Destination Prediction by Sub-Trajectory Synthesis and Privacy Protection Against Such Prediction

Andy Yuan Xue [#1], Rui Zhang [#2], Yu Zheng [‡3], Xing Xie [‡4], Jin Huang [#5], Zhenghua Xu [#6]

[#] *University of Melbourne, Victoria, Australia*
[1] andy.xue@unimelb.edu.au [2] rui@csse.unimelb.edu.au
[5] jin.h@iojin.com [6] zhxu@student.unimelb.edu.au

[‡] *Microsoft Research Asia, Beijing, P.R.China*
[3,4] {yuzheng, xingx}@microsoft.com

***Abstract*—Destination prediction is an essential task for many emerging location based applications such as recommending sightseeing places and targeted advertising based on destination. A common approach to destination prediction is to derive the probability of a location being the destination based on historical trajectories. However, existing techniques using this approach suffer from the "data sparsity problem", i.e., the available historical trajectories is far from being able to cover all possible trajectories. This problem considerably limits the number of query trajectories that can obtain predicted destinations. We propose a novel method named *Sub-Trajectory Synthesis* (SubSyn) algorithm to address the data sparsity problem. SubSyn algorithm first decomposes historical trajectories into sub-trajectories comprising two neighbouring locations, and then connects the sub-trajectories into "synthesised" trajectories. The number of query trajectories that can have predicted destinations is exponentially increased by this means. Experiments based on real datasets show that SubSyn algorithm can predict destinations for up to ten times more query trajectories than a baseline algorithm while the SubSyn prediction algorithm runs over two orders of magnitude faster than the baseline algorithm. In this paper, we also consider the privacy protection issue in case an adversary uses SubSyn algorithm to derive sensitive location information of users. We propose an efficient algorithm to select a minimum number of locations a user has to hide on her trajectory in order to avoid privacy leak. Experiments also validate the high efficiency of the privacy protection algorithm.**

I. Introduction

As the usage of smart phones and in-car navigation systems becomes part of our daily lives, we benefit increasingly from various types of location based services (LBSs) such as route finding and location based social networking. A number of new location based applications require *destination prediction*, for example, to recommend sightseeing places, to send targeted advertisements based on destination, and to automatically set destination in navigation systems. Fig. 1 provides a schematic with the lines representing roads and the circles representing locations of interests (They may be road intersections, sightseeing places, shopping centres, etc.). If one drives from l_1 to l_4, an LBS provider may predict the most probable destinations to be l_7, l_8 and l_9 based on past popular routes taken by other drivers. As a result, the LBS provider can push advertisements of products currently on sale at those locations.

A common approach to destination prediction is to make use of historical spatial trajectories [30] of the public, available from trajectory sharing websites [10, 22], or large sets of taxi trajectories [21]. If an ongoing trip matches part of a popular route derived from historical trajectories, the destination of the popular route is very likely to be the destination of the ongoing trip (we refer to the ongoing trip as the *query trajectory*). Shown in Fig. 1 are five historical trajectories: $T_1 = \{l_1, l_2, l_5, l_6, l_9\}$, $T_2 = \{l_6, l_3, l_2\}$, $T_3 = \{l_4, l_5, l_8\}$, $T_4 = \{l_9, l_8, l_7\}$, and $T_5 = \{l_1, l_4, l_7\}$. Each trajectory is represented by a different type of line. For instance, a trip is taken from l_1 to l_4, and this query trajectory $\{l_1, l_4\}$ matches part of the historical trajectory T_5. Therefore, the destination of T_5 (i.e., l_7) is the predicted destination of the query trajectory. In practice, each trajectory here may be associated with a weight denoting the number of historical trajectories that exactly match this one, and the most popular trajectories are used for destination prediction. This idea has been described in further details in [31].

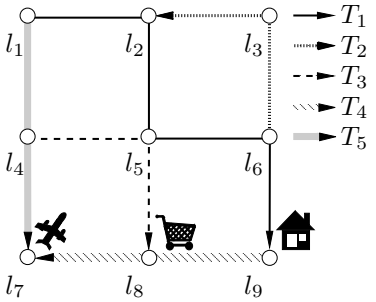


Fig. 1. An example of destination prediction

However, the above method has a significant drawback. A location l can be predicted as a destination only when the query trajectory matches a historical trajectory and the destination of the historical trajectory is l. In practice, l_8 and l_9 are also very likely to be the destination of the query trajectory, but will not be recommended to the user due to the limitation of the historical dataset. Moreover, if the query trajectory continues to l_5, the above method will not be able to predict any destination since no historical trajectory contains the trajectory $\{l_1, l_4, l_5\}$. We refer to this phenomenon as the **data sparsity problem**. This problem is inevitable in practice due to the following reasons. First, given a road network,

TABLE I
FREQUENTLY USED SYMBOLS

Symbol	Explanation
l_s, l_c, l_d	Starting, current, destination locations
D	The historical trajectory dataset
g	Granularity of a grid graph
$n_i(, n_s, n_c, n_d)$	i^{th} (, starting, current, destination) node
p_{ij}	Transition probability from n_i to adjacent n_j
$p_{i \to k}$	Total transition probability from n_i to n_k
$T_{i(,j)}$	Trajectories in D that contain $\{n_i(, n_j)\}$
$T_{d \in n_j}$	Trajectories in D with destination lying in n_j
T^p	Partial or query trajectory from n_s to n_c
$L_{s \to d}$	Length of $T_{s \to d}$ in ℓ_1 space grid graph
$L_{de,c \to d}$	Detour distance from n_c to n_d
M, M_{ij}	Transition matrix and its entry

the number of possible routes between all pairs of origin-destination is prohibitively large (exponential to the number of edges in the network), and currently the largest available real-life trajectory dataset covers only a tiny portion of it. Second, even trips with the same origin-destination pair may vary on their routes, making it unlikely to have identical trajectories.

In this paper, we propose a novel method to address the data sparsity problem. To begin with, we decompose all the trajectories into sub-trajectories comprising two neighbouring locations, then the sub-trajectories are connected together into "synthesised" trajectories. As long as the query trajectory matches part of any synthesised trajectory, the destination of the synthesised trajectory can be used for destination prediction. By this means, the coverage of trips on which we can make destination predictions is exponentially increased. The underlying process is formulated by a Markov model quantifying the correlation between adjacent locations with transition probabilities. We can compute the probability of reaching all the reachable locations from a given origin, and the top ranked ones are returned as predicted destinations. We call the above method the ***Sub**-Trajectory **Syn**thesis* (**SubSyn**) algorithm. For the aforementioned query trajectory $\{l_1, l_4, l_5, l_6\}$, SubSyn algorithm will be able to predict other destinations such as l_8 and l_9 since they can be synthesised using sub-trajectories of T_1, T_3, and T_5. The outcome of the destination prediction process will depend on the transition probabilities and the number of top destinations to be returned.

While SubSyn algorithm largely enhances the destination prediction capability of LBS providers, it can also be used by a malicious party to derive destinations which users do not wish to disclose such as homes and hospitals. Location based social networks such as Foursquare and Facebook Places allow users to automatically check in at locations they have visited [4, 7]. Using Fig. 1 as an example, a user leaves her workplace at l_1 and goes to a restaurant at l_4, then goes to a café at l_6 before heading home at l_9. She checks in at l_1, l_4 and l_6 sequentially. This allows for her trajectory $\{l_1, l_4, l_6\}$ to be revealed publicly on the social network. Suppose l_9 is a popular living area and plenty of historical trajectories go to l_9 from l_6; even though she does not check in at her home address, an adversary can predict her destination to be l_9 using SubSyn algorithm. Trajectory sharing websites and trajectory publication for data mining purposes also pose similar dangers.

In this paper, we also investigate on how to counter any privacy breach caused by destination prediction using algorithms like SubSyn. In particular, a user may choose not to check in (or publish) certain locations to prevent adversaries from deriving her destination. In the previous example, the user may manually choose not to check in at l_6 in order to reduce the probability of l_9 being the destination below a certain threshold. To mitigate the disturbance to the check-in service, we study which locations in the trajectory the user should not check in such that the number of locations that the user does not check in is minimised.

We make the following specific contributions in this paper:

- We identify the data sparsity problem in destination prediction and propose a novel *Sub-Trajectory Synthesis* (SubSyn) algorithm to address this problem. SubSyn algorithm decomposes historical trajectories into sub-trajectories and connect them into "synthesised" trajectories for destination prediction. This process is formulated based on a Markov model.
- SubSyn algorithm also achieves very high runtime efficiency because most values are directly fetched from pre-computed matrices. This is much faster than the baseline algorithm, which has to perform a large number of computation in order to find all matching trajectories.
- Concerned with potential privacy leak from abusive use of SubSyn algorithm, we further propose an algorithm named End-Points Generation Method to help identify locations on a trajectory which a user should not publish in order to retain confidential locational information.
- We conduct extensive experiments using real taxi trajectory datasets to investigate the effectiveness and efficiency of both SubSyn algorithm and the End-Points Generation Method for privacy protection. The results show that:
  - compared with a baseline algorithm, SubSyn algorithm can predict destinations for up to ten times more query trajectories while the SubSyn prediction algorithm runs over two orders of magnitude faster.
  - compared with a naive algorithm, the End-Points Generation Method is more than two orders of magnitude faster.

The remainder of the paper is organised as follows: Section II discusses related work and preliminaries. Our proposed SubSyn algorithm is presented in Section III. Section IV presents the privacy issue and our algorithm for privacy protection. Experimental results are reported in Section V. Section VI concludes the paper. Frequently used symbols are listed in Table I.

II. RELATED WORK AND PRELIMINARIES

In this section, we first discuss existing work on destination prediction. Then we focus on a Bayesian inference based approach to the destination prediction problem. Finally, we discuss the methods of protecting users' privacy.

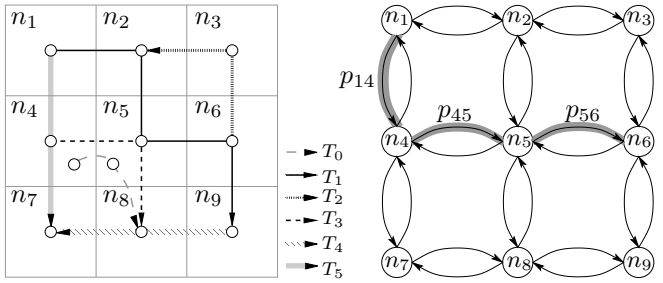
A. Destination Prediction

Although most destination prediction studies make use of historical trajectories, their focuses have mainly followed two streams: (i) using external information in addition to historical trajectories to help improve the accuracy of predicted destinations; (ii) personalised destination prediction for individual users. We describe each stream in more details below.

Employing external information in addition to historical trajectories can often enhance the prediction accuracy. For example, distributions of different districts (i.e., ground cover), travel time, trajectory's length [13–15], accident reports, road condition, and driving habits [31], have been incorporated into Bayesian inference to compute the probabilities of predicted destinations. Similarly, context information such as time-of-day, day-of-week, and velocity has been incorporated as the features in training the Bayesian network model for prediction [9]. The major inspiration behind these studies is that certain travelling pattern which fits into the acknowledged external settings shall bring higher possibilities to locations corresponding to those external settings in the historical dataset. However, since these studies mainly focus on the benefits brought by external information, their solutions are of little interest in the absence of the aforementioned ad-hoc external information. Our work considers a generic setting where only a historical trajectory set is assumed, and our focus is to solve the data sparsity problem which cannot be solved by adding external information. Therefore, the above studies are not applicable to our problem.

Personalised destination prediction trains prediction models using historical trajectories from an individual and then predicts destinations for this same individual. Thus, these predictions for the same query trajectory from different users may vary. Natalia and Chris [17] and Patterson *et al.* [20] used a Bayesian method to predict destination for specific individuals based on their historical transport modes. Markov model has been widely applied in predicting destinations for a specific individual as well [2, 3, 16, 23]. Tiesyte and Jensen [25] proposed a *Nearest-Neighbour Trajectory* (NNT) method that utilised distance measures to identify the historical trajectory which was the most similar to the current partial trajectory. Chen *et al.* [6] used a tree structure to represent the historical movement patterns and then matched the current partial trajectory by stepping down the tree. All these studies focused on predicting the repeated destinations of one or a group of specific individuals based on their own habits and historical travelling records. Our work considers a query trajectory from an unknown individual (without available personalised information). This is different from the personalised destination prediction studies. Therefore their solutions would be inapplicable to the data sparsity problem.

Amongst the above studies, Bayesian inference is the most popular framework used for deriving the probability of destinations based on historical trajectories [14, 15, 17, 20, 31]. Our approach also follows this framework. In the following subsection, we describe the Bayesian inference framework in more details and present a naive destination prediction method under this framework. It will be adapted to the baseline method for comparison in our experimental study.



(a) 3×3 grid on the example (b) 3×3 Markov model

Fig. 2. Grid graph representation and Markov model

B. Bayesian Inference Framework for Predicting Destination

Most studies [14, 27, 31] using the Bayesian inference have employed a grid representation of the data space including the road network as follows. The map is constructed as a two-dimensional grid consisting of $g \times g$ square cells. The granularity of this representation is a cell, i.e., all the locations within a single cell are considered to be the same object. Each cell has the side length of 1 and adjacent cells have the distance of 1. The whole grid is modelled as a graph where each cell corresponds to a node in the graph. A trajectory can be represented as a sequence of nodes according to the sequence of locations of the trajectory. An example of a 3×3 grid graph is given in Fig. 2a, where the trajectory T_1 can be represented as $\{n_1, n_2, n_5, n_6, n_9\}$. By representing the trajectories using nodes in a grid graph, similar trajectories are considered identical because a cell is the granularity of the graph. For example, in Fig. 2a, T_0 and T_3 are identical, both of which are represented as $\{n_4, n_5, n_8\}$. It is easy to observe that when the area of each grid cell becomes smaller, the different trajectories become more distinguishable from each other in the grid graph model.

Since query trajectories are incomplete trajectories whose destinations should be predicted by prediction algorithms, we denote them by *partial trajectories*, i.e., T^p. With a grid representation, two trajectories T_1 and T_2 have *exact match* with each other if and only if their sequences of nodes are identical, denoted by $T_1 = T_2$; a partial trajectory T^p *partially matches* a trajectory T if and only if their sequences start from the same node and the node sequence of T^p is fully contained by the node sequence of T, denoted by $T^p \subset T$. In the example shown in Fig. 2a, $T^p = \{n_1, n_4\}$ partially matches $T_5 = \{n_1, n_4, n_7\}$.

The Bayesian inference framework for destination prediction problem contains two phases: a *training phase* where the historical trajectories are mined offline and a *prediction phase* where a given query trajectory is analysed and answered with predicted destinations online [14, 15, 28, 31]. Specifically, the probability of a node n_j being the destination can be computed as the probability that n_j contains the destination location l_d, conditioning on the query trajectory T^p. Formally, the probability is computed using Bayer's rule as

$$P(d \in n_j|T^p) = \frac{P(T^p|d \in n_j)P(d \in n_j)}{\sum_{k=1}^{g^2} P(T^p|d \in n_k)P(d \in n_k)}, \quad (1)$$

where the prior probability $P(d \in n_j)$ can be easily computed as the number of trajectories terminating at n_j divided by the number of trajectories in the dataset. Formally,

$$P(d \in n_j) = \frac{|T_{d \in n_j}|}{|D|}, \quad (2)$$

where $|D|$ is the cardinality of the training dataset, and $|T_{d \in n_j}|$ is the number of trajectories in D that terminates at a location in n_j. As indicated by (2), only locations that are the destinations of historical trajectories will have non-zero prior probabilities, reflecting the fact that only locations that are popular among users are of interests.

Therefore, the problem lies in computing the posterior probability $P(T^p|d \in n_j)$. Ziebart *et al.* [31] described a method which first counts the number of trajectories satisfying two conditions: (i) it is partially matched by the query trajectory T^p; (ii) it terminates at a location in n_j. The count is then divided by the number of trajectories that terminate at a location in n_j to serve as the posterior probability. Formally,

$$P(T^p|d \in n_j) = \frac{|\{T_{d \in n_j}|T^p \subset T_{d \in n_j}\}|}{|T_{d \in n_j}|}, \quad (3)$$

where $|\{T_{d \in n_j}|T^p \subset T_{d \in n_j}\}|$ denotes the number of trajectories that satisfy both aforementioned conditions and $|T_{d \in n_j}|$ denotes the number of trajectories that terminate at a location in n_j. We refer to the above method as the ZMDB method after the authors' names of [31], which formulates the idea described in the second paragraph of Section I. As discussed there, this method may suffer from the data sparsity problem, i.e., the fact that the query trajectory cannot partially match any trajectory in D. This makes $|\{T_{d \in n_j}|T^p \subset T_{d \in n_j}\}|$ zero and all nodes to have zero probability of being the destination, i.e., $P(d \in n_j|T^p) = 0$. Consequently, no predicted destination can be returned.

C. Privacy Protection in Trajectory Publication

On protecting the users from privacy leak caused by trajectory based spatial-temporal queries, there are mainly four approaches. First, by *clustering* the trajectories within the same time period using the distance amongst locations as an indicator, trajectories can be aggregated in groups and a representative trajectory can be computed for the group, essentially hiding the original trajectories [1]. Second, in addition to grouping trajectories, the *generalisation-based* approach will further pick *atomic* points from the group and generate trajectories needed by the spatial-temporal queries based on these points [18]. Third, by iteratively *suppressing* (deleting) locations in trajectories, the results of spatial-temporal queries can also be tuned to exclude privacy concerns [24]. Fourth, grouping neighbouring cells in a *grid* into groups also meliorates the anonymisation of the original trajectories [8]. Our study on privacy protection against destination prediction lies in the third group since it attempts to delete locations in the query trajectories to alter the prediction result. The difference between our study and existing studies is that we focus on preventing locations with privacy concerns from appearing in the prediction results produced by SubSyn algorithm, where we are able to leverage the distinct property that only endpoints of a trajectory would affect the prediction results in order to dramatically eliminate unnecessary search space. Privacy protection and trajectory mining have also been studied in other contexts such as moving KNN query [11, 19] and group NN query [12]. Reference [30] contains a comprehensive survey on computation with spatial trajectories.

III. Destination Prediction Based on Sub-Trajectory Synthesis

In order to overcome the data sparsity problem, we propose a novel ***Sub**-Trajectory **Syn**thesis* (SubSyn) algorithm, which uses a Markov model to offline prepare the probabilities needed to efficiently compute the posterior probability for any given query trajectory online. Following the general framework mentioned in Section II-B, we use a grid graph to abstract the map and apply Bayer's rule as the prediction tool, i.e., using (2) to compute the prior probability and (1) to predict the probability of being a destination. Hence, the focus of this section is computing the posterior probability using SubSyn algorithm. We will first present the details of constructing Markov model to obtain transition probabilities between adjacent nodes in the grid graph. Next, we will propose an approach to synthesise the sub-trajectories using these transition probabilities. Finally, we will formulate the posterior probability equation $P(T^p|d \in n_j)$.

A. Constructing the Markov Model

To fully leverage the information of historical trajectories, a Markov model is constructed by associating a state to each node n_i in the grid graph. Two directed transitions of states corresponding to adjacent nodes n_i and n_j are established, i.e., n_i to n_j and n_j to n_i. The transition probability of travelling from a location in n_i to a location in n_j is denoted by p_{ij}. Fig. 2b shows an example of the Markov model associated with the example in Fig. 2a. These transition probabilities are conditional probabilities and can be computed as the number of trajectories that contain the sequence $\{n_i, n_j\}$ divided by the number of trajectories that contain the node n_i. Formally,

$$p_{ij} = P(n_j|n_i) = \frac{|T_{i,j}|}{|T_i|}. \quad (4)$$

For each pair of adjacent nodes in the grid graph, we compute the transition probabilities offline using (4). These probabilities are stored as entries of a two-dimensional $g^2 \times g^2$ matrix where one dimension corresponds to the node of current state and the other dimension corresponds to the next state. In the following sections, we denote the matrix and its entries by the transition matrix M and M_{ij}, respectively. Matrix (5) is the transition matrix of the example presented in Fig. 2.

As indicated by (4), since a first order Markov model is used here, only the current state determines the probability

of transiting to the next state. Higher order Markov models could be applied by involving previous states in addition to the current state in computing the probabilities. However, higher order models raise serious concerns when there is insufficient number of trajectories that support the computation of higher order transition probabilities [3, 5]. Following the practice of previous work, we also use the first order Markov model.

$$M = \begin{pmatrix} 0 & p_{12} & 0 & p_{14} & 0 & 0 & 0 & 0 & 0 \\ p_{21} & 0 & p_{23} & 0 & p_{25} & 0 & 0 & 0 & 0 \\ 0 & p_{32} & 0 & 0 & 0 & p_{36} & 0 & 0 & 0 \\ p_{41} & 0 & 0 & 0 & p_{45} & 0 & p_{47} & 0 & 0 \\ 0 & p_{52} & 0 & p_{54} & 0 & p_{56} & 0 & p_{58} & 0 \\ 0 & 0 & p_{63} & 0 & p_{65} & 0 & 0 & 0 & p_{69} \\ 0 & 0 & 0 & p_{74} & 0 & 0 & 0 & p_{78} & 0 \\ 0 & 0 & 0 & 0 & p_{85} & 0 & p_{87} & 0 & p_{89} \\ 0 & 0 & 0 & 0 & 0 & p_{96} & 0 & p_{98} & 0 \end{pmatrix} \quad (5)$$

B. Sub-Trajectory Synthesis

In the previous subsection (Section III-A), a Markov model based transition matrix M is constructed and filled with probabilities of travelling from a node to its adjacent node. This process, when inspected from another approach, is effectively the process of decomposing each trajectory in $\boldsymbol{D}$ into a set of sub-trajectories with length 2 (i.e, ordered pairs of neighbours). For instance, the trajectory T_1 in Fig. 1 is decomposed into $T^p_{1,2}$, $T^p_{2,5}$, $T^p_{5,6}$, and $T^p_{6,9}$ which in turn contribute to the transition probabilities p_{12}, p_{25}, p_{56}, and p_{69}, respectively. To synthesise sub-trajectories is to utilise the transition matrix M to compute the probability of being a destination of a query trajectory. The actual methods of sub-trajectory synthesis are presented in this subsection. Equations derived in this subsection will be incorporated in Section III-C to formulate the posterior probability $P(T^p|d \in n_j)$.

Synthesis of Detouring Path towards Destination: A useful value that can be generated from M is the sum of the probabilities of all possible paths between two nodes n_i and n_k. The following example will demonstrate the concept of this probability. By referring to (5) and Fig. 2, the probability of travelling from n_1 to n_6 is found to be zero in M (i.e., $M_{16} = 0$) because M stores the probability of travelling from one node to another in exactly one step, and there is no way of travelling between these two nodes within one step. Furthermore, when M is multiplied by itself to form M^2, its entries are the probabilities of travelling from one node to another in two steps. In general, M^r ($r \in [0, \infty)$) holds the probabilities of transition from one node to another in exactly r steps (i.e., M^r holds r-step transition probabilities). Since the ℓ_1 distance between n_1 and n_6 (i.e., $L_{1\to6}$) is 3, the probability of travelling from n_1 to n_6 via all the shortest paths can be found in matrix entry M^3_{16}. Intuitively, we wish to use this property to replace terms in both the numerator and denominator in (3). Two problems also remain: (i) the ℓ_1 distance does not necessarily correspond to the actual travelling distance from n_1 to n_6 because sometimes a small detour is taken due to various reasons. Hence we wish to find the sum of r-step transition probabilities of various steps; (ii) the number of paths from one node to another is infinitely large without restrictions, i.e., $r \in [L_{1\to6}, \infty)$. By examining the dataset used in our experiment, it is found that the distances of most trips do not exceed 1.2 of the ℓ_1 distance between the starting and finishing nodes. In other words, the detour distance L_{de} is, in most cases, less than 0.2 of the ℓ_1 distance of a trip. We set $L_{de,i\to k}$ to be $\lceil 0.2 L_{i\to k} \rceil$ as a typical value of detour distance. Therefore we define the *total transition probability* as follows.

Definition 1: **Total Transition Probability** The total transition probability of travelling from one node n_i to another node n_k, denoted by $p_{i\to k}$ [1], is the sum of the r-step transition probabilities of all possible paths (with the detour distance restriction) between n_i and n_k. Formally:

$$\begin{aligned} p_{i\to k} &= \sum_{r=L_{i\to k}}^{L_{i\to k}+L_{de,i\to k}} M^r_{ik} \\ &= M^{L_{i\to k}}_{ik} + M^{L_{i\to k}+1}_{ik} + \cdots + M^{L_{i\to k}+L_{de,i\to k}}_{ik}. \end{aligned} \quad (6)$$

In the equation above, the last term after expanding the summation equation, $M^{L_{i\to k}+L_{de,i\to k}}_{ik}$, gives the probability of travelling from n_i to n_k in exactly $L_{i\to k}+L_{de,i\to k}$ steps. This term is our longest distance restriction obtained by examining our experiment dataset. In (5) and Fig. 2, $p_{1\to6} = M^3_{16} + M^4_{16}$ since $L_{1\to6} = 3$ and $L_{de,1\to6} = 1$. The usage of (6) will be revealed in the following subsection (Section III-C) when formulating the posterior probability equation.

While (6) is in its simplest form, it is not computationally friendly. For a 30×30 grid graph[2], M becomes a $30^2 \times 30^2 = 900 \times 900$ matrix. For a typical travel distance of ten nodes, $p_{i\to k} = M^{10}_{ik} + M^{11}_{ik} + M^{12}_{ik}$ ($12 = \lceil 10 \times 1.2 \rceil$) which means that matrix multiplication operation needs to be performed on the huge matrix M more than 30 times where each matrix multiplication requires $O(n^{2.3736})$ ($n = 900$) time complexity [26]. While already inefficient, the same operation needs to be carried out for all pairs of nodes $\{n_i, n_k\}$ ($900^2 \approx 8.1 \times 10^5$ pairs). It is therefore infeasible in terms of running time. A tailored and efficient algorithm is introduced to solve the aforementioned issues that are presented to us. We use pseudo-code in Algorithm 1 to summarise the procedure (named *SubSyn-Training*) described below. As the name suggests, this stage is done as a pre-computation training stage which allows the actual online query stage to be processed instantaneously. The runtime efficiency measurements are presented in Section V.

Firstly, Equation (6) is reformed so that few redundant computations are carried out.

$$\begin{aligned} p_{i\to k} &= \sum_{r=L_{i\to k}}^{L_{i\to k}+L_{de,i\to k}} M^r_{ik} \\ &= M^{L_{i\to k}}_{ik} \sum_{r=0}^{L_{de,i\to k}} M^r_{ik} \\ &= M^{L_{i\to k}}_{ik} \cdot \left(M^0_{ik} + M^1_{ik} + \cdots + M^{L_{de,i\to k}}_{ik} \right). \end{aligned} \quad (7)$$

[1]Note the difference between $p_{i\to j}$ and p_{ij}. The latter is the transition probability between two **adjacent** nodes, and its definition was given in (4).

[2]30×30 grid graph gives the best accuracy in our experimental study.

For the purpose of explanation, we use a 30×30 grid graph as an example without any loss of generality. Since the longest distance $\max(L_{i\to k}) = L_{1\to 900}$ is $2 \times (30-1) = 58$, the maximum possible value of $L_{de,i\to k} = \lceil 0.2 L_{i\to k} \rceil$ is therefore $\lceil 0.2 \times 58 \rceil = \lceil 11.6 \rceil = 12$. By taking advantage of the concept of dynamic programming, an array of size 13 can be used to store all M^r, $r \in [0, 12]$ which are computed in ascending exponent order (Algorithm 1: lines 5-6) such that only 11 matrix multiplications are required since we already have M^1 and M^0 is the identity matrix I. Afterwards, we sequentially add each array element to the next element to form $\sum_{r=0}^{i} M^r$ (second factor in (7)) where i is an array index (Algorithm 1: lines 7-8). The benefit is evident because all possible values of $\sum_{r=0}^{L_{de,i\to k}} M_{ik}^r$ can be directly retrieved from this array for further computations.

Algorithm 1: SubSyn-Training(D, g)

```
1  M^T ← 0;                                          // total transition matrix
2  n ← ⌈0.2 × 2(g − 1)⌉;                             // maximum detour distance
3  A[n + 1] ← I;                                     // define an array to store Σ_{r=0}^{n} M^r
4  A[1] ← M ← D;                                     // construct transition matrix
5  for i ← 2 to n do
6      A[i] ← M · A[i − 1];                          // A[i] now holds M^i
7  for i ← 1 to n do
8      A[i] ← A[i] + A[i − 1];                       // A[i] now holds Σ_{r=0}^{i} M^r
9  list ← ∅;                                         // a list to store all node pairs
10 foreach n_i in grid graph do
11     if M_{i*} contains only zero entries then
12         continue;
13     foreach n_j in grid graph do
14         if M_{*j} contains only zero entries then
15             continue;
16         add node pair (n_i, n_j) to list;
17 sort list;                                        // increasing order of ℓ1 distance
18 M_power ← M;                                     // matrix to store intermediate result
19 M^T_temp ← M_power · A[1];                       // matrix to store intermediate result
20 L_prev ← 1 ;                                     // record distance of previous iteration
21 foreach (n_i, n_j) ∈ list do
22     while L_{i→j} >= L_prev + 1 do
23         M_power ← M · M_power ;
24         M^T_temp ← M_power · A[L_{de,i→j}];
25         L_prev++;
26     M^T_ij ← M^T_{temp,ij} ;                      // i.e., p_{i→j}
   return: M and M^T
```

Regarding the first factor $M_{ik}^{L_{i\to k}}$ in (7), we could use the same strategy except that in order to store this term for all pairs of nodes, too much memory is required, especially in a fine grid. Specifically, in a 30×30 grid graph, each M requires $30^2 \times 30^2 \times 8$Bytes ≈ 6.1MB of storage space. Since the maximum ℓ_1 distance in such a grid graph is 58, the total amount of memory required will exceed 350MB (i.e., 58×6.1MB). When using a finer grid graph, the amount of memory required increases rapidly. For instance, a 50×50 grid graph will require 4.6GB of memory to store all M^r, and an 80×80 grid graph will require 48GB. Therefore we need to seek a scalable and robust solution. Fortunately, these matrices do not have to be stored. Instead, we enumerate all pairs of nodes in the grid graph, sort these pairs in ascending order of their distance between each other, and compute $p_{i\to k}$ in this order (Algorithm 1: lines 9-17). Using Fig. 2 as an example, the all pairs of nodes and their distances are generated to be $\{n_1, n_2\}(1)$, $\{n_1, n_4\}(1)$, $\cdots$, $\{n_1, n_3\}(2)$, $\cdots$, $\{n_2, n_6\}(2)$, $\cdots$, $\{n_1, n_8\}(3)$, $\cdots$,$\{n_1, n_9\}(4)$, $\{n_9, n_1\}(4)$. In order to compute the total transition probability of each pair, M^1 and $\sum_{r=0}^{1} M_{ik}^r$ are retrieved from memory, and they are multiplied together to form a matrix containing $p_{i\to k}$ of distance 1 (Algorithm 1: lines 18-19). The total transition probabilities of all pairs of distance 1 can be obtained directly from this matrix (M_{power} in Algorithm 1). After all pairs of distance 1 are obtained, M^2 is computed by multiplying M, and a matrix containing $p_{i\to k}$ of distance 2 is obtained (Algorithm 1: lines 22-25). Utilising this algorithm, only less than 100 matrix multiplications are carried out (in the case of a 30×30 grid graph) to compute all total transition probabilities, whereas the intuitive approach requires millions of matrix multiplications. During the process, each found $p_{i\to k}$ is stored in a separate matrix M^T (Algorithm 1: line 26) which we call the *total transition matrix* and it holds the same number of entries as the transition matrix M.

In the process of enumerating all pairs of nodes, more computational steps could be eliminated by pruning unpromising pairs of nodes. For two nodes n_i and n_k, if either the entire row containing n_i, M_{i*}, or the entire column containing n_k, M_{*k} comprises only zero entries, the pair is discarded (Algorithm 1: lines 11-12, 14-15). It indicates a lack of training data in these nodes and the probability is always confined to zero in such case. Hence there is no need to compute their total transition probabilities. The running time of the algorithm is examined in the experiment section (Section V) and is found to be totally acceptable.

The memory space occupied by SubSyn-Training is trivial as explained below. By referring to Algorithm 1, the memory space needed is $(\lceil 0.2 \times 2(g-1) \rceil + 5) \cdot g^4 \times 8$Bytes. (i.e., $A[n+1]$, M, M_{power}, M^T, M^T_{temp}, and $list$). In a 30×30 grid graph, the space needed is 105MB, making the algorithm implementation feasible in most modern computers.

Synthesis of Path for Query Trajectory We provide a definition of *path probability* which will be used to compute the posterior probability. The path probability is the probability of a person travelling from one location to another via a specific path. Typically the path is the query trajectory provided by a user. The value of the path probability can be obtained through multiplying the transition probabilities between all pairs of nodes in this partial path T^p. For example, given the transition matrix M, the path probability of moving from a location in n_1 to another location in n_6 via the path $T^p_{1,4,5,6}$ can be obtained as follows: $P(T^p_{1,4,5,6}) = p_{14} \cdot p_{45} \cdot p_{56}$ where p_{14}, p_{45}, and p_{56} are the transition probabilities in the matrix M between consecutive and adjacent[3] node pairs $\{n_1, n_4\}$, $\{n_4, n_5\}$, and $\{n_5, n_6\}$, respectively. In general, given any partial trajectory

[3] *consecutive* nodes are two nodes next to each other in a trajectory; *adjacent* nodes are two nodes next to each other in a grid graph.

$T^p_{1,2,\cdots,k}$, the definition of path probability is:

$$P(T^p) = P(T^p_{1,2,\cdots,k}) = \prod_{i=1}^{k} p_{i(i+1)}. \quad (8)$$

When the sequence of nodes in a query trajectory does not fall into adjacent nodes, the transition probability would be zero. In such cases we use a linear interpolation to fill the gap between two non-adjacent consecutive nodes.

C. Computing the Posterior Probability

After defining the total transition probability in (6) and the path probability in (8), given a query trajectory T^p, we calculate the posterior probability of a user travelling from the staring node n_s to the current node n_c via T^p conditioned on the destination being in node n_j by (9):

$$P(T^p|d \in n_j) = \frac{P(T^p) \cdot p_{c\to j}}{p_{s\to j}}, \quad (9)$$

where $P(T^p)$ is the path probability of the given partial trajectory T^p; $p_{c\to j}$ is the total transition probability of moving from the current node of T^p, n_c, to a predicted destination $d \in n_j$; and $p_{s\to j}$ is the total transition probability of travelling from the starting node of T^p, n_s, to a predicted destination $d \in n_j$.

The posterior probability is used when a user issues a query to compute destination probabilities. We summarise the *SubSyn-Prediction* algorithm and present the pseudo-code in Algorithm 2. Given a transition matrix M, a total transition matrix M^T generated from taxi trajectories in a grid graph, and a partial query trajectory T^p, the overview of the *SubSyn-Prediction* algorithm is as follows: (i) We first calculate the path probability of the partial travelling trajectory T^p using (8); (ii) Then, for each node n_j, we calculate the posterior probability $P(T^p|d \in n_j)$ of moving via T^p given n_j being the destination; (iii) For each node n_j, we compute the destination probability $P(d \in n_j|T^p)$ based on $P(d \in n_j)$ and $P(T^p|d \in n_j)$; (iv) Finally, we sort the nodes according to their destination probabilities and return the top-k elements in the sorted list (i.e., a list of predicted destinations in descending order of their destination probabilities). It is clearly observed that the algorithm is extremely straightforward because of the offline training stage *SubSyn-Training* presented in Section III-B. In *SubSyn-Prediction*, few computations are carried out since most of the probabilities required can be directly fetched from pre-computed matrices M and M^T.

Algorithm 2: SubSyn-Prediction$\big(M, M^T, T^p\big)$

```
list ← ∅;                                   // a list to store the output
construct path probability P(T^p) from M;
foreach n_j in grid graph do
    retrieve p_{c→j} and p_{s→j} from M^T;
    compute P(T^p|d ∈ n_j), and hence P(d ∈ n_j|T^p) ;
    store P(d ∈ n_j|T^p) in list ;
sort list;
return: top-k elements in list
```

IV. Privacy Protection Against Prediction based on SubSyn Algorithm

As explained in Section I, LBS benefits both users and business providers. For instance, when a user uses her smart phone to "check in" at various places and share with her friends on social websites, she benefits from receiving special offers and discounts (e.g., voucher at a coffee shop), discovering new places (e.g., a restaurant recommended by a friend), and sharing cheerful moments (e.g., published location at the venue of an Olympic Games opening ceremony). As a business provider, the owner is able to create brand loyalty, have a social medium to engage with customers, and receive advertisement. Despite the aforementioned benefits that LBS brings, it also incurs possible locational *privacy leak*. Privacy is of high concern to a lot of people, and in many occasions privacy leak can cause serious safety threats. It is therefore vital that we develop a solution to counteract the potential privacy leak threat from abuse of SubSyn by malicious parties. In order to achieve this goal, we propose a privacy protection solution in this section to make the destination probability (or the destination rank) of a private location lower than a chosen threshold k through deleting the smallest number of nodes from the query trajectory (constructed from a list of user's locations).

Consider the following scenario in which a user takes advantage of our privacy protection solution to avoid privacy leak. Jane has a geo-social application on her smart phone, and during a Sunday afternoon she is on her way home from a public event. Right before arriving at her house, four locations (e.g., $\{l_1, l_4, l_5, l_6\}$ in Fig. 2) are recorded from auto check-ins when uploading a photo taken by her smart phone and when posting a message, and manual check-ins at the event venue and in a restaurant. If our privacy protection solution is in place, before publishing each location, Jane receives a confirmation dialogue showing a list of predicted destinations (in decreasing order of destination probability) should the locations be published. After taking a photo (auto check-in) at l_6 and using the list of locations as the query trajectory $T^p_{1,4,5,6}$, the confirmation dialogue reports a list of predicted nodes $\{n_9, n_3, n_8\}$ (cf. Fig.2) amongst which her actual residential place lies in the grid node n_9. Due to the concern of privacy leak of her residential place, she selects a rank threshold 3 which indicates that, after applying our solution, Jane would like the probability of n_9 fall below the third predicated destination. Our solution processes this privacy protection request and returns a new list of locations $\{l_4, l_5\}$ with predicted destinations $\{n_6, n_2, n_8\}$ which are the results of deleting the smallest number of locations (in this case l_1 and l_6 have been deleted) such that n_9 is not amongst the top-3 predicted destinations. Jane is satisfied with the resulting list of locations, and publishes them without worrying the issue of privacy leak. One could argue that the first location l_1 was deleted after being published in the first place. This generally does not pose a serious concern to a user because this location is deleted within a few hours, and hence it is only a threat

provided that a malicious party is constantly monitoring her activities on social websites. Furthermore, if a user requires a higher protection level, she should be able to select an option to save the list of check-in locations along a trip, and only publishes them after arriving at her destination. Based on the solution presented in this scenario, we give a formal definition of the *privacy protection* problem in our paper:

Definition 2: **Privacy Protection Against Prediction based on SubSyn Algorithm** This task identifies a set of locations in the query trajectory to be removed from publication such that the destination of the query trajectory will be predicted with a probability lower than a given threshold and the number of locations not published is minimised.

The remainder of this section presents our proposed solution. In order to respond to a large number of online queries submitted by users, the solution should be able to process these queries in a highly efficient way. We propose two methods to achieve this goal. Firstly we present an *Exhaustive Generation Method* (Section IV-A) which is intuitive, but suffers from low efficiency issue. We then present another method named *End-Points Generation Method* (Section IV-B) which is more efficient.

A. Exhaustive Generation Method

We first present the intuitive *Exhaustive Generation Method*. An example is used here to explain this method: Given a partial trajectory consisting of $r = 4$ nodes $T^p_{1,4,5,6}$, we first iteratively delete one node from this given partial trajectory. Four resulting sub-trajectories, each consisting of three nodes, will form (i.e., $T^p_{1,4,5}$, $T^p_{1,4,6}$, $T^p_{1,5,6}$, and $T^p_{4,5,6}$). We find the destination probability (and the destination rank) of the given private destination in each formed sub-trajectory using SubSyn-Prediction (linear interpolation is used to handle non-adjacent nodes). If any result satisfies the privacy threshold, we return the corresponding sub-trajectory to the user. Otherwise we continue generating a total of six new sub-trajectories of length 2 (i.e., $T^p_{1,4}$, $T^p_{1,5}$, $T^p_{1,6}$, $T^p_{4,5}$, $T^p_{4,6}$, and $T^p_{5,6}$). This process is repeated until either we find a suitable sub-trajectory which satisfies the privacy protection criterion (i.e., below a rank threshold), or when there is no valid node to be deleted.

However, we note that the Exhaustive Generation Method is so inefficient that it is impossible to adopt to answer an online query which is expected to run in a fraction of a second. The reason is as follows. The original partial trajectory needs to be decomposed into a list of sub-trajectories up to a certain extent (e.g., restrictions can be imposed such as setting the maximum number of deleted nodes to be three, and the number of sub-trajectories is $\sum_{i=r-3}^{r} C_i^r$). Based on the theories of mathematical combination, the complexity of this process is factorial. It is precisely the reason that the Exhaustive Generation Method is inefficient. Its running time is measured and presented in the experiment (Section V). Hence a revised method (presented below in Section IV-B) is introduced which does not suffer from this issue and is efficient in terms of running time as the experiment will show.

B. End-Points Generation Method

We start by presenting the theoretical findings underneath the *End-Points Generation Method* which extensively reduces the number of sub-trajectories that we need to examine, hence significantly reduces the running time of processing online queries.

Theorem 1: Using a first order Markov model based SubSyn algorithm and given a partial trajectory, the probability of any potential destination depends only on the starting node n_s and the current (i.e., most recent) node n_c of this partial trajectory.

Proof: Given a partial trajectory T^p, we combine the destination probability equation (1) and the posterior probability equation (9) to obtain the following equation:

$$P(d \in n_j|T^p) = \frac{P(T^p) \cdot \frac{p_{c\to j}}{p_{s\to j}} \cdot P(d \in n_j)}{\sum_{k=1}^{g^2} \left[P(T^p) \cdot \frac{p_{c\to k}}{p_{s\to k}} \cdot P(d \in n_k)\right]} = \frac{\frac{p_{c\to j}}{p_{s\to j}} \cdot P(d \in n_j)}{\sum_{k=1}^{g^2} \left[\frac{p_{c\to k}}{p_{s\to k}} \cdot P(d \in n_k)\right]}. \quad (10)$$

Equation 10 shows that, besides n_j, the values of $p_{c\to j}$ and $p_{c\to k}$ only depend on the current node n_c while the values of $p_{s\to j}$ and $p_{s\to k}$ only depend on the starting node n_s. Therefore the value of $P(d \in n_j|T^p)$ only depends on the starting node and the current node of the given partial trajectory. □

The above theorem indicates that, given a first order Markov model and a partial trajectory, the destination probability and the destination rank of the user's private destination can be altered by only deleting the two *end points* (i.e., n_s and n_c) from this partial trajectory. Hence we call this method the End-Points Generation Method. This way we decrease the computational cost significantly.

V. Experimental Study

In this section, we conduct an extensive experimental study to evaluate the performance of our SubSyn algorithm. The only available algorithm that can perform generic destination prediction is the ZMDB algorithm described in Section II, and we need to adapt it in the following way in order to make the comparison. The original ZMDB algorithm can only give suggestions (i.e., predicted destinations) provided that a query trajectory has a partial match in the training dataset. Consequently it can not be compared with our algorithm when non-matching query trajectories[4] are present. An adapted version of ZMDB algorithm has been implemented such that the current node n_c in the query trajectory is used as a predicted destination in the case where insufficient predicted destinations are generated by ZMDB algorithm. Same implementation is done for SubSyn. In this section, we call this adapted ZMDB algorithm the *baseline algorithm*.

[4]non-matching query trajectories are those query trajectories which have no partial match in the training dataset.

The effectiveness subsection (Section V-B) will focus on the prediction accuracy while the efficiency subsection (Section V-C) measures and presents the running time of both training and prediction stages. The runtime efficiency of the locational privacy protection methods proposed are also tested and compared with each other. The settings of our experimental study are presented in Section V-A.

A. Dataset

We use a real-world large scale taxi trajectory dataset from the *T-drive* project [28, 29] in our experiments. It contains a total of 580,000 taxi trajectories in the city of Beijing, 5 million kilometres of distance travelled, and 20 million GPS data points. The GPS points are plotted in Fig. 3. We randomly pick 1,000 trajectories from this dataset to be the query trajectories and the remaining trajectories are used as training data.

B. Evaluation of Effectiveness

Evaluation Measures: To evaluate the performance of our system on various user queries, we use the following two means of measurement: *Coverage* and *Prediction Error*. The former counts the number of query trajectories for which at least k suggested destinations are provided. The parameter k is determined by the number of predicted destinations that we set. For instance, when we examine top three predicted destinations, k is set to three. In other words, due to the problem of data sparsity presented, it is highly likely that insufficient predicted destinations will be suggested for certain non-matching query trajectories. Hence we utilise this property to demonstrate the difference in robustness between the baseline algorithm and SubSyn. The prediction error for a single predicted destination of a query trajectory is the ℓ_1 distance between this predicted destination and the true destination of the query trajectory. The aggregated *Prediction Error* is the average of all distance deviations across each predicted destinations of all query trajectories. It is used to indicate how far the prediction results deviate from the true destinations. It should be made clear that the prediction error does not indicate the best prediction accuracy that an algorithm can achieve. For instance, a prediction error of 2km for the top three predicted destinations is the averaged distance deviation of all of these three predicted destinations, and it is likely that the true destination is amongst these three predicted destinations. Better algorithm has a higher coverage and a lower prediction error (i.e., lower average distance deviation).

The two aforementioned means of measurement will be evaluated against varying four parameters one at a time: Firstly we will vary the *grid granularity* g (20-50 with 10 units increment) to select a best grid granularity for our training dataset. This chosen grid granularity will be used for the remainder of the experiment. The second and third parameters are the *trip completed percentage* (10%-90% with 20% increment) and the *top-k predicted destinations* (1-5 with 1 unit increment). Finally, instead of randomly selecting query trajectories from the training dataset, we manually mix the proportion of matching and non-matching query trajectories and vary the *Match Ratio* (denoted by τ) which is the proportion of matching query trajectories in the test dataset (0-1 with 0.25 increment).

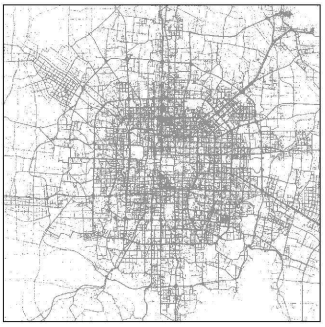


Fig. 3. Training dataset: 20 million taxi GPS points in Beijing

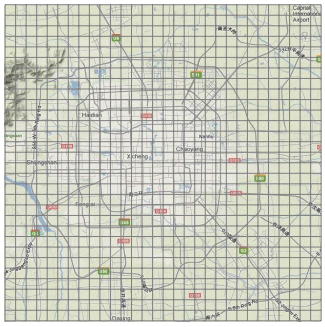
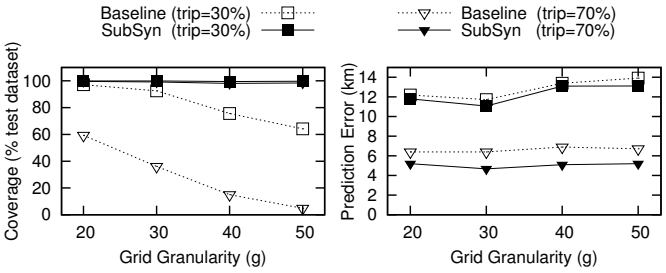


Fig. 4. Map of Beijing with a 30×30 grid graph overlay

In the following experimental result figures, a convention is set that *dashed lines* and *hollow shapes* are used to represent the baseline algorithm, and *solid lines* and *filled shapes* are for our SubSyn algorithm.

Varying the grid granularity: First of all, a suitable grid granularity needs to be decided for our training dataset. On one hand, a coarse grid (e.g., 20×20) may have a very low prediction accuracy because the area covered by each grid node is too large. On the other hand, it has the benefit that the number of matching query trajectories is much higher since more trajectories in the training dataset may fall into identical nodes, hence increasing prediction accuracy. A fine grid (e.g., 50×50) has the advantage of higher prediction accuracy that the small node area brings, but training data become even sparser because less locations will lie in a same node, making the task of destination prediction more difficult. Fine grid also has a drawback that it requires (much) more time to complete the offline training stage. Therefore, we need to find a balanced and compromised grid granularity that is neither too small nor too large, and can achieve the best prediction accuracy.



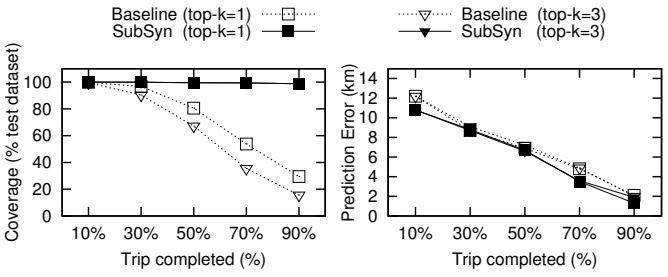
(a) Coverage (b) Prediction Error

Fig. 5. Varying the grid granularity g

Fig. 5 shows the trends in both coverage and prediction error with respect to grid granularity. The coverage of the baseline algorithm drops rapidly due to the data sparsity problem caused by smaller nodes in a fine grid, but the drop in coverage of SubSyn-Prediction is extremely small. The optimal grid granularity for our training dataset is selected to be 30 according to the global minimum point in Fig. 5b. In a typical setting where $g = 30$ and trip completed percentage is

70%, the coverage of SubSyn-Prediction algorithm is roughly three times the coverage of the baseline algorithm while having a more than 1km reduction in prediction error. Although the optimal value of g is dependent on different training dataset, it does not need to be modified often because the update in training dataset is rare (e.g., one update every few months), and for a satisfactory prediction accuracy, it is not essential to modify g for each training dataset update. All following experiments are done using the grid granularity $g = 30$ (cf. Fig. 4).

Varying the percentage of trip completed: Fig. 6 shows the effectiveness performance versus the percentage of trip completed for both top-k values 1 and 3. For the baseline algorithm, the amount of query trajectories for which sufficient predicted destinations are provided decreases as the length of the trip increases due to the fact that longer query trajectories (i.e., higher trip completed percentage) are less likely to have a partial match in the training dataset. Specifically, when trip completed percentage increases towards 90%, the coverage of the baseline algorithm decreases to almost 0%. Our SubSyn-Prediction algorithm successfully coped with it as expected with only an unnoticeable drop in coverage, and can constantly answer almost 100% of query trajectories. It proves that the baseline algorithm cannot handle (relatively) long trajectories since the chances of finding a matching trajectory decrease when the length of a query trajectory grows. The coverage performance of the baseline algorithm when top-$k = 3$ is even worse then that of top-$k = 1$ because the metric *coverage* counts the number of query trajectories that gives k predicted destinations. Therefore the number of query trajectories for which SubSyn-Prediction gives three suggestions is clearly less than those which can provide only one suggested destination.

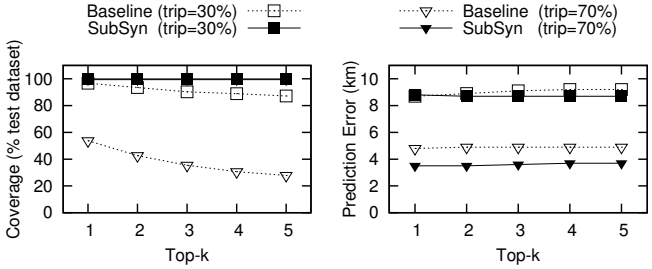


(a) Coverage (b) Prediction Error

Fig. 6. Varying the percentage of trip completed

Apart from the huge advantage of SubSyn-Prediction in coverage, its prediction error is comparable with that of the baseline algorithm. For the baseline algorithm, despite the negative influence of the coverage problem, its prediction error reduces as the trip completed percentage increases for a simple reason. When the baseline algorithm fails to find adequate predicted destinations, we use the current node in the query trajectory as the predicted destination. Because higher trip completed percentage yields a closer distance between the current node and the true destination, the prediction error reduces accordingly. For SubSyn-Prediction, closer to the true destination means that there are fewer potential destinations and intuitively the prediction error reduces. It is observed that SubSyn-Prediction outperforms the baseline algorithm throughout the progress of a trip.

Varying the number of predicted destinations: We also investigate the effect of the number of predicted destinations on the performance of both algorithms by examining the top-k (from 1 to 5) predicted destinations. We are interested in this metric since it reveals more vulnerability of the baseline algorithm in that although it can make prediction for matching trajectories, the number of predicted destinations may still be insufficient (e.g., only one). Therefore in such circumstances where insufficient predicted destinations are returned, we consider them unsatisfactory in the coverage test. The



(a) Coverage (b) Prediction Error

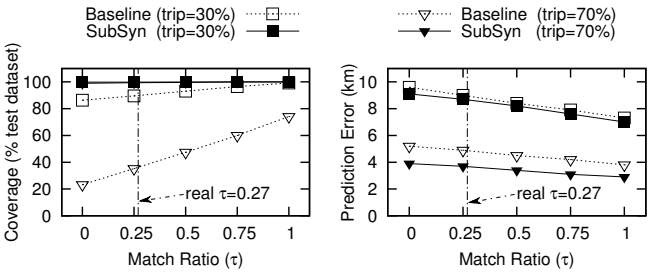
Fig. 7. Varying the value of k

experimental results are shown in Fig.7. In this figure, the comparative performances of both algorithms are similar to that of the experiment of varying the percentage of trip completed. Specifically, observations which can be made from the figure are as follows. The SubSyn-Prediction algorithm shows a more stable coverage and a more accurate prediction accuracy than the baseline algorithm. For the baseline algorithm, the number of query trajectories which have sufficient suggestions (i.e., the coverage) drops due to the data sparsity problem since, for certain query trajectories, it cannot find adequate (i.e., no less than k) predicted destinations. The same problem does not affect SubSyn-Prediction and it remains an almost 100% suggestion offer rate. In a typical setting when $k = 3$, the coverage of SubSyn-Prediction is almost three times the coverage of the baseline algorithm.

Varying top-k has little correlation with the prediction error because we compute the prediction error (i.e., average distance deviation) by averaging amongst all predicted destinations.

Varying the ratio of matching and non-matching query trajectories: The query trajectories used in the above experiments are drawn randomly from the training dataset. They reflect the real distribution of matching and non-matching query trajectories in both the test dataset and the training dataset. It is found that the real match ratio (denoted by τ) decreases while the grid granularity g increases because finer grid yields sparser data. For a 30×30 grid graph, the averaged real match ratio is found to be approximately 0.27 (indicated by the vertical dashed line in Fig. 8). It indicates that, in average, only 27% of query trajectories will be able to find a partial match in the training dataset. A low match ratio hence is fatal to the original ZMDB (and the baseline) algorithm, but has little negative impact on SubSyn-Prediction.

In this experiment, we elaborate further on the concept of match ratio by manually selecting a mixture of matching and non-matching query trajectories, and comparing the influence of different match ratios. For simplicity while maintaining an indicative results, a trajectory is said to have a partial match if the first 70% nodes have an exact match in the training dataset. This indicates that, when $\tau = 0.27$ and the trip completed percent is higher than 70%, the coverage is at most 27% for the baseline algorithm.



(a) Coverage (b) Prediction Error

Fig. 8. Varying the match ratio τ

As shown in Fig. 8, by varying the match ratio τ, the performance of the baseline algorithm deteriorates rapidly when τ is tuned towards zero while little impact is observed for SubSyn-Prediction. The baseline algorithm functions well provided that abundant data are given (i.e., $\tau \rightarrow 1$), but the performance starts to decrease to an unacceptable status when there are insufficient training data. Particularly, when the match ratio is low (i.e., $\tau \rightarrow 0$) and the trip completed percentage is high (e.g., 70%), the baseline algorithm has a coverage towards 0%. From Fig. 8a, our SubSyn-Prediction algorithm provides adequate (i.e., at least three since the default value of top-k is 3) predicted destinations for almost every query trajectory. It proves that our algorithm can overcome the data sparsity problem while maintaining a stable performance, whereas the baseline algorithm is unable to achieve this objective.

In Fig. 8b, it is observed that the prediction errors of both algorithms drop when more relevant training data are available (i.e., $\tau \rightarrow 1$). Once again, it proves that the prediction accuracy of SubSyn-Prediction leads the baseline algorithm.

C. Evaluation of Efficiency

Apart from the prediction accuracy, runtime efficiency is as important since the algorithms need to be evoked to answer real-time queries. For each user supplied query, it must report a list of predicted destinations instantaneously. Otherwise the whole purpose of the solution is meaningless. This section verifies the swiftness of SubSyn-Prediction which outperforms the baseline algorithm by at least two orders of magnitude in most cases. Result is presented with respect to varying both trip completed percentage and match ratio. The reason for not including the parameter top-k is that all potential nodes are computed for a destination probability before selecting the first k destinations to report. Therefore the value of k does not affect the computation process at all. The running time of SubSyn-Training is also presented for completeness. In the context of privacy protection, the two methods Exhaustive Generation Method and End-Points Generation Method are compared for running time with respect to the number of nodes in query trajectories, which is directly related to the number of to-be-published locations. It is worth mentioning that this part of the experiment was run on a commodity computer with Intel i7-860 CPU (2.8GHz) and 4GB RAM.

TABLE II
AVERAGE RUNNING TIME OF SUBSYN-TRAINING ALGORITHM

Grid Granularity	20	30	40	50
Running Time (h:mm:ss)	0:02:35	0:32:35	3:08:53	17:13:55

Runtime efficiency of SubSyn-Training algorithm: In Algorithm 1, we introduced the SubSyn-Training algorithm which synthesises sub-trajectories in the training dataset and generates matrices M and M^T to be used in the online query stage. Especially, several enhancements were implemented to increase the runtime efficiency of the time-consuming large matrix multiplications. Our experiments have proven that the running time of SubSyn-Training is totally acceptable. Table II summarises the time taken in the training stage with respect to various grid granularities. In a 30×30 grid graph, the average running time of SubSyn-Training is approximately 30 minutes which is negligible, especially when the training stage is only run occasionally (e.g., once in a few months). Even when the grid granularity reaches 50, the training stage can still be completed within 18 hours on a commodity computer with ordinary hardware configuration.

Runtime efficiency of SubSyn-Prediction algorithm: We compare the runtime performance of our SubSyn-Prediction algorithm with the baseline algorithm in terms of online query response time. Due to the information stored in the training stage, SubSyn-Prediction requires little extra computation when answering a user's query. As Fig. 9 shows, the baseline algorithm requires too much time to run, whereas SubSyn-Prediction algorithm is at least two orders of magnitude better constantly. The reason is that the baseline algorithm is forced to make a full sequential scan of the entire training dataset in order to compute the posterior probability, whereas SubSyn-Prediction can fetch most probability values directly from the stored matrices M and M^T. It is worth mentioning that varying either parameter (i.e., trip completed percentage or match ratio) has little influence on the running time of the two algorithms. These parameters mainly affect the effectiveness of the prediction process rather than the runtime efficiency.

Runtime efficiency of privacy protection methods: We introduced two methods for privacy protection in Section IV, namely the Exhaustive Generation Method and the End-Points Generation Method. Experiment was conducted to compare the runtime efficiency of both methods by varying the number of nodes in query trajectories.

The result presented in Fig. 10 clearly shows the gap in running time even when illustrated in a log-scale plot. The simple reason is that the Exhaustive Generation Method suffers from generating excessive number of sub-trajectories, especially when the number of locations, and hence the number of nodes, in a query trajectory is large since more nodes in a

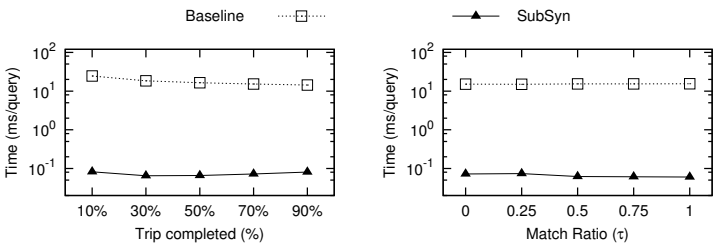


Fig. 9. Running Time (Destination Prediction)

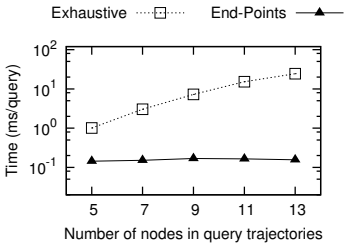


Fig. 10. Running Time (Privacy Protection)

query trajectory generates exponentially more sub-trajectories. While the number of nodes in query trajectories increases, the rise in running time of the End-Points Generation Method is infinitesimally small. It is demonstrated from this performance study that, similar to the runtime efficiency of the two destination prediction algorithms, the running time of End-Points Generation Method is at least two orders of magnitude faster than that of the Exhaustive Generation Method.

VI. Conclusion

In this paper, we have identified the data sparsity problem in destination prediction and proposed a novel *Sub-Trajectory Synthesis* (SubSyn) algorithm to address this problem. SubSyn algorithm decomposes historical trajectories into sub-trajectories and connect them into "synthesised" trajectories for destination prediction. This process is formulated based on a Markov model. The number of query trajectories that can have predicted destinations is exponentially increased by this means. Experiments based on real datasets have shown that SubSyn algorithm can predict destinations for up to ten times more query trajectories than the baseline algorithm. SubSyn algorithm consistently predicts destinations for all the query trajectories in all the experiments we have performed, and it have successfully addressed the data sparsity problem. At the same time, the SubSyn prediction algorithm runs over two orders of magnitude faster than the baseline algorithm.

We have also taken into account the privacy protection issue in case an adversary uses SubSyn algorithm to derive sensitive location information of users. We proposed an efficient algorithm for selecting the smallest number of locations a user has to hide on her trajectory in order to avoid privacy leak. Compared with a naive algorithm, our proposed algorithm is more than two orders of magnitude faster.

Acknowledgment

This work is supported by *Australian Research Council (ARC) Future Fellowships* Project FT120100832. Conference expenses are partially supported by *Google Ph.D. Travel Prize*.

Bibliography

[1] O. Abul, F. Bonchi, and M. Nanni, "Never walk alone: Uncertainty for anonymity in moving objects databases," in *Proc. of ICDE*, 2008.

[2] J. A. Alvarez-Garcia, J. A. Ortega, L. Gonzalez-Abril, and F. Velasco, "Trip destination prediction based on past GPS log using a hidden markov model," *Expert Systems with Applications: An International Journal*, vol. 37, pp. 8166–8171, 2010.

[3] D. Ashbrook and T. Starner, "Using GPS to learn significant locations and predict movement across multiple users," *Personal Ubiquitous Computing*, vol. 7, pp. 275–286, 2003.

[4] Auto4Sq. (2012) Schedule automatic foursquare checkin. [Online]. Available: http://www.auto4sq.com/

[5] A. Bhattacharya and S. K. Das, "Lezi-update: An information-theoretic approach to track mobile users in pcs networks," in *Proc. MobiCom*, 1999, pp. 1–12.

[6] L. Chen, M. Lv, and G. Chen, "A system for destination and future route prediction based on trajectory mining," *Pervasive and Mobile Computing*, vol. 6, pp. 657–676, 2010.

[7] D'Keesto. (2012) Fast facebook checkin. [Online]. Available: https://play.google.com/store/apps/details?id=com.dkeesto.fbcheckin

[8] G. Gidofalvi, X. Huang, and T. B. Pedersen, "Privacy-preserving data mining on moving object trajectories," in *Proc. of MDM*, 2007.

[9] V. Gogate, R. Dechter, and B. Bidyuk, "Modeling transportation routines using hybrid dynamic mixed networks," in *Proc. UAI*, 2005.

[10] GPSExchange. (2012) GPS track log route exchange forum. [Online]. Available: http://www.gpsexchange.com/

[11] T. Hashem, L. Kulik, and R. Zhang, "Privacy preserving group nearest neighbor queries," in *EDBT*, 2010, pp. 489–500.

[12] T. Hashem, L. Kulik, and R. Zhang, "Countering overlapping rectangle privacy attack for moving knn queries," in *Information Systems*, 2012.

[13] E. Horvitz and J. Krumm, "Some help on the way: opportunistic routing under uncertainty," in *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ser. UbiComp '12, 2012, pp. 371–380.

[14] J. Krumm and E. Horvitz, "Predestination: Inferring destinations from partial trajectories," in *Proc. UbiComp*, 2006, pp. 243–260.

[15] J. Krumm and E. Horvitz, "Predestination: Where do you want to go today?" *IEEE Computer*, pp. 105–107, 2007.

[16] L. Liao, D. J. Patterson, D. Fox, and H. Kautz, "Learning and inferring transportation routines," *Artificial Intelligence*, 2007.

[17] N. Marmasse and C. Schmandt, "A user-centered location model," *Personal and Ubiquitous Computing*, vol. 6, pp. 318–321, 2002.

[18] M. E. Nergiz, M. Atzori, and Y. Saygin, "Towards trajectory anonymization: a generalization-based approach," in *Proc. of GIS*, 2008.

[19] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "Analysis and evaluation of V*-*k*nn: an efficient algorithm for moving *k*nn queries," *VLDB J.*, vol. 19, no. 3, pp. 307–332, 2010.

[20] D. J. Patterson, L. Liao, D. Fox, and H. Kautz, "Inferring high-level behavior from low-level sensors," in *Proc. UbiComp*, 2003, pp. 73–89.

[21] M. Research. (2012) T-drive trajectory data sample. [Online]. Available: http://research.microsoft.com/apps/pubs/?id=152883

[22] ShareMyRoute. (2012) Share my route. [Online]. Available: http://www.sharemyroutes.com

[23] R. Simmons, B. Browning, Y. Zhang, and V. Sadekar, "Learning to predict driver route and destination intent," in *ITSC*, 2006, pp. 127–132.

[24] M. Terrovitis and N. Mamoulis, "Privacy preservation in the publication of trajectories," in *Proc. of MDM*, 2008.

[25] D. Tiesyte and C. S. Jensen, "Similarity-based prediction of travel times for vehicles traveling on known routes," in *GIS*, 2008, pp. 14:1–14:10.

[26] V. V. Williams, "Breaking the Coppersmith-Winograd barrier," 2011.

[27] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *Proc. KDD*, 2012.

[28] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. KDD*, 2011, pp. 316–324.

[29] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang, "T-drive: Driving directions based on taxi trajectories," in *Proc. GIS*, 2010, pp. 99–108.

[30] Y. Zheng and X. Zhou, Eds., *Computing with Spatial Trajectories*. Springer, 2011.

[31] B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell, "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior," in *Proc. UbiComp*, 2008, pp. 322–331.