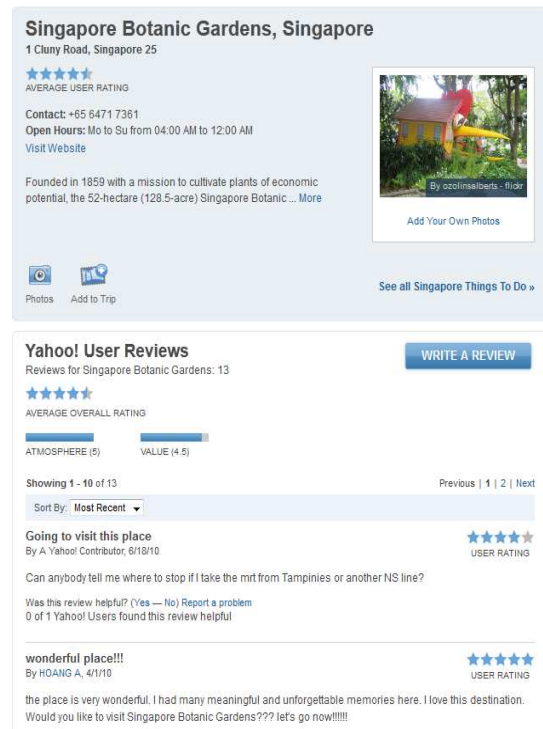


Figure 5.8: User Reviews

After removing the duplicated and meaningless POIs, we keep 400 POIs for our experiments. As far as we know, this is the largest dataset for the automatic itinerary recommendation. In [11], the largest dataset only contains 163 POIs.

The POI's weight is also crawled from Yahoo Travel. As shown in Figure 5.8, for each POI, Yahoo maintains a page for users' reviews. We accumulate the user scores for each POI as its weight. If a POI has not been reviewed, we assign it an initial weight (e.g., 1).

The average visiting time of a POI is estimated from the shared travel plans in Yahoo Travel. The edge cost between any two POIs are estimated using Google Map. Specifically, the public transit time for the shortest path between two POIs is used as the edge cost. We assume that each user will spend at most 8 hours for traveling per day.

In our experiments, the query is (S_p, k) , where S_p is randomly selected from the non-hotel POIs. We allow the users to select the same hotel POIs for different

days. The traveling time is set to 3 days by default. For comparison, we implement the original TOP algorithm proposed in [25].

Table 5.1 lists the parameters used in our experiments. The experiments are conducted on our in-house cluster, Awan (<http://awan.ddns.comp.nus.edu.sg/ganglia/>). We use 64 nodes exclusively. Each node has one Intel X3430 2.4GHz processor, 8GB of memory, two 500GB SATA hard disks and gigabit ethernet. Hadoop ² is used as our MapReduce engine.

Table 5.1: Experiment Settings for Chapter 5

Settings	
Parameter	Range and Default Value
k	3 (1-5)
α	2
buffer size	5 million itineraries
size of S_p	10 (5-20)
total POIs	400 (100-400)
number of MapReduce nodes	32 (8-64)

5.5.2 Single-day itinerary generation

In the preprocessing, m MapReduce jobs are submitted sequentially to iterate all possible single-day itineraries. The input are our crawled POIs and the output contain all single-day itineraries. This is, in fact, a brute-force search strategy, but we exploit the parallel processing engine to reduce its cost. After the single-day itineraries are generated, we start another MapReduce job to remove the duplicate itineraries. We call it the *Dup-Clean* job (the previous m jobs are named *MR-Scan*). *Dup-Clean* generates a special namespace for each itinerary by combining its POIs. The namespace is used as the key in the shuffling phase. All duplicated itineraries will be shuffled to the same *reducer*, where a local clean process is conducted.

Figure 5.9 shows the accumulate costs of all m jobs and the cost of the clean

²<http://hadoop.apache.org/>

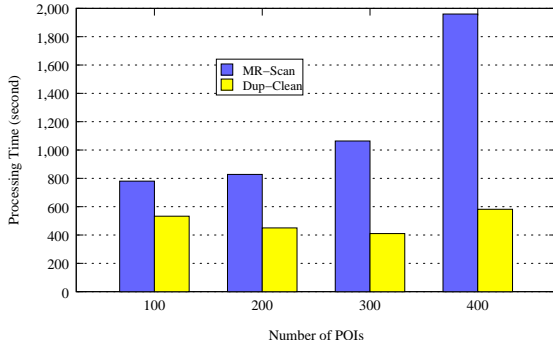


Figure 5.9: Preprocessing Cost

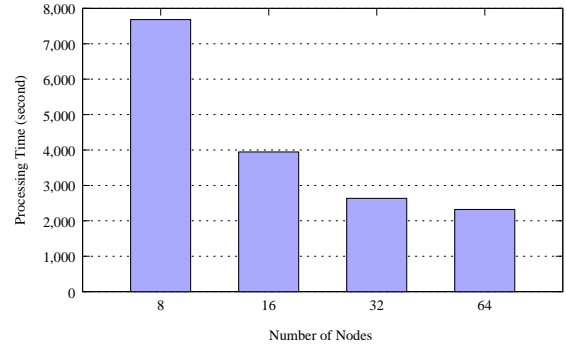


Figure 5.10: Scalability of Preprocessing

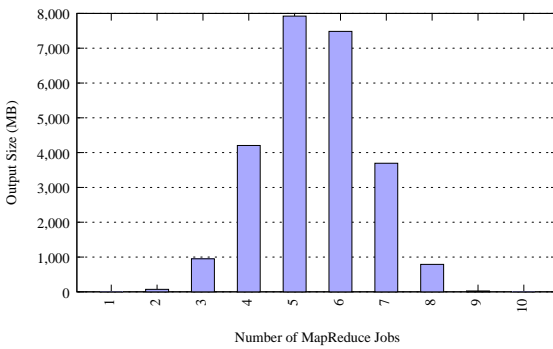


Figure 5.11: Size of Single Day Itinerary

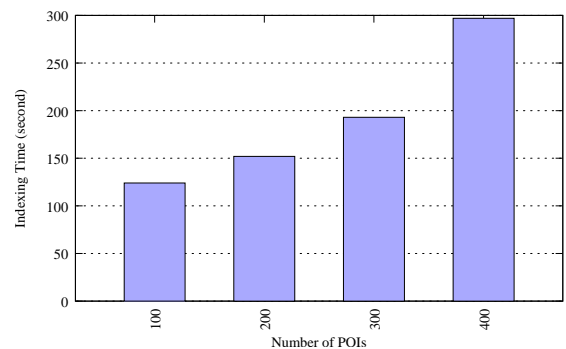


Figure 5.12: Indexing Cost

job. We vary the number of POIs in the preprocessing from 100 to 400. The cost of the *MR-Scan* increases for a larger POI set. However, even for 400 POIs, the preprocessing only takes less than 1 hour. This is an offline process and only needs to be invoked once. In fact, most travel agencies do not maintain such a large number (400) of POIs for a single city. Interestingly, the performance *Dup-Clean* is not correlated to the POI number. Its cost is neutralized by the parallel processing strategy. We observe that most nodes are not fully exploited in *Dup-Clean*. Figure 5.10 shows the scalability of the MapReduce jobs (*MR-Scan+Dup-Clean*). We vary the number of nodes in our cluster from 8 to 64 and we observe a near-linear improvement over the performance. Therefore, to handle a larger POI-graph, we can simply add more processing nodes into our cluster.

In the preprocessing, the maximal number of MapReduce jobs (m) is set to 10. Namely, each single-day itinerary can contain at most 10 POIs. m is a configurable parameter. As shown in Figure 5.11, in our dataset, most itineraries consist of 4-7

POIs. Setting m to 10 can iterate most itineraries in our case.

5.5.3 Itinerary indexing

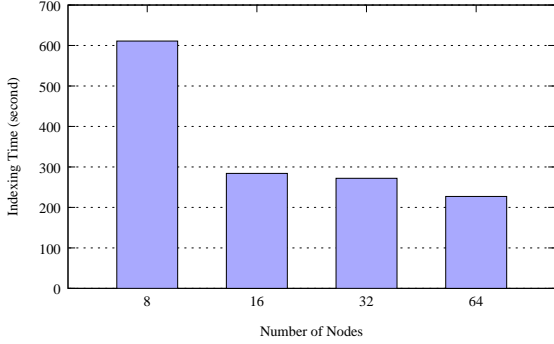


Figure 5.13: Scalability of Indexing

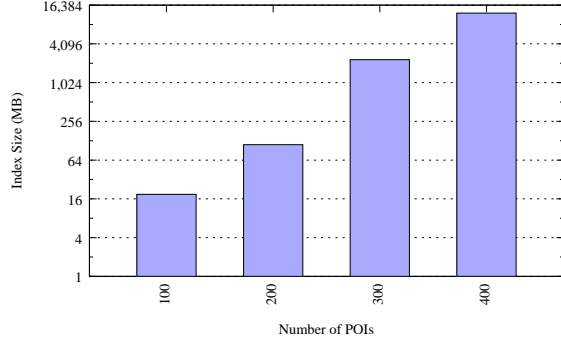


Figure 5.14: Size of Index

The second step of preprocessing is to build the itinerary index. The index process only requires one MapReduce job and is much faster than the itinerary iteration process. In Figure 5.12, we show the indexing cost for different sizes of POI graphs. We can efficiently recreate the index within a few minutes. Figure 5.13 conducts the scalability test for the indexing process. The indexing process benefits from a larger cluster. Figure 5.14 shows the total index size for different POI graphs. The size of index increases exponentially with the size of POI graph. But even for the graph with 400 POIs (a large enough POI graph for most cities), only 12GB index data are generated. The index is maintained in the DFS and hence, the storage cost is not the system bottleneck.

5.5.4 Effect of POI Graph size

In the experiments, we compare our approach (*MR-Set*) with the original *TOP* approach in [25]. To evaluate the query performance, two metrics, processing time and weight ratio are adopted. The weight ratio is used to measure the quality of the generated itineraries. In particular, let W_i and W_j denote the total weights of *MR-Set* and *TOP* respectively. The weight ratio is defined as $\frac{W_i}{W_j}$. In idea case,

we should compare the approximate results with the optimal ones. However, it is impossible to generate the optimal results, given the size of POI graph and complexity of the problem (the TOP is NP-complete problem).

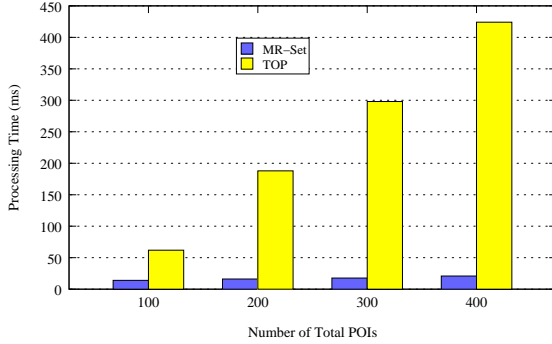


Figure 5.15: Effect of Graph Size (Processing Time)

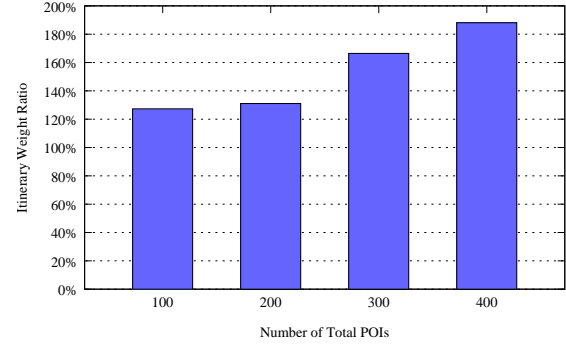


Figure 5.16: Effect of Graph Size (Quality)

We first vary the graph size to test the query performance. Figure 5.15 and 5.16 show the processing time and weight ratio respectively. Our new approach significantly reduces the processing cost, as we have already computed the single-day itineraries in the preprocessing. Previous *TOP* approach is not scalable. The query cost increases linearly with the number of POIs. If more POIs (e.g. restaurants) are added in the set, the *TOP* approach may fail to provide a satisfied performance. On the contrary, our technique enables the itinerary to be generated within 30 milliseconds. It is not affected by the POI graph size. Moreover, the travelling plan system is accessed by multiple users concurrently. In the case of 400 POIs, *TOP* approach can serve up to 2 requests per second, while our approach can provide a throughput of 40 requests per second. Our approach is more scalable and feasible for the real-time processing.

In fact, our approach not only reduces the processing overhead, it also provides results with higher qualities. Figure 5.16 shows the change of weight ratio. We have 20% to 80% improvement over the original TOP algorithm. The gap increases for a larger POI graph, as our approach can efficiently exploit the POI combinations. More POIs indicate a higher possibility of finding a good itinerary.

5.5.5 Effect of selected POIs

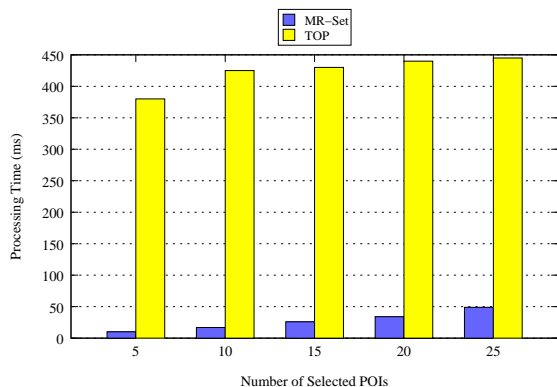


Figure 5.17: Effect of Selected POIs (Processing Time)

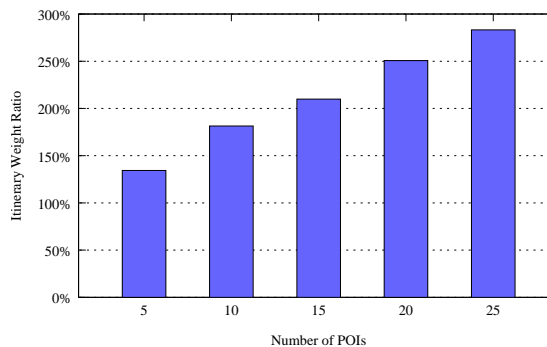


Figure 5.18: Effect of Selected POIs (Quality)

In our query model, we allow the user to explicitly select some POIs as their preferences. The weights of the selected POIs are adjusted to reflect the selection. This strategy may increase the importance of some unpopular POIs and avoids generating the itinerary with the same set of top-popular POIs. This is how the users customize their itineraries in our system. Figure 5.17 and Figure 5.18 show the effect of varied number of selected POIs (from 5 to 25). The default traveling time is set to 3 days. In fact, most people will not select too many POIs (e.g., 25) for a 3-day itinerary.

In Figure 5.17, the cost of *MR-Set* increases for a larger number of selected POIs. This is because in the adjustment phase, *MR-Set* needs to look up the index of the corresponding POIs to search for the replacements. Index is maintained by the DFS and the I/O costs dominate the query cost. However, *MR-Set* is still much more efficient than the original TOP.

Figure 5.18 reveals that the quality gap between *MR-Set* and the TOP approach enlarges, when the user selects more POIs as his preference. *MR-Set* can effectively find the itinerary that includes as many selected POIs as possible. It can optimize the way of how to combine the selected POIs and other POIs into the itinerary.

5.5.6 Effect of traveling budget

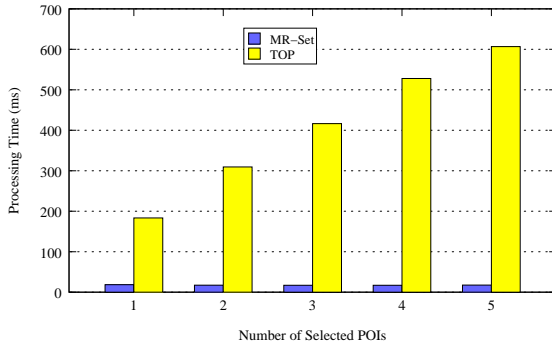


Figure 5.19: Effect of Traveling Time (Processing Time)

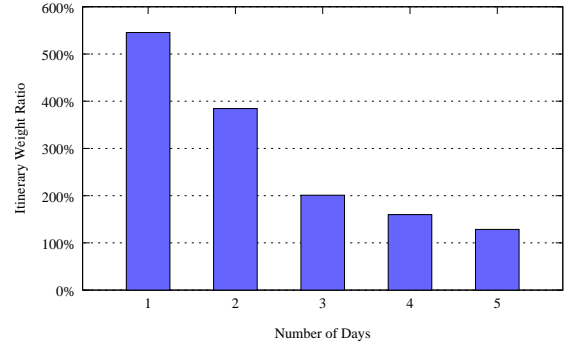


Figure 5.20: Effect of Traveling Time (Quality)

Besides the POIs, the user can change his expected traveling time as well. With more time budget, his itinerary can include more interested POIs. Figure 5.19 and Figure 5.20 show the effect of varied time budgets. The original TOP algorithm incurs a higher overhead for the increased time budget, because it needs to generate and refine each single-day itinerary progressively. *MR-Set* adopts a different strategy. When it tries to adjust the itinerary, it may replace multiple single-day itineraries with new ones. It considers the k -day itinerary as a whole solution, instead of treating each single-day itinerary independently. It is interesting to observe that Figure 5.20 shows a different result from Figure 5.18. The weight ratio decreases, when more traveling budget is given. In fact, the TOP algorithm benefits from a loose time budget, as it can arrange more high-weight POIs into different single-day itineraries,

5.5.7 Effect of adjustment

The query processing of *MR-Set* splits into the initialization phase and adjustment phase. In this experiments, we show the effect of the adjustment phase. We vary the number of selected POI from 1 to 15.

Figure 5.21 shows that the adjustment phase greatly increases the processing cost. Algorithm 5.6 may repeat for several iterations before converging to a high-

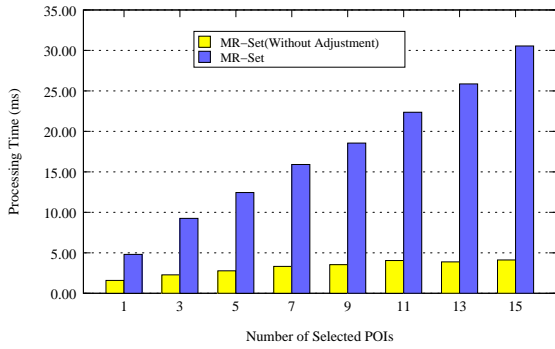


Figure 5.21: Effect of Adjustment (Processing Time)

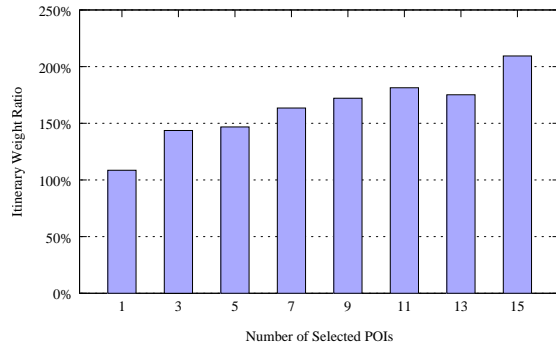


Figure 5.22: Effect of Adjustment (Quality)

quality result. As mentioned before, in the adjustment phase, the query engine loads the itinerary index from the DFS, which incurs high I/O cost. One way to reduce the cost is to increase the index buffer size. After an indexed itinerary is loaded from the DFS, we cache it in the buffer. If the buffer is full, we apply the LRU strategy to remove the less used entries.

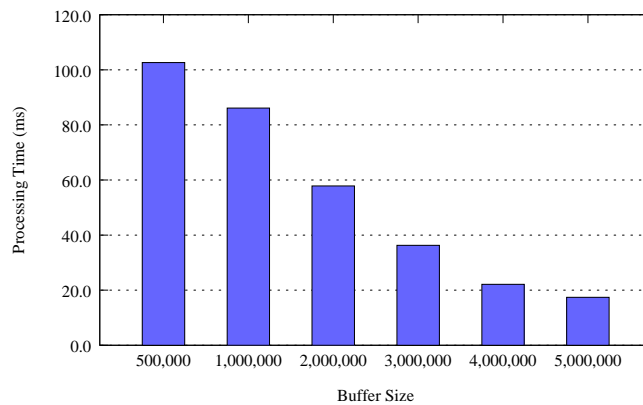


Figure 5.23: Buffer Size

In Figure 5.23, we change the number of buffered single-day itineraries in the index buffer and test the query performance. Not surprisingly, we can get a huge performance boost by deploying a large enough index buffer. In fact, the single-day itinerary is less than 64 bytes and caching 5 million entries only takes about 300M memory. Any modern server can effectively reduce the processing cost by employing a large buffer.

Although the adjustment phase incurs high processing cost, it can significant-

ly improve the result quality. As shown in Figure 5.22, the adjustment phase can double the weight of generated itinerary if more than 15 POIs are selected³. With more POIs selected, the adjustment phase can generate more replacement itineraries and therefore, has a better chance of finding the high-quality result.

5.5.8 Effect of single hotel selection

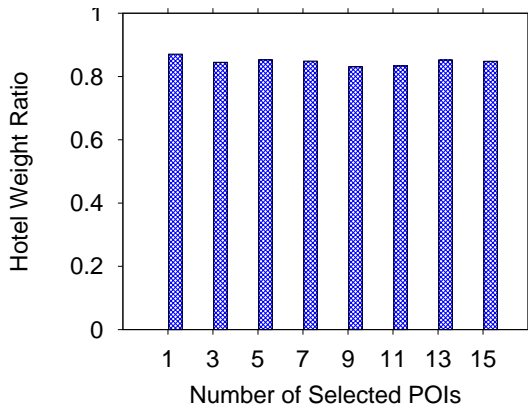


Figure 5.24: Effectiveness of Single Hotel Selection

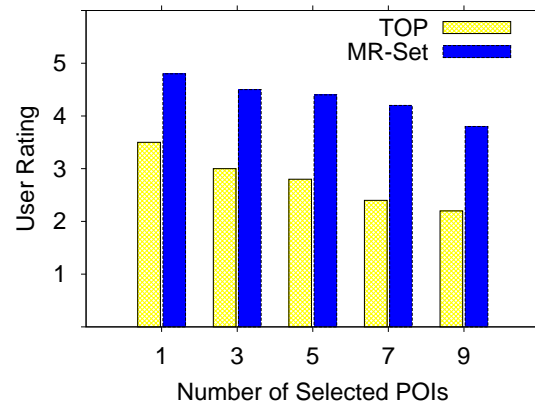


Figure 5.25: User study

In this section, we justify the effectiveness of hotel selection algorithm. In Algorithm 7, we adopt a “best-effort” solution to append the hotel to the end of each itinerary. To evaluate the performance of such solution, we define a new metric, the hotel weight ratio. In particular, let W_m and W_s denote the total weights of generated itineraries in the multiple hotel case and single hotel case, respectively. The hotel weight ratio is defined as $\frac{W_s}{W_m}$. Our “best-effort” solution still provides high quality results. Figure 5.24 shows the change of hotel weight ratio. We can see that, in the single hotel case, the total weight of generated itineraries is penalized as each single day itinerary should end in the same hotel POI. However, the “best-effort” solution can provide an approximate result with 85%-90% of the total weight as in the multiple hotel case. This indicates that

³In this figure, the weight ratio is computed between the *MR-Set* with adjustment and *MR-Set* without adjustment

Algorithm 7 is still able to find good itineraries with the single hotel constraint.

5.5.9 User study

To evaluate the quality of the generated itineraries, we conduct a user study, which asks the users to manually rank the itineraries. Our study hires 20 undergraduate students as the users. Given a set of selected POIs, we use *TOP* and *MR-Set* methods to generate 20 groups of itineraries (3-day itineraries in the experiment). Each participant assigns a score (ranging from 1 to 5) to each itinerary in his group. The average ranks are then computed for the itineraries generated by different approaches. Figure 5.25 shows the results. Most users prefer the results generated by *MR-Set*. We also observe that the ratings of both *TOP* and *MR-Set* are reduced, when more POIs are selected as the necessary POIs. It is because that some of user selected POIs are missing in the itineraries due to the constraint of travel time.

5.6 Conclusions

In this chapter, we present a dynamic itinerary recommendation system for the backpacking travelers based on our semi-lazy learning approach. While the eager learning recommendation method cannot return personalized itineraries to satisfy different users' requirement, it is better to provide a service which dynamically recommends a customized multi-day itinerary tailored to the user's preference. However, such dynamic itinerary recommendation problem is a famous NP-complete problem, TOP (Team Orienting Problem), which has no polynomial-time approximate algorithm.

To provide for a near-optimal solution for itinerary recommendation, the semi-lazy learning approach is adopted. First, we iterate all candidate single-day itineraries using the MapReduce framework and index them as the reference itinerary database. Then, after selecting top- k best candidate itineraries from

the inverted index, the system returns the recommendation result which is further adjusted by the initialization-adjustment model.

From a theoretical perspective, our semi-lazy learning approach indeed transform the Team Orienting Problem into the weighted set-packing problem, which has efficient approximate algorithms. The initialization-adjustment model is indeed an approximate algorithm for the set-packing problem, which is at most $\frac{2(m+1)}{3}$ worse than the optimal result. Experiments on real dataset from Yahoo's traveling website show that our proposed approach can efficiently recommend high quality customized itineraries.

Chapter 6

Conclusions

6.1 Summary

The purpose of this thesis is to overcome certain difficulties in dynamic spatio-temporal data analysis. Our principle intuition is that the historical spatio-temporal data itself captures much more information than the models or patterns extracted using the the eager learning approach. In this regard, our breakthrough solution is that, for each analysis request, we first retrieve related items from historical data, followed by building an analysis model upon the search result on the fly. According to this idea, the present study aimed to implement the semi-lazy approach into real-life applications to assess its effectiveness and efficiency. After conducting an in-depth study, we have given an account of the reasons for the utility of the semi-lazy approach. In particular, this thesis contributes by applying the semi-lazy learning approach to three practical real-life applications, which are trajectory prediction, time series prediction and itinerary recommendation. Summaries of these works are listed below.

- **Trajectory Prediction.** We explored the usability of the semi-lazy approach for trajectory prediction in Chapter 3. The study has revealed the effectiveness of the semi-lazy learning trajectory prediction method, especial-

ly in dynamic environments. Unlike previous approaches, which adopt the eager learning approach to construct complex models [109][70] or mine numerous patterns [86][83], we propose to leverage on the growth of computing power by building a prediction model on the fly. More specifically, the idea of the semi-lazy learning approach is injected into the proposed trajectory prediction model, which utilizes dynamically selected historical trajectories. We also implemented a demonstration prototype to show the key aspects of our system. The experiment shows that our method can outperform competitors in terms of prediction rate and prediction distance error, by 2 to 5-fold. A possible explanation for the improvement of our method is that the target trajectories to be predicted are known before the models are built, which allows us to construct models that are deemed relevant to the target trajectories. The results in this study indicate that the semi-lazy learning approach is sound, and promising for prediction analysis in dynamic environments. This result is of considerable importance, since this study may pave the way to a wide range of applications related to trajectory prediction in dynamic environments such as event prediction and outlier detection.

- **Time Series Prediction.** We assessed the performance of the semi-lazy learning approach to time series prediction in Chapter 4. An automatic time series prediction system for sensors was developed under the semi-lazy learning approach, which is significantly different from the classical time series prediction models such as statistical regression models (e.g. ARIMA [20] and GRACH [16]) and eager learning models (e.g. SVMs [87; 126; 99] and GPs [57; 90; 21; 59; 125]). Two demanding problems in the system are tackled: fast k-nearest neighbor (kNN) search under Dynamic Time Warping (DTW) distance and applicable model selection for semi-lazy learning time series prediction. To attack the former problem, a GPU-based index and a search method were designed to accelerate the DTW kNN search from time series data. For the latter problem, we contrived an extensive study for model

selection of the semi-lazy time series prediction. Extensive experiments on several real-world datasets demonstrate that our system does predict the future trend of sensors properly in real time.

- **Itinerary Recommendation.** We also investigated the effect of the semi-lazy learning approach for itinerary recommendation in Chapter 5. The result of this investigation shows that the semi-lazy approach can recommend customized multi-day itineraries based on the individual users' preferences. To our best knowledge, most of the existing methods on itinerary recommendation utilize an eager learning scheme [97; 39; 35; 138]. They first adopt the eager learning models to discover users traveling patterns. Next, these methods recommend prevalent itineraries to users, based on the discovered patterns. However, this lacks customization, so this scheme cannot satisfy individual dynamic requirements. In contrast, our semi-lazy method can help the traveling agency provide a customized recommendation service. In this way, our method recommends personalized itineraries for each user instead of adopting the most popular ones. Experiments on a real data set from Yahoo's traveling website illustrates that our approach can efficiently recommend high quality customized itineraries. The results of this study suggest that the semi-lazy learning approach can produce more practical solutions than the eager learning approach, since the individual users' dynamic requirements are taken into account.

Taken together, the above three works suggest that the semi-lazy learning approach is a practical and promising method for dynamic spatio-temporal data analysis. The semi-lazy approach may take a major step towards solving the difficulties of dynamic spatio-temporal data analysis.

Moreover, the semi-lazy learning approach may open a door for other data analysis tasks, instead of only spatio-temporal data analysis. We understand that all the learning approaches (i.e. lazy learning, eager learning or semi-lazy learning)

are not only applicable for spatio-temporal data, but many other data analysis tasks as well. For example, by combining with other data mining techniques, the semi-lazy learning approach can be extended to support data streaming mining and video surveillance analysis. Yet, these are not central to this study and hence are beyond the scope of this thesis.

6.2 Future work

The semi-lazy learning approach offers a new paradigm for predictive analysis on spatio-temporal data. In addition to problems mentioned in the previous section, there are some potential avenues for future work involving the theoretical study and generalizations in the semi-lazy learning approach that may be fruitful:

- **Theoretical Study.** Further research might be undertaken to establish the theoretical foundation of the semi-lazy learning approach. From the theoretical perspective, this approach has thrown up many questions in need of further investigation. For example, the lazy learning approach has been proved to be very stable [19]. However, many important eager learning algorithms are unstable [65] such as decision-tree and neural network. Since the semi-lazy approach is a combination of the lazy learning approach and the eager learning approach, it will be appealing to study the stability of the semi-lazy learning approach. Further research could also attempt to investigate the theoretical properties of the semi-lazy approach from several points, including the Vapnik-Chervonenkis (VC) theory, empirical error and sensitivity analysis.
- **Efficient Similarity Search.** One crucial part of the semi-lazy learning approach is to retrieve similar neighbors from the whole dataset. This problem becomes severe if the data is essentially in high dimensional space. One feasible solution is to undertake an approximate nearest neighbor (ANN)

search method, like Locality Sensitive Hashing [56], to facilitate the similarity search process. A further solution is to integrate the Locality Sensitive Hashing method with the modern Massive Parallel Processing (MPP) architecture, which is especially intriguing and promising in the era of big data.

- **Dedicated Model Selection.** It is desirable to design a dedicated model selection process for the semi-lazy learning approach, where a prediction query is known before the model is derived. In this regard, there is some priori information that can be integrated into the training process to improve the model. Hence, it is better to develop a specialized training process which is biased (or “over-fitted”) for the prediction query. Several problems are worthy of further investigation such as how to extend the idea to Maximum Likelihood Estimation (MLE). It is also fascinating to integrate other mature machine learning techniques, such as online gradient descent [18] and low-rank approximation [81], into the semi-lazy learning approach.

Our work on the practical spatio-temporal data analysis problems also has some limitations that might be interesting to study in further extensions. Reiterating the limitations, the main points for extensions are:

- For trajectory prediction, the most important limitation lies in the fact that our prediction method has a longer response time than the existing methods. Hence, more work should be done to invent a more novel index structure and model inference method to speed up our method.
- For time series prediction, one limitation of the system is that, for a batch of prediction requests, the index of the historical time series of all sensors has to be buffered in the global memory of the GPU. Since the largest memory of the GPU is only 6GB, this requirement limits the number of sensors to be predicted within one batch. However, with the rapid advancement of the GPU technology, we think a GPU with a larger memory will be feasible

soon. The other limitation is that the training process of the Gaussian Process prediction model is still highly expensive. It is possible to accelerate the GP training process by utilizing the powerful GPU parallel computation capability. However, this is out of the scope of this thesis and is worthy of a future study.

- For itinerary recommendation, a limitation of this study is that the method requires a huge amount of storage space to store the candidate itineraries, therefore, we resorted to using the Hadoop platform to solve this problem. Further research may be undertaken to design a compression algorithm to reduce the huge itinerary storage requirement.

Bibliography

- [1] R. Adhikari and R. Agrawal. A novel weighted ensemble technique for time series forecasting. In *PAKDD*, pages 38–49, 2012.
- [2] T. Alabi, J. D. Blanchard, B. Gordon, and R. Steinbach. Fast k-selection algorithms for graphics processing units. *Journal of Experimental Algorithms (JEA)*, 17:4–2, 2012.
- [3] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [4] C. Archetti, A. Hertz, and M. G. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13:49–76, February 2007.
- [5] E. M. Arkin and R. Hassin. On local search for weighted k-set packing. *Math. Oper. Res.*, 23:640–648, March 1998.
- [6] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.
- [7] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: efficient time series search and retrieval. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, pages 252–263, 2008.
- [8] K. Bache and M. Lichman. UCI machine learning repository. <http://archive.ics.uci.edu/ml>, 2013.
- [9] T. Ban, R. Zhang, S. Pang, A. Sarrafzadeh, and D. Inoue. Referential knn regression for financial time series forecasting. In *Neural Information Processing*, pages 601–608. Springer, 2013.
- [10] R. Barillec, B. Ingram, D. Cornford, and L. Csató. Projected sequential gaussian processes: A c++ tool for interpolation of large datasets with heterogeneous noise. *Computers & Geosciences*, 37(3):295–309, 2011.
- [11] S. Basu Roy, G. Das, S. Amer-Yahia, and C. Yu. Interactive itinerary planning. In *ICDE*, pages 15–26, 2011.

- [12] J. L. Bentley. Multidimensional binary search trees used for associative searching. *COMMUN. ACM*, 18(9):509–517, 1975.
- [13] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *AAAI94 workshop on knowledge discovery in databases*, pages 359–370, 1994.
- [14] G. Biau, K. Bleakley, L. Györfi, and G. Ottucsák. Non-parametric sequential prediction of time series. *Journal of Nonparametric Statistics*, 22(3):297–317, 2010.
- [15] E. Blanzieri and F. Melgani. Nearest neighbor classification of remote sensing images with the maximal margin principle. *Geoscience and Remote Sensing, IEEE Transactions on*, 46(6):1804–1811, 2008.
- [16] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.
- [17] G. Bontempi, S. B. Taieb, and Y.-A. Le Borgne. Machine learning strategies for time series forecasting. In *Business Intelligence*, pages 62–77. Springer, 2013.
- [18] L. Bottou. On-line learning and stochastic approximations. In *On-line learning in neural networks*, pages 9–42, 1999.
- [19] O. Bousquet and A. Elisseeff. Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526, 2002.
- [20] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, 4th edition, 2008.
- [21] S. Brahim-Belhouari and A. Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, 2004.
- [22] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.
- [23] Y. Bu, L. Chen, A. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In *SIGKDD*, pages 159–168, 2009.
- [24] D. Chakrabarti and C. Faloutsos. F4: large-scale automated forecasting using fractals. In *CIKM*, pages 2–9, 2002.
- [25] I.-M. Chao, B. L. Golden, and E. A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464–474, February 1996.

- [26] G. Chen, S. Wu, J. Zhou, and A. K. Tung. Automatic itinerary planning for traveling services. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):514–527, 2014.
- [27] L. Chen and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.
- [28] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.
- [29] Y. Chen, M. Nascimento, B. Ooi, and A. Tung. Spade: On shape-based pattern detection in streaming time series. In *ICDE*, pages 786–795, 2007.
- [30] R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 16(9):1112–1127, 2004.
- [31] Y.-C. Cheng and S.-T. Li. Fuzzy time series forecasting with a probabilistic smoothing hidden markov model. *Fuzzy Systems, IEEE Transactions on*, 20(2):291–304, 2012.
- [32] M. D. Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *HT*, pages 35–44, 2010.
- [33] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh*, 1976.
- [34] M. Clements, P. Serdyukov, A. P. de Vries, and M. J. Reinders. Using flickr geotags to predict user travel behaviour. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR, 2010.
- [35] M. Clements, P. Serdyukov, A. P. De Vries, and M. J. Reinders. Using flickr geotags to predict user travel behaviour. In *SIGIR*, pages 851–852, 2010.
- [36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to algorithms. In *Second Edition. The MIT Press and McGraw-Hill Book Company*, 2001.
- [37] A. Cotter, N. Srebro, and J. Keshet. A gpu-tailored approach for training kernelized svms. In *KDD*, pages 805–813, 2011.
- [38] T. M. Cover. Estimation by the nearest neighbor rule. *Information Theory, IEEE Transactions on*, 14(1):50–55, 1968.
- [39] D. J. Crandall, L. Backstrom, D. P. Huttenlocher, and J. M. Kleinberg. Mapping the world’s photos. In *WWW*, pages 761–770, 2009.

- [40] L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668, 2002.
- [41] C. Cuda. Programming guide. *NVIDIA Corporation*, 2014.
- [42] M. Cuturi. Fast global alignment kernels. In *ICML*, pages 929–936, 2011.
- [43] M. Cuturi. Pems-sf data set. <https://archive.ics.uci.edu/ml/datasets/PEMS-SF>, 2014.
- [44] DataMarket. Internet traffic data. <http://data.is/19Cbyed>, 2014.
- [45] I. Davidson and K. Yin. Semi-lazy learning: combining clustering and classifiers to build more accurate models. In *MLMTA*, 2003.
- [46] W. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *J. of classification*, 1(1):7–24, 1984.
- [47] J. G. De Gooijer and R. J. Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.
- [48] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53:72–77, Jan. 2010.
- [49] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.
- [50] S. Dunstall, M. E. Horn, P. Kilby, M. Krishnamoorthy, B. Owens, D. Sier, and S. Thiebaut. An automated itinerary planning system for holiday travel. *Information Technology and Tourism*, 6(3), 2004.
- [51] D. Ellis, E. Sommerlade, and I. Reid. Modelling pedestrian trajectory patterns with gaussian processes. In *ICCV Workshops*, 2009.
- [52] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.
- [53] C. Faloutsos and R. Snodgrass. Fast subsequence matching in time-series databases. In *Proceedings of the 1994 ACM SIGMOD International Conference on*, pages 419–429. ACM Press, 1994.
- [54] G. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.
- [55] K. Fujinaga, M. Nakai, H. Shimodaira, and S. Sagayama. Multiple-regression hidden markov model. In *ICASSP*, volume 1, pages 513–516, 2001.

- [56] A. Gionis, P. Indyk, R. Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.
- [57] A. Girard, C. E. Rasmussen, J. Quinero-Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs-application to multiple-step ahead time series forecasting. In *NIPS*, 2002.
- [58] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
- [59] T. Hachino and V. Kadiramanathan. Time series forecasting using multiple gaussian process prior model. In *CIDM*, pages 604–609, 2007.
- [60] M. M. Halldórsson and B. Chandra. Greedy local improvement and weighted set packing approximation. *J. Algorithms*, 39:223–240, May 2001.
- [61] W.-S. Han, J. Lee, Y.-S. Moon, S.-w. Hwang, and H. Yu. A new approach for processing ranked subsequence matching based on ranked union. In *Proceedings of the 2011 international conference on Management of data, SIGMOD '11*, pages 457–468. ACM, 2011.
- [62] I. Hefez, Y. Kanza, and R. Levin. Tarsius: a system for traffic-aware route search under conditions of uncertainty. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS*, pages 517–520, 2011.
- [63] I. Hendrickx and A. Van Den Bosch. Hybrid algorithms with instance-based classification. In *ECML*, pages 158–169. 2005.
- [64] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5–10, 2004.
- [65] D. Hush, C. Scovel, and I. Steinwart. Stability of unstable learning algorithms. *Machine learning*, 67(3):197–206, 2007.
- [66] R. J. Hyndman and Y. Khandakar. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software*, 27(i03).
- [67] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa. Extracting mobility statistics from indexed spatio-temporal datasets. In *STDBM*, 2004.
- [68] H. Jeung, Q. Liu, H. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, pages 70–79, 2008.
- [69] H. Jeung, M. Yiu, X. Zhou, and C. Jensen. Path prediction and predictive range querying in road network databases. *The VLDB Journal*, 19(4):585–602, 2010.

- [70] J. Joseph, F. Doshi-Velez, and N. Roy. A bayesian nonparametric approach to modeling mobility patterns. In *AAAI*, pages 1587–1593, 2010.
- [71] H. Karimi and X. Liu. A predictive location model for location-based services. In *GIS*, pages 126–133, 2003.
- [72] E. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.
- [73] M. Khashei and M. Bijari. A novel hybridization of artificial neural networks and arima models for time series forecasting. *Applied Soft Computing*, 11(2):2664–2675, 2011.
- [74] J. Kleinberg. Computing: The wireless epidemic. *Nature*, 449:287–288, 2007.
- [75] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp*, pages 243–260, 2006.
- [76] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, June 1992.
- [77] R. Levin, Y. Kanza, E. Safra, and Y. Sagiv. Interactive route search in the presence of order constraints. *PVLDB*, 3(1):117–128, 2010.
- [78] X. Li, Z. Li, J. Han, and J. Lee. Temporal outlier detection in vehicle traffic data. In *ICDE*, pages 1319–1322, 2009.
- [79] M. Lin, W.-J. Hsu, and Z. Q. Lee. Predictability of individuals’ mobility with high-resolution positioning data. In *UbiComp*, pages 381–390, 2012.
- [80] L. P. Maguire, B. Roche, T. M. McGinnity, and L. McDaid. Predicting a chaotic time series using a fuzzy neural network. *Information Sciences*, 112(1):125–136, 1998.
- [81] I. Markovskiy. Low rank approximation: Algorithms, implementation, applications. 2012.
- [82] J. McNames. A nearest trajectory strategy for time series prediction. In *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*, pages 112–128, 1998.
- [83] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *SIGKDD*, pages 637–646, 2009.
- [84] Y. Moon, K. Whang, and W. Loh. Duality-based subsequence matching in time-series databases. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, pages 263–272. IEEE, 2001.

- [85] M. Morzy. Prediction of moving object location based on frequent trajectories. In *ISCIS*, pages 583–592, 2006.
- [86] M. Morzy. Mining frequent trajectories of moving objects for location prediction. In *MLDM*, pages 667–680, 2007.
- [87] K.-R. Müller, A. J. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In *ICANN*, pages 999–1004. Springer-Verlag, 1997.
- [88] M. Musolesi and C. Mascolo. Mobility models for systems evaluation. a survey. *Middleware for Network Eccentric and Mobile Applications*, pages 43–62, 2009.
- [89] T. Nguyen, Z. He, R. Zhang, and P. Ward. Boosting moving object indexing through velocity partitioning. *PVLDB*, 5(9):860–871, 2012.
- [90] C. J. Paciorek and M. J. Schervish. Nonstationary covariance functions for gaussian process regression. *Advances in neural information processing systems*, 16:273–280, 2004.
- [91] J. Patel, Y. Chen, and V. Chakka. Stripes: an efficient index for predicted trajectories. In *SIGMOD*, pages 635–646, 2004.
- [92] PeMS. Freeway performance measurement system. <http://pems.dot.ca.gov/>, 2014.
- [93] D. S. Poskitt and A. R. Tremayne. The selection and use of linear and bilinear time series models. *International Journal of Forecasting*, 2(1):101–114, 1986.
- [94] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, pages 262–270, 2012.
- [95] C. Rasmussen and C. Williams. Gaussian processes for machine learning. 2006.
- [96] C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *SDM*, pages 506–510, 2005.
- [97] T. Rattenbury, N. Good, and M. Naaman. Towards automatic extraction of event and place semantics from flickr tags. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 103–110, 2007.

- [98] G. Ristanoski, W. Liu, and J. Bailey. A time-dependent enhanced support vector machine for time series regression. In *KDD*, pages 946–954, 2013.
- [99] G. Ristanoski, W. Liu, and J. Bailey. Time series forecasting using distribution enhanced linear regression. In *PAKDD*, pages 484–495. 2013.
- [100] A. Sadilek and J. Krumm. Far out: Predicting long-term human mobility. In *AAAI*, pages 814–820, 2012.
- [101] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.
- [102] D. Sart, A. Mueen, W. Najjar, E. Keogh, and V. Niennattrakul. Accelerating dynamic time warping subsequence search with gpus and fpgas. In *ICDM*, pages 1001–1006, 2010.
- [103] N. Segata and E. Blanzieri. Fast and scalable local kernel machines. *The Journal of Machine Learning Research*, 11:1883–1926, 2010.
- [104] L. T. A. Singapore. Carpark lots availability. http://www.mytransport.sg/content/mytransport/home/dataMall.html#Traffic_Related, 2014.
- [105] S. Sundararajan and S. Keerthi. Predictive approaches for choosing hyperparameters in gaussian processes. *Neural Computation*, 13(5):1103–1118, 2001.
- [106] C.-H. Tai, D.-N. Yang, L.-T. Lin, and M.-S. Chen. Recommending personalized scenic itinerary with geo-tagged photos. In *ICME*, pages 1209–1212, 2008.
- [107] L. Tang, X. Yu, S. Kim, J. Han, W. Peng, Y. Sun, H. Gonzalez, and S. Seith. Multidimensional analysis of atypical events in cyber-physical data. In *ICDE*, pages 1025–1036, 2012.
- [108] L. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C. Hung, and W. Peng. On discovery of traveling companions from streaming trajectories. In *ICDE*, pages 186–197, 2012.
- [109] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, pages 611–622, 2004.
- [110] Y. Tao, D. Papadias, J. Zhai, and Q. Li. Venn sampling: A novel prediction technique for moving objects. In *ICDE*, 2005.
- [111] M. K. C. Tay and C. Laugier. Modelling smooth paths using gaussian processes. In *Field and Service Robotics*, pages 381–390, 2008.

- [112] Y. Tremblay, S. A. Shaffer, S. L. Fowler, C. E. Kuhn, B. I. McDonald, M. J. Weise, C.-A. Bost, H. Weimerskirch, D. E. Crocker, M. E. Goebel, et al. Interpolation of animal tracking data in a fluid environment. *Journal of Experimental Biology*, 209(1):128–140, 2006.
- [113] I. W. Tsang, J. T. Kwok, and P.-M. Cheung. Core vector machines: Fast svm training on very large data sets. In *Journal of Machine Learning Research*, pages 363–392, 2005.
- [114] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209:1–10, February 2011.
- [115] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series. *VLDBJ*, 15(1):1–20.
- [116] M. Vlachos, G. Kollios, and D. Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.
- [117] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, 2000.
- [118] Y. Wang and B. Chaib-Draa. A knn based kalman filter gaussian process regression. In *AAAI*, pages 1771–1777, 2013.
- [119] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.
- [120] T. White. *Hadoop: The definitive guide*. OReilly Media, Inc., 2012.
- [121] R. R. Wilcox. *Introduction to robust estimation and hypothesis testing*. Elsevier Academic Press, 2005.
- [122] P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management Science*, 6(3):324–342, 1960.
- [123] W. Wu, W. S. Ng, S. Krishnaswamy, and A. Sinha. To taxi or not to taxi? - enabling personalised and real-time transportation decisions for mobile users. In *MDM*, pages 320–323, 2012.
- [124] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *ICDE*, 2013.
- [125] W. Yan, H. Qiu, and Y. Xue. Gaussian process for long-term time-series forecasting. In *IJCNN*, pages 3420–3427, 2009.

- [126] H. Yang, L. Chan, and I. King. Support vector machine regression for volatile stock market prediction. In *Intelligent Data Engineering and Automated Learning*, pages 391–396. 2002.
- [127] G. Yavaş, D. Katsaros, Ö. Ulusoy, and Y. Manolopoulos. A data mining approach for location prediction in mobile environments. *Data & Knowledge Engineering*, 54(2):121–146, 2005.
- [128] J. Ying, W. Lee, T. Weng, and V. Tseng. Semantic trajectory mining for location prediction. In *GIS*, pages 34–43, 2011.
- [129] H. Yoon, Y. Zheng, X. Xie, and W. Woo. Smart itinerary recommendation based on user-generated gps trajectories. In *Proceedings of the 7th international conference on Ubiquitous intelligence and computing*, UIC, pages 19–34, 2010.
- [130] H. Yoon, Y. Zheng, X. Xie, and W. Woo. Social itinerary recommendation from user-generated digital trails. *Personal and Ubiquitous Computing*, 16(5):469–484, 2012.
- [131] G. U. Yule. On a method of investigating periodicities in disturbed series, with special reference to wolfer’s sunspot numbers. *Philos. Trans. Roy. Soc.*, A:267–298, 1927.
- [132] A. Zakai and Y. Ritov. Consistency and localizability. *The Journal of Machine Learning Research*, 10:827–856, 2009.
- [133] G. P. Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.
- [134] H. Zhang, A. C. Berg, M. Maire, and J. Malik. Svm-knn: Discriminative nearest neighbor classification for visual category recognition. In *CVPR*, volume 2, pages 2126–2136, 2006.
- [135] P. Zhang, B. J. Gao, X. Zhu, and L. Guo. Enabling fast lazy learning for data streams. In *ICDM*, pages 932–941, 2011.
- [136] Y. Zheng and X. Xie. Learning travel recommendations from user-generated gps traces. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(1):2, 2011.
- [137] Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W.-Y. Ma. Recommending friends and locations based on individual location history. *ACM Transactions on the Web (TWEB)*, 5(1):5, 2011.
- [138] Y. Zheng and X. Zhou. *Computing with spatial trajectories*. Springer, 2011.

- [139] B. Zhou, X. Wang, and X. Tang. Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents. In *CVPR*, pages 2871–2878, 2012.
- [140] J. Zhou, A. K. H. Tung, W. Wu, and W. S. Ng. A “Semi-Lazy” approach to probabilistic path prediction in dynamic environments. In *KDD*, 2013.
- [141] J. Zhou, A. K. H. Tung, W. Wu, and W. S. Ng. R2-d2 a system to support probabilistic path prediction in dynamic environments (demo). In *VLDB*, 2013.
- [142] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD*, pages 181–192, 2003.