A "Semi-Lazy" Approach to Probabilistic Path Prediction in Dynamic Environments

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 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 path 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 approaches, 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 term of time. Users can control the confidence of the path prediction by setting a probability threshold. We conducted a comprehensive experimental study on real-world and synthetic datasets to show the effectiveness and efficiency of our approach.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database Applications-Data mining; I.2.6 [**ARTIFICIAL INTELLIGENCE**]: Learning

Keywords

Trajectory Analysis; Spatial-temporal Data Mining; Lazy Learning

1. INTRODUCTION

Path prediction is presently a popular area of research [13, 17, 18, 8, 16, 9, 28, 15, 20]. Predicting the future path of moving objects has a broad range of applications, including navigation, traffic management, personal positioning, actionable advertising [16], epidemic prevention [12], event prediction [21], anomaly detection [4, 14] and even spatial query optimization [5].

We study the path 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 [17, 8, 18, 28, 20], 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, new game settings), the models or patterns can become invalid.

To overcome these problems, we propose a "semi-lazy" approach, 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. 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.

Fig.1(a) 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 Fig.1(a)). In Fig.1(b), the red solid line is the trajectory of object O^p whose future path is to be predicted; while the blue dash lines are the historical trajectories which are selected as reference trajectories for predicting the path of O^p.

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

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

KDD'13, August 11–14, 2013, Chicago, Illinois, USA.

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.

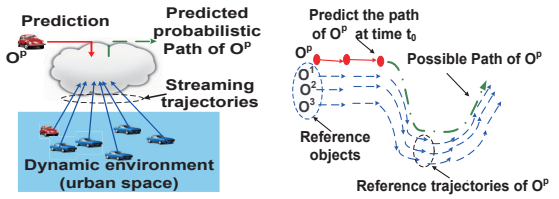


Figure 1: (a) An application example of R2-D2 in vehicle path prediction; (b) Path prediction based on reference trajectories.

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 term 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 be used in a broad range of applications that have different requirements for prediction confidence and predicted path length.

The rest of the paper is organized as follows. Section 2 gives a review of related work. Section 3 formalizes the problem 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 4, Section 5 and Section 6, respectively. We evaluate R2-D2 in Section 7 and conclude the paper in Section 8.

2. RELATED WORK

Approaches for path prediction can be categorized into pattern-based path prediction and descriptive model-based path prediction.

2.1 Pattern-based prediction

Pattern-based prediction methods can be future categorized into two classes: personal pattern-based methods and general pattern-based methods.

Personal pattern-based prediction methods consider the movement of each object to be independent. In [27], Yavas et al. propose a method to predict a user's future paths based on the user's mobility patterns. Jeung et al. [8] propose a hybrid prediction model which combines motion function and mobility patterns. Sadilek et al. [20] propose a model to support long-term mobility prediction.

General pattern-based prediction methods use common mobility patterns to predict future locations. In [17, 18], Morzy proposes methods to extract association rules from trajectories for prediction. For more of such methods, please refer to [28]. Mobility patterns are also used for destination prediction, such as WhereNext [16] and SubSyn [26], which predicts moving objects' destinations without concerning paths of reaching the destinations.

Pattern-based methods do not work well in dynamic environments. The main problem is that pattern mining is an expensive (in terms of time) process; therefore, it cannot quickly capture new patterns that emerge in a dynamic environment. Furthermore, the movement of many moving objects may not match any pattern, making it impossible to do prediction [28].

2.2 Descriptive model-based prediction

Descriptive model-based prediction methods use mathematical models to describe the movement of moving objects.

Motion function methods estimate objects' future locations by motion functions [24, 23]. Recursive motion function [23] is one of the most accurate methods. The problem is that these methods do not take environment constraints into account.

Markov model methods are suitable for estimating future locations of moving objects in a discrete location space [19]. However, these methods can only predict the short-term path. Some advanced mathematical models, such as Gaussian process and Dirichlet process [10], have been used to model objects' behavior, but they do not address the location prediction problem.

There are also some path prediction methods that utilize road networks [11, 9]. Those methods can only be applied in network-constraint spaces. R2-D2 does not have such limitations.

As far as we known, R2-D2 is the first "semi-lazy" approach that on the fly finds reference trajectories to learn a path prediction model and then make path prediction.

3. OVERVIEW AND PRELIMINARIES

Table 1: Table of Notations

O^i	moving object	O^p	object to be predicted
O^i_t	location of O^i at time t	RO	reference objects of O^p
t_0	current time	C^i_t	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^i_l	ζ^i in level l of cluster tree
s_k	state of O^p at t_0+k	S_k	state space at t_0+k
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		
$RO_{1:k}$	all reference points of O^p from t_0+1 to t_0+k		
$SS_{1:k}$	a path of O^p from t_0+1 to t_0+k		
$s^i_{k,l}$	the i-th state in level l of state space tree at time t_0+k		
CR_t	query range of O^p at time t for reference objects		
H	TG buffer interval threshold for dropping old trajectories		

3.1 Overview

Fig. 2 illustrates the 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.

There are also two separate processes: "Update" and "Prediction". In Fig. 2, we denote the "Update" process by blue solid lines and denote the "Prediction" process by red dotted lines. The "Update" process continuously collects streaming trajectories from the dynamic environment and stores them in TG (see **Section 4**).

The "Prediction" process makes path prediction. This process has two sub-processes: "Lookup" and "Construction". In Fig. 2, 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 5**). 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 6**).

3.2 Preliminaries

Table 1 lists notations used throughout the paper. We use O^i to denote a moving object. A trajectory of O^i is a sequence of timestamped locations $T^i =< O^i_1, ..., O^i_t, ... >$, where $O^i_t = ((x, y), t)$

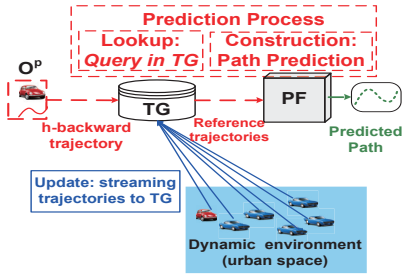


Figure 2: Architecture of R2-D2. It has an "Update" process (blue solid lines) and a "Prediction" process (red dotted lines). "Prediction" process has two sub-process: "Lookup" process and "Construction" process. (Better viewed in color.)

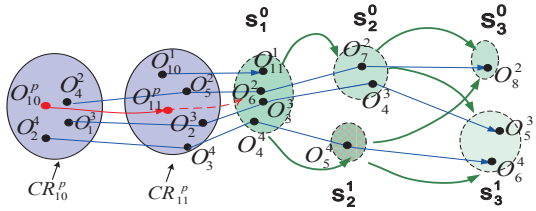


Figure 3: An example of using R2-D2 to predict O^p's path.

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.

We refer to the trajectory of O^p in the last h time steps as **h-backward trajectory** and denote it as W_h. If t_0 is the current time, then $W_h =< O^p_{t_0-h+1}, O^p_{t_0-h+2}, ..., O^p_{t_0} >$.

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 5.

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 t_0 are defined as $RO_0 = \{O^i_v | O^i \in RO\}$. The **reference points** of O^p at $t_0 + k$, namely RO_k, are defined as $RO_k = \{O^i_{v+k} | O^i \in RO\}$. Note that the definition of RO_k is based on RO_0, because the value of 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 $t_0 + 1$ to $t_0 + k$, i.e., $RO_{1:k} = \bigcup_{i=1}^{k} RO_i$. We also call $RO_{1:k}$ as **reference trajectories** of O^p.

Let $\odot(centroid, radius)$ 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. Since there could be several possible states for O^p at time $t_0 + k$, we use s^i_k 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 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, $p(SS_{1:k+1}|RO_{1:k+1}) < \theta$.*

EXAMPLE 1. *In Fig. 3, there are five moving objects: $\{O^p, O^1, O^2, O^3, O^4\}$. TG stores all their trajectories (see Section 4). 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. $W_2 =< O^p_{10}, O^p_{11} >$, to retrieve reference objects from TG (see Section 5), which are $RO = \{O^2, O^3, O^4\}$.*

Then, PF estimates a future path of O^p in two steps. First, at each future time $t = t_0 + k$, we generate the state space S_k of O^p (see Section 6.2). For example, when $k = 2$ we have $S_2 = \{s^0_2, s^1_2\}$. 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 6.1). In this example, if the user sets $\theta = 0.4$, the predicted path is $SS_{1:2} =< s^0_1, s^0_2 >$. If $\theta = 0.2$, the predicted path is $SS_{1:3} =< s^0_1, s^0_2, s^0_3 >$.

4. TG AND UPDATE PROCESS

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. Fig. 4(b) 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 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 "Prediction Filter" (see Eqn. (14)). 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 Fig. 4(a)).

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 Fig. 4(c)).

EXAMPLE 2. *In Fig. 4(a), O^1 is in $cell(2, 2)$ at time t, then it moves to $cell(5, 4)$ at time $t + 1$. 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 Fig. 4(c)).*

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 trahHashs and updates density counters.

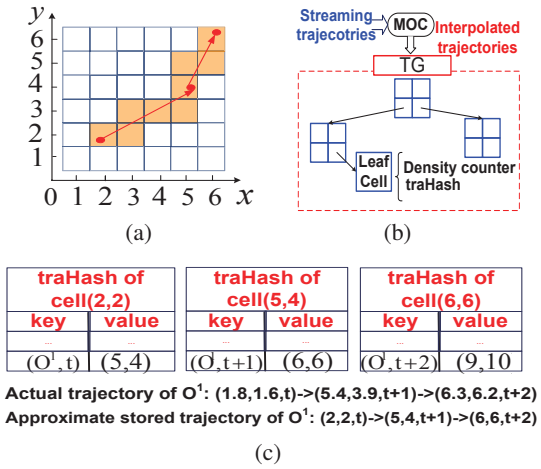


Figure 4: (a) Density update; (b) Overall structure of TG; (c) Example of stored trajectory.

5. 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 $dis(T^i, T^j) \leq \tau$, where $dis()$ 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 ($dis()$ is the Euclidean distance function):

DEFINITION 5.1 (**MATCH FUNCTION**). *Suppose $T_n^i = < O_1^i, ..., O_n^i >$ is a sub-trajectory of O^i and $W_h = \{O_1^p, O_2^p, ..., O_h^p\}$ is the h-backward trajectory of O^p, then we have* ***match*(T_n^i, W_h)=true** *if they satisfy:*

- ***(c1)*** $dis(O_1^i, O_1^p) \leq \epsilon$ *and* $dis(O_n^i, O_h^p) \leq \epsilon$;
- ***(c2)*** $\forall O_u^p \in W_h \exists O_v^i \in T_n^i (dis(O_v^i, O_u^p) \leq \epsilon)$;
- ***(c3)*** $\forall O_{u1}^p \in W_h \forall O_{u2}^p \in W_h (u1 < u2 \Rightarrow (\exists O_{v1}^i \in T_n^i \exists O_{v2}^i \in T_n^i (dis(O_{v1}^i, O_{u1}^p) \leq \epsilon \wedge dis(O_{v2}^i, O_{u2}^p) \leq \epsilon \wedge v1 < v2)))$

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. We omit the proof of this algorithm due to space constraints.

One 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. *In Fig. 3, 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 $RO_0 = \{O_5^2, O_2^3, O_3^4\}$.*

6. PF AND 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 6.1. Then, we introduce a hierarchical method to generate state spaces in Section 6.2. We then explain some important functions used in PF in Section 6.3. Finally, we discuss self-correcting continuous prediction in Section 6.4.

6.1 PF and probabilistic path prediction

PF is based on the Grid-based Filter model[1], 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 [1]:

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

where

$$w_{k|k-1}^i \triangleq \sum_j w_{k-1|k-1}^j p(s_k^i|s_{k-1}^j) \quad (2)$$

$$w_{k|k}^i \triangleq \frac{w_{k|k-1}^i p(RO_k|s_k^i)}{\sum_j w_{k|k-1}^j p(RO_k|s_k^j)} \quad (3)$$

Note that $\delta()$ 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 6.3. How to generate a state space is discussed in Section 6.2.

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 $SS_{1:k} =< s_1, ..., s_k >$ 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:

$$\eta_{k-1}(j) = \max_{<s_1,...,s_{k-2}>} p(< s_1, ..., s_{k-2}, s_{k-1}^j > |RO_{1:k-1})$$

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

$$\eta_k(i) = \max_j [\eta_{k-1}(j) p(s_k^i|s_{k-1}^j)] p(RO_k|s_k^i) \quad (4)$$

Eqn. (4) can be solved by dynamic programming algorithms, such as the Viterbi Algorithm [7]. Algorithm 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 term of time) is longest.

EXAMPLE 4. *In Fig. 3, 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 $SS_{1:2} =< s_1^0, s_2^0 >$; if $\theta = 0.2$, then $SS_{1:3} =< s_1^0, s_2^0, s_3^0 >$.*

6.2 Generate states by a hierarchical method

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

Algorithm 1: Probabilistic Path Prediction Algorithm

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

$\eta_0(0) = 1, \psi_0(0) = 0, \theta^* = 1, k = 0$
// "Lookup" process, see Section 5
get RO and RO_0 from TG by W_h
If $|RO| < \xi_m$ **Then return** NIL
// "Construction" process, see Section 6
while $\theta^* \geq \theta$ **do**
  k=k+1
  get RO_k from TG based on RO_{k-1}
  generate state space S_k// Section 6.2
  for i=*1 to* $|S_k|$ **do**
    $\eta_k(i) = \max_j[\eta_{k-1}(j)p(s_k^i|s_{k-1}^j)]p(RO_k|s_k^i)$
    $\psi_k(i) = \arg\max_j[\eta_{k-1}(j)p(s_k^i|s_{k-1}^j)]$
  $\theta^* = \max_i[\eta_k(i)]$
Backtrack SS from matrix ψ and return SS

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 Fig. 5). Moreover, we define a score function to find local optimal that balances the area size against the probability of the state.

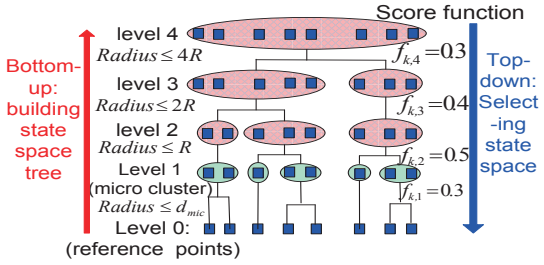


Figure 5: State generation at $t = t_0 + k$

6.2.1 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:

$$Centroid(\zeta) = \frac{\sum_{O_v^i \in \zeta} O_v^i}{|\zeta|}$$
$$Radius(\zeta) = \frac{\sum_{O_v^i \in \zeta} ||O_v^i - Centroid(\zeta)||}{|\zeta|} \quad (5)$$

The mean distance between two clusters is defined as :

$$dis_{mean}(\zeta^i, \zeta^j) = ||Centroid(\zeta^i) - Centroid(\zeta^j)|| \quad (6)$$

The bottom-up red arrow in Fig. 5 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 5) 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 $s_{k,l}^i = (\odot(Centroid(\zeta_l^i), Radius(\zeta_l^i)), k)$. 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 [2]. 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 [6], and every point needs to merge with others only $O(1)$ times. To sum up, the total time complexity is $O(n^{\frac{3}{2}})$.

6.2.2 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:

$$f_{k,l} = \frac{p(s_{k,l}^*|RO_{1:k})}{(r_{k,l}^*)^\alpha} \quad (7)$$

α in Eqn. (7) 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 $t_0 + k$, e.g. in Fig. 5 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.

6.2.3 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 [22]. By reusing previous clusters at time $t_0 + k - 1$, we can reduce the time cost for state generation at time $t_0 + k$.

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 points 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 $O_v^i \in \zeta^j$, we check whether it is within the circle $\odot(Centroid(\zeta^j), d_{mic})$. If O_v^i is outside of the circle, it will be split out as a new micro cluster [22]. 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 6.2.1.

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 $m' << n$.

6.3 Functions in the Prediction Filter

6.3.1 State Transition Function

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 at time t_0+k given that O^p is in state s^j_{k-1} at time t_0+k-1. 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^j_{k-1} and s^i_k, and is defined as follows:

$$J(s^i_k, s^j_{k-1}) = \frac{|s^i_k \bigcap s^j_{k-1}|}{|s^i_k \bigcup s^j_{k-1}|} \tag{8}$$

$|s^i_k \bigcap s^j_{k-1}|$ denotes the intersection area size of s^i_k and s^j_{k-1}, and $|s^i_k \bigcup s^j_{k-1}|$ denotes the union area size of s^i_k and s^j_{k-1}.

Connection factor is based on the common reference objects between s^j_{k-1} and s^i_k. We use $RO(s^i_k)$ to denote the reference objects within state s^i_k at t_0+k. For example, in Fig. 3, we have $RO(s^0_2) = \{O^2, O^3\}$. The connection factor is defined as

$$C(s^i_k, s^j_{k-1}) = \frac{|RO(s^i_k) \bigcap RO(s^j_{k-1})|}{|RO(s^i_k) \bigcup RO(s^j_{k-1})|} \tag{9}$$

We combine Eqn. (8) and Eqn. (9) into a linear function:

$$f(s^i_k, s^j_{k-1}) = \lambda J(s^i_k, s^j_{k-1}) + (1-\lambda) C(s^i_k, s^j_{k-1}) \tag{10}$$

where λ is used to adjust the weight of these two factors. In our experiment, we set $\lambda = 0.5$. Initially, we have $J(s^i_1, s^0_0) = 0$ and $C(s^i_1, s^0_0) = \frac{|RO(s^i_1)|}{|RO|}$. We can see that a larger value of $f(s^i_k, s^j_{k-1})$ leads to a larger state transition probability. To sum up, we have:

$$p(s^i_k|s^j_{k-1}) \propto f(s^i_k, s^j_{k-1}) \tag{11}$$

6.3.2 Likelihood Function

The likelihood function represents the probability that RO_k is observed given O^p is in state s^i_k at time t_0+k. By Bayes' theorem, we have:

$$p(RO_k|s^i_k) = \frac{p(s^i_k|RO_k)p(RO_k)}{p'(s^i_k)} \tag{12}$$

$p(s^i_k|RO_k)$ 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^i_k, the higher value of $p(s^i_k|RO_k)$ is. Then we have:

$$p(s^i_k|RO_k) = \frac{|RO(s^i_k)|}{|RO|} \tag{13}$$

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

$$p'(s^i_k) = \frac{\rho(s^i_k)}{\sum_j \rho(s^j_k)} \tag{14}$$

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

$$p(RO_k|s^i_k) \propto \frac{p(s^i_k|RO_k)}{p'(s^i_k)} \tag{15}$$

6.4 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 5. *In Fig. 3, 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^0_1. 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^i_t ($C^i_t \in \mathbb{N}^0$) 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 $t+1$, the credits are computed as follows: **(s1)** Suppose O^p is in state s^j_1 at time $t+1$. For each $O^i \in RO^t$, if O^i contributes to generate s^j_1, we have $C^i_t = C^i_t + 1$ (*linear growth*); otherwise, $C^i_t = \lfloor \frac{1}{2} C^i_t \rfloor$ (*exponential decay*). **(s2)** Retrieve RO^{t+1} from TG at time $t+1$, and initialize the credits of the reference objects as 1. **(s3)** For each $O^i \in RO^t$, if $C^i_t > 0$, put O^i into RO^{t+1}; if O^i is already in RO^{t+1}, sum its credits.

To integrate this self-correcting method into our prediction model, we need two modifications. First, during the state generation (see Section 6.2), we use the weighted center of cluster $Centroid'(\zeta_k)$ to replace Eqn. (5):

$$Centroid'(\zeta) = \frac{\sum_{O^i_v \in \zeta} C^i_t O^i_v}{\sum_{O^i_v \in \zeta} C^i_t} \tag{16}$$

Second, we integrate credits (as weighted coefficients) into Eqn. (9) and Eqn. (13). For example, we use Eqn. (17) to replace Eqn. (13). Eqn. (9) is modified in the same way.

$$p'(s^i_k|RO_k) = \frac{\sum_{O^u \in RO(s^i_k)} C^u_t}{\sum_j C^j_t} \tag{17}$$

7. EXPERIMENT

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 [23] and TraPattern [18]; and study R2-D2' distinct features, such as the confidence threshold θ and the self-correcting continuous prediction. Efficiency issue is also discussed.

7.1 Experiment setup and measurement

Data sets: Two real-world data sets and four synthetic data sets are used in our experiment, Table 2 summaries their details:

Table 2: Data sets

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

Table 3: Parameter settings

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**)

[ST] is a collection of trajectories of 13,200 taxies in Singapore over one week [25]. 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 [29]. The video is 24 fps with a resolution 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 [3] to generate four collections of synthetic trajectories on the road map of Oldenburg (see Table 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 lists all parameters used throughout the experiments. Parameters are set to default values (**bold font**) unless explicitly specified.

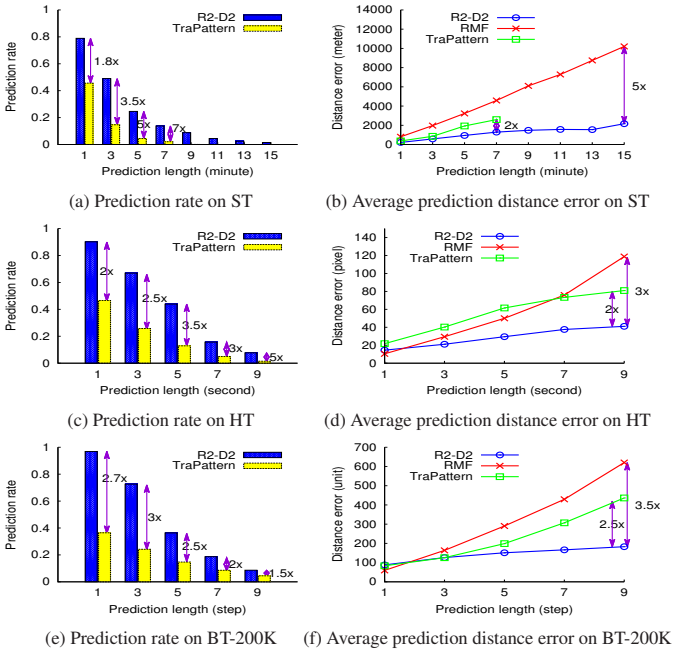
Competitors: R2-D2 is compared with two state-of-the-art algorithms: (1) Recursive Motion Function (RMF) [23] that is a descriptive model-based path prediction method and is the most accurate motion function in literature [23, 8]; and (2) Frequent Trajectory Pattern (TraPattern) [18] that is a general (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 between 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 [16, 8, 9].

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.

7.2 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.



(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

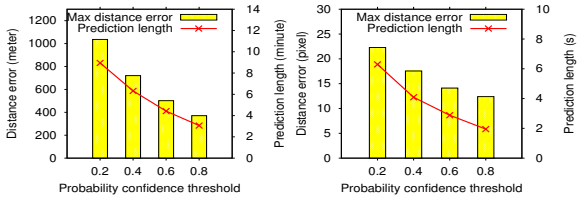
Figure 6: Comparison with competitors.

7.2.1 *Prediction rate*

We show the result on prediction rate in Fig. 6 (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 Fig. 6(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 Fig. 6(b).

7.2.2 *Average prediction distance error*

Fig. 6 (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 term 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



(a) ST (b) HT

Figure 7: Effect of confidence threshold θ

mined by TraPattern is about 8 minutes, therefore, in Fig. 6(b) the TraPattern cannot predict any path longer than 8 minutes.

7.3 Study of R2-D2's distinct features

In this section, 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%.

7.3.1 *Effect of confidence threshold*

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 Fig. 7, we show the result on ST and HT data sets. (R2-D2 has the similar result on BT, we do not show it due to space constraints.) 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.

7.3.2 *Self-correcting continuous prediction*

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 Fig. 8 show a clear trend that the performance of *conR2-D2* improves gradually with the continuous prediction. (BT data sets have the same trend, and we do not show them due to space constraints.) Since the ratios are noised over time, we use Bežier Curve to fit them over all the prediction steps.

7.4 Response time and scalability

We show that R2-D2 makes path prediction in real time. In Table 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. From BT-25K to BT-200K, we can see R2-D2 can scale linearly with the number of objects. Besides, the update process of TG is quite efficient, we do not discuss it due to space constraints.

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.

Table 4: Response time of R2-D2

data set	HT	ST	BT			
			25K	50K	100K	200K
avg. time (ms)	3.8	15.3	13.3	24.9	46.2	105.0

Effectiveness of reusing micro clusters: Table 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 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. d_{mic} has little effect on the prediction error and the prediction length, we do not show them due to space constraints.

Table 5: Response time VS. d_{mic} on ST data set

d_{mic} ($\times$ cell width)	1	3	5	7	9	11
avg. time (ms)	27.0	17.9	15.3	14.5	14.2	13.8

7.5 Effect of parameters

We present the effect of parameters in R2-D2. We only show the experimental results on ST due to space constraints. In Fig. 9, we use two y-axes: the left one is the maximal distance error and the right one 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 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 (in terms of time) of trajectories indexed in TG. From Fig. 9(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: Fig. 9(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 (Equ. (7)), shown in Fig. 9(c). A larger α leads to shorter prediction time length and smaller maximal distance error.

Table 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

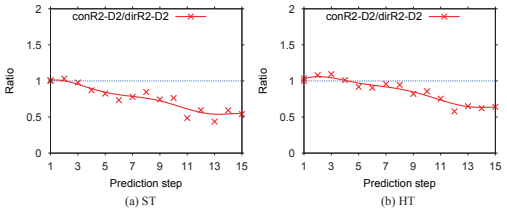


Figure 8: Self-correcting continuous prediction: ratio<1 indicates R2-D2 with self-correcting is better than R2-D2 without self-correcting.

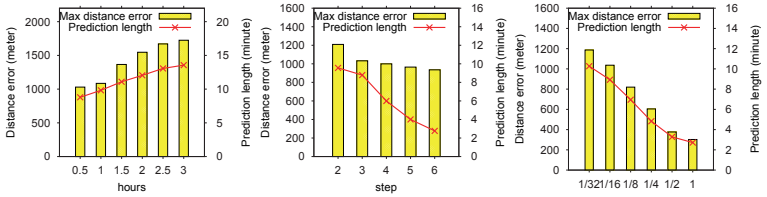


Figure 9: Effect of parameters

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.

8. CONCLUSION

In this paper, 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 "predict-on-the-go" approach can outperform existing eager learning methods in dynamic environments. We plan to apply this approach on other data mining tasks that are required to work in a dynamic environment.

9. ACKNOWLEDGMENTS

We thank all anonymous reviewers for insightful comments. 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.

10. 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] J. L. Bentley. Multidimensional binary search trees used for associative searching. *COMMUN. ACM*, 18(9):509–517, 1975.

[3] T. Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[4] Y. Bu, L. Chen, A. Fu, and D. Liu. Efficient anomaly monitoring over moving object trajectory streams. In *SIGKDD*, pages 159–168, 2009.

[5] R. Cheng, D. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *TKDE*, 16(9):1112–1127, 2004.

[6] W. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *J. of classification*, 1(1):7–24, 1984.

[7] G. Forney Jr. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[8] H. Jeung, Q. Liu, H. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, pages 70–79, 2008.

[9] 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.

[10] J. Joseph, F. Doshi-Velez, and N. Roy. A bayesian nonparametric approach to modeling mobility patterns. In *AAAI*, pages 1587–1593, 2010.

[11] H. Karimi and X. Liu. A predictive location model for location-based services. In *GIS*, pages 126–133, 2003.

[12] J. Kleinberg. Computing: The wireless epidemic. *Nature*, 449:287–288, 2007.

[13] J. Krumm and E. Horvitz. Predestination: Inferring destinations from partial trajectories. In *UbiComp*, pages 243–260, 2006.

[14] X. Li, Z. Li, J. Han, and J. Lee. Temporal outlier detection in vehicle traffic data. In *ICDE*, pages 1319–1322, 2009.

[15] M. Lin, W.-J. Hsu, and Z. Q. Lee. Predictability of individuals' mobility with high-resolution positioning data. In *UbiComp*, pages 381–390, 2012.

[16] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *SIGKDD*, pages 637–646, 2009.

[17] M. Morzy. Prediction of moving object location based on frequent trajectories. In *ISCIS*, pages 583–592, 2006.

[18] M. Morzy. Mining frequent trajectories of moving objects for location prediction. In *MLDM*, pages 667–680, 2007.

[19] M. Musolesi and C. Mascolo. Mobility models for systems evaluation. a survey. *Middleware for Network Eccentric and Mobile Applications*, pages 43–62, 2009.

[20] A. Sadilek and J. Krumm. Far out: Predicting long-term human mobility. In *AAAI*, pages 814–820, 2012.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] J. Ying, W. Lee, T. Weng, and V. Tseng. Semantic trajectory mining for location prediction. In *GIS*, pages 34–43, 2011.

[29] 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.