R2-D2: a System to Support Probabilistic Path Prediction in Dynamic Environments via "Semi-Lazy" Learning

Jingbo Zhou [†], Anthony K. H. Tung [†], Wei Wu [#] and Wee Siong Ng [#]
[†] School of Computing, National University of Singapore
[#] Institute for Infocomm Research, A*STAR, Singapore
{jzhou, atung}@comp.nus.edu.sg, {wwu, wsng}@i2r.a-star.edu.sg

ABSTRACT

Path prediction is presently an important area of research with a wide range of applications. However, most of the existing path prediction solutions are based on eager learning methods which commit to a model or a set of patterns extracted from historical trajectories. Such methods do not perform very well in dynamic environments where the objects' trajectories are affected by many irregular factors which are not captured by pre-defined models or patterns.

In this demonstration, we present the "R2-D2" system that supports probabilistic path prediction in dynamic environments. The core of our system is a "*semi-lazy*" learning approach to probabilistic path prediction which builds a prediction model on the fly using historical trajectories that are selected dynamically based on the trajectories of target objects. Our "R2-D2" system has a visual interface that shows how our path prediction algorithm works on several real-world datasets. It also allows us to experiment with various parameter settings.

1. INTRODUCTION

Path prediction is very useful in a broad range of applications, including location-based services, traffic management, epidemic prevention, event prediction, and anomaly detection[7]. For example, knowing the future path of mobile phone users gives mobile advertisement companies the opportunities to push more relevant advertisements to them. As another example, being able to predict the future path of vehicles makes it possible for traffic management officers to control traffic flow in advance to prevent traffic jams.

We observe that, in these real world applications, the environments where the objects move in are rather dynamic. The objects' trajectories in such environments are frequently affected by some short-term and irregular factors such as traffic signals, traffic jams, weather conditions, and events such as festival and sports games.

Although a few path prediction methods have been proposed [6, 3, 5], they do not work well in dynamic environments. This is because most of them adopt the eager learning approach and are not able to capture the dynamic characteristics of the environments. In eager learning, a model or a set of patterns are extracted from historical data in a pre-processing stage. The model/patterns are then used to predict the paths of target objects (whose future paths need to be predicted). The pre-processing step typically takes a long time and is not run frequently. As a result, the models or patterns may have been invalid when they are applied.

We develop a system, called "R2-D2"[1], for pRobabilistic path pReDiction in Dynamic environments. The core of our system is a "semi-lazy" learning approach to probabilistic path prediction [10]. The key idea of the "semi-lazy" approach is to retrieve relevant trajectories on the fly, and then use the relevant trajectories to build a path prediction model that is specific for the target object. In our "R2-D2" system, all objects' historical trajectories are kept and indexed. To perform prediction for a target object, the system uses the trajectory of the target object in the last few time steps as query trajectory to retrieve a small set of reference trajectories from historical trajectories. Then we apply sophisticated machine learning techniques on the reference trajectories to construct a local model, which can predict the future path of the target object.

Our "semi-lazy" learning path prediction approach has the following distinct features. First, unlike the eager learning approaches, we build prediction model on the fly. Since the reference trajectories are selected particularly for the target object based on its recent trajectory, the prediction model built on the fly is specific for the target object. Second, unlike the lazy learning approaches, we use slightly more sophisticated learning algorithms to derive accurate local models/patterns with acceptable delay. Since the number of reference trajectories is typically small, we can afford building more complex local models. Third, we dynamically construct new models/patterns if the predicted path does not match the actual path well, giving rise to a self-correcting continuous prediction method. Finally, our system outputs the longest path (in terms of time) with probability higher than a given threshold. Users can use the probability threshold to control the tradeoff between prediction length and prediction accuracy.

In this demonstration, we showcase the above key aspects of the "R2-D2" system using several real-world datasets. The system provides a visual interface that shows moving objects and their predicted path. The demonstration system also allows users to play with various parameter settings.

2. RELATED WORK

Existing path prediction methods can be grouped into two categories: pattern-based prediction method and model-based prediction method. Pattern-based methods can be future categorized into two classes: personal pattern-based methods [8] and general

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

Proceedings of the VLDB Endowment, Vol. 6, No. 12

pattern-based methods [4, 6]. Personal pattern-based methods require enough historical trajectories of the target object, and there could be privacy problems. Existing pattern-based methods take the eager learning approach that learns a model or a set of patterns in a time-consuming learning/mining step. Such methods do not work well in dynamic environments that require the system to capture new patterns as time goes.

Model-based methods use mathematical models to describe the movement of objects, such as recursive motion function model [9] and Markov model [2]. However, these methods can only predict a short-term path.

The main difference between our approach and existing solutions is that our algorithm takes a "semi-lazy" approach that learns a prediction model on the fly while most of the existing solutions rely on models or a set of patterns discovered in an offline learning stage.

3. PRELIMINARIES AND PROBLEM DEFINITION

Let $T^i =< O_1^i, ..., O_t^i, ... >$ be a trajectory, where $O_t^i = ((x, y), t)$ is a vector denoting that object O^i locates at location (x, y) at time t. We assume that all trajectories have synchronized timestamps. When this assumption is not valid, we interpolate the trajectories. We refer to the trajectory of the target object O^p in the last h time steps as h-backward trajectory and denote it as W_h, i.e., $W_h =< O^p_{t_0-h+1}, O^p_{t_0-h+2}, ..., O^p_{t_0} >$ where t_0 is the current time.

Our key idea is to find objects on the fly whose trajectories are similar to the target object's recent trajectory and then use their trajectories to build a prediction model. Such objects are called *reference objects* and are referred as RO of O^p. The trajectory points of the reference objects form the *reference points* of O^p at different timestamps. They are further used to derive possible future *states* and *path* of O^p.

Formally, RO of O^p are the set of objects which have sub-trajectories that are similar to W_h. For each object $O^i \in RO$, we denote O^i_v as the timestamped location of O^i that is nearest to $O^p_{t_0}$. In other words, v is the timestamp when object O^i's location is closest to $O^p_{t_0}$.

The reference points of O^p at time t_0 are defined as $RO_0 = \{O^i_v | O^i \in RO\}$. The reference points of O^p at time t_0+k, namely RO_k, are defined as $RO_k = \{O^i_{v+k} | O^i \in RO\}$. Moreover, we use $RO_{1:k}$ to denote the reference trajectories of O^p, which are all the reference points of O^p from $t_0 + 1$ to $t_0 + k$, i.e., $RO_{1:k} = \bigcup_{i=1}^{k} RO_i$.

Let $\odot(centroid, radius)$ denote a circle. We call a vector $s_k = (\odot(centroid, radius), k)$ a state of O^p, which means O^p may be within the circle at time $t_0 + k$. We define the set of all possible states of O^p at time $t_0 + k$ as a state space, i.e., $S_k = \bigcup s^i_k$. We denote a sequence of states of O^p from t_0+1 to t_0+k as $SS_{1:k} =< s_1, s_2, ..., s_k >$. 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 $p(SS_{1:k}|RO_{1:k})$. Now, we can define the probabilistic path prediction problem.

DEFINITION 3.1 (**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) $p(SS_{1:k}|RO_{1:k}) \geq \theta$, and (2) for any path of length $k + 1$ time steps, $p(SS_{1:k+1}|RO_{1:k+1}) < \theta$.*

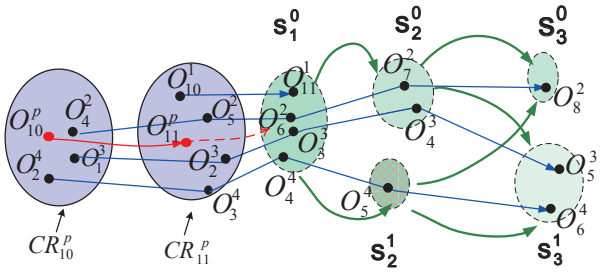

Figure 1: An illustration of probabilistic path prediction. Current time is $t_0 = 11$. 2-backward trajectory of O^p is $W_2 =< O^p_{10}, O^p_{11} >$. Reference objects of O^p are $RO = \{O^2, O^3, O^4\}$. Reference points at $k = 2$ are $RO_2 = \{O^2_7, O^3_4, O^4_5\}$. If $\theta = 0.4$, a possible path of O^p is $SS_{1:2} =< s_1^0, s_2^0 >$; if $\theta = 0.2$, a possible path of O^p is $SS_{1:3} =< s_1^0, s_2^0, s_3^1 >$.

4. SYSTEM ARCHITECTURE

4.1 Overview

Figure 2 depicts the architecture of our "R2-D2" system. It has two main components: the Trajectory Grid (TG) and the Prediction Filter (PF). TG indexes the moving objects' trajectories, and PF performs probabilistic path prediction. There are also two processes: "Update" and "Prediction". The "Update" process continuously collects trajectories of moving objects and stores them into TG. The "Prediction" process is further divided into two sub-processes:"Lookup" and "Construction". The "Lookup" process retrieves reference trajectories from TG, and the "Construction" process uses the reference trajectories to build path prediction model for the target object O^p.

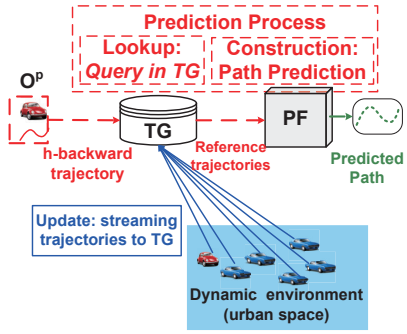

Figure 2: Architecture of the "R2-D2" system. It has a "Update" process (blue solid lines) and a "Prediction" process (red dotted lines). The "Prediction" process has two sub-processes: "Lookup" process and "Construction" process.

4.2 Trajectory Grid and Update Process

The Trajectory Grid (TG) is a multi-level grid structure that indexes the trajectories. It divides the area of interest into a set of rectangular cells with fixed size. Trajectories are indexed in the cells they pass.

In each cell, we maintain two data structures: a density counter and a hash table called traHash. The density counter records the number of moving objects that visit the cell. It indicates the popularity of the cell. The key of the hash table traHash is a vector (O^{id}, t), and the value of that is a vector (x, y). The key (i.e., (O^{id}, t)) records that O^{id} passes this cell at the time t, and the value (i.e., (x, y)) is the coordinate of the next cell that O^{id} will pass.

With the help of traHash, knowing O^i is in cell(i,j) at the time t immediately enables us to retrieve its following trajectory after time t.

The "Update" process continuously collects streaming trajectories of moving objects from the dynamic environment (see Figure 2) and TG indexes these trajectories. TG only buffers the moving objects' trajectories in the most recent H time units. We use a list to link all the elements of traHash, and the oldest element is always at the end of the linked list. When the moving objects report their new locations, TG not only updates the relevant cells' density and traHash, but also checks the oldest element of each traHash. Then TG discards the expired elements of each traHash and updates corresponding density counters.

4.3 Lookup Process

The "Lookup" process retrieves reference objects from TG. The idea is that if O^i has a sub-trajectory that matches very well with the h-backward trajectory of O^p, we say O^i is a reference object of O^p. We define a formal matching function which has clear semantic meanings and can support high performance query in TG. The query process works as follows. For each $O^p_u \in W_h$, we define a circle $CR_u = \odot(O^p_u, \epsilon)$. Then we obtain the objects that have visited any CR_u from TG. Finally the objects that visited all the CR_u in the same order as O^p did are reference objects. The value of the radius parameter ϵ is computed by multiplying the average velocity by half of the sampling time interval of the trajectories.

4.4 Prediction Filter and Construction Process

The Prediction Filter (PF) is a model for path prediction. The input of PF is a set of reference trajectories, and the output is the predicted path. The "Construction" process runs within PF. It iteratively constructs the state space and makes path prediction.

4.4.1 Prediction Filter

PF is based on the Grid-based Filter model [1], which is a generalization of Hidden Markov Model. We use RO_k, i.e. reference points of O^p at the time $t_0 + k$, as observations. The observations from time $t_0 + 1$ to time $t_0 + k$ are $RO_{1:k} = \bigcup_{j=1}^{k} RO_j$. 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 [1]:

$$p(s_k|RO_{1:k}) = \sum_i w^i_{k|k}\delta_i(s_k) \tag{1}$$

where

$$w^i_{k|k-1} \triangleq \sum_j w^j_{k-1|k-1} p(s^i_k|s^j_{k-1}) \tag{2}$$

$$w^i_{k|k} \triangleq \frac{w^i_{k|k-1} p(RO_k|s^i_k)}{\sum_j w^j_{k|k-1} p(RO_k|s^j_k)} \tag{3}$$

where $\delta()$ is a Dirac measure function and $w_{0|0} = 1$. The state transition function $p(s^i_k|s^j_{k-1})$ represents the probability that O^p will go to state s^i_k given that O^p is in state s^j_{k-1}at the time t_0+k-1. We model the state transition function by considering spatial relations and common reference objects between states s^j_{k-1} and s^i_k. The likelihood function $p(RO_k|s^i_k)$ represents the probability that RO_k is observed given O^p is in state s^i_k. By Bayes' theorem, we have $p(RO_k|s^i_k) \propto p(s^i_k|RO_k)p(RO_k)/p'(s^i_k)$, where $p(s^{p,i}_k|RO^p_k)$ can be computed according to the distribution of reference points RO^p_k in different states, and $p'(s^i_k)$ can be computed by the cells' average density covered by state s^i_k.

4.4.2 Construction Process

The "Construction" process has two stages, which are (1) to generate the possible state space of O^p at a future time $t = t_0 + k$ (t_0 is the current time) and (2) to make path prediction. The path prediction can be done by the maximum *a posteriori* (MAP) estimation in the state space by applying the Viterbi Algorithm. The returned result is a predicted path, whose probability (confidence) is larger than θ (set by users) and whose length (in term of time) is the longest. Now we focus on the state generation.

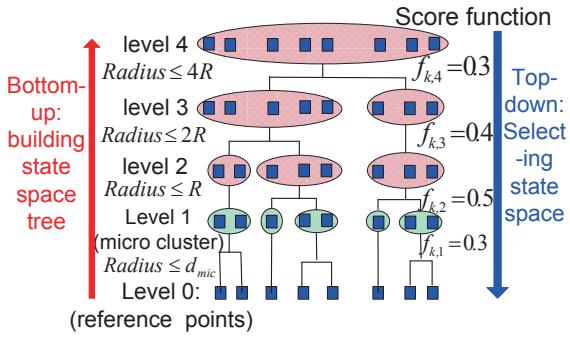

Figure 3: State generation at $t = t_0 + k$

In a nutshell, we cluster the reference points RO_k, and then convert each reference points cluster to a state. We propose a hierarchical method to generate the states. Figure 3 shows an illustration of the state generation process. The whole process of state generation has two steps. First, we build a state space tree in a bottom-up manner. We use a modified agglomerative hierarchical clustering algorithm to merge the reference points in RO_k until there is only one root cluster. All clusters in each level of the cluster tree are converted to one candidate state space of O^p (i.e., one cluster for one state). In this way, we get a state space tree from the cluster tree. Second, we use a score function to select the state space with the highest score from the state space tree in a top-down manner. Suppose the state with largest probability in state space $S_{k,l}$ is $s^*_{k,l}$, and the radius of $s^*_{k,l}$ is $r^*_{k,l}$, then the score function is:

$$f_{k,l} = \frac{p(s^*_{k,l}|RO_{1:k})}{[r^*_{k,l}]^\alpha} \tag{4}$$

Here, α controls the compromise between the probability and the radius. $p(s^*_{k,l}|RO_{1:k})$ is introduced in section 4.4.1.

Based on the observation that objects' locations change gradually, we also improve the efficiency of state generation by reusing micro clusters. Details can be found in [10].

4.4.3 Self-correcting continuous prediction

In real-life applications, we may need to continuously predict the path of a moving object. Continuous prediction gives us an opportunity to incrementally improve the path prediction result. During the prediction, the actual movement of the prediction target (O^p) can be compared against the predicted movement. With this information, we can incrementally refine our model.

The basic idea is to give more weight to the reference objects that have helped us make the correct prediction. We introduce a new attribute, called credit, for each moving object. We use a linear growth and exponential decay method to update the moving object's credit. If a reference point O^i contributed to generate the correct state which target object O^p will pass, we linearly increase

the credit of O^i; otherwise, we halve the credit of O^i. Then we integrate the credit (as weighted coefficient) into the state generation and path prediction algorithm.

5. DEMONSTRATION

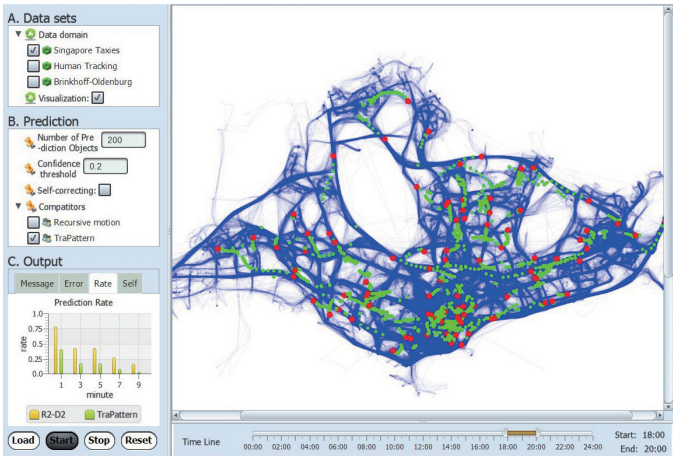

Figure 4: Screenshot of the main interface

We have implemented a web based system[2] and studied its performance with comprehensive experiments [10]. We would use this demonstration to show the key aspects of our system. We will run our system on different real-world and synthetic datasets. Users will be able to interact with our system by setting different application scenarios with regard to datasets and parameter values.

5.1 System Setup

We will demonstrate our system with two real-world datasets and one synthetic data set. The two real-world datasets are (1) the Singapore Taxies (ST) dataset, which is a collection of trajectories of about 15,000 taxis in Singapore over one day and contain more than 268 million points; and (2) Human Tracking (HT) dataset, which is a collection of human trajectories of 10,000 persons extracted from 30 minutes surveillance video in a train station and contains about 160 thousand points.

The synthetic dataset is a collection of synthetic trajectories of 200,000 objects on the road network of Oldenburg generated by Brinkhoff generator over 400 time units. The synthetic data set has 10 different kinds of moving objects and contains 80 million points. We also put 400 moving obstacles in the space to simulate the changes of the environment. Note that the system performance is mainly affected by the number of moving objects; therefore, the larger synthetic dataset is used to illustrate our system's scalability.

We compare with two existing path prediction methods: Recursive motion function [9] that is the most accurate motion function in literature, and TraPattern [6] that is a general pattern-based path prediction method.

5.2 Demo Interface

Figure 4 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 4, the red points represent the target objects whose path need to be predicted, and the sequences of green dots are the predicted path. The background blue map is the visualization of trajectories of other objects. The timeline bar lets us set the time interval for selecting datasets.

[2]http://db128gb-b.ddns.comp.nus.edu.sg/jzhou/R2-D2/

6. CONCLUSIONS

In this paper, we briefly present our approach to path prediction in dynamic environments and describe a demonstration of our "R2-D2" system. We describe the main components of "semi-lazy" path prediction method, explain how they work, and show its performance on real world and synthetic datasets. 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.

7. ACKNOWLEDGMENTS

This research was carried out at the SeSaMe Centre. It is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO. The work by Wei Wu and Wee Siong Ng was partially supported by the A*STAR Grant No. 1021580037.

8. REFERENCES

[1] M. 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.

[2] Y. Ishikawa, Y. Tsukamoto, and H. Kitagawa. Extracting mobility statistics from indexed spatio-temporal datasets. In *STDBM*, 2004.

[3] H. Jeung, Q. Liu, H. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, 2008.

[4] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD*, 2004.

[5] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *SIGKDD*, 2009.

[6] M. Morzy. Mining frequent trajectories of moving objects for location prediction. *Machine Learning and Data Mining in Pattern Recognition*, pages 667–680, 2007.

[7] C. Ratanamahatana, J. Lin, D. Gunopulos, E. Keogh, M. Vlachos, and G. Das. Mining time series data. In *Data Mining and Knowledge Discovery Handbook*, pages 1049–1077. 2010.

[8] A. Sadilek and J. Krumm. Far out: Predicting long-term human mobility. In *AAAI*, 2012.

[9] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, 2004.

[10] 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.