Forecasting Gathering Events through Continuous Destination Prediction on Big Trajectory Data

Amin Vahedian, Xun Zhou, Ling Tong
The University of Iowa
{amin-vahediankhezerlou,xun-zhou, ling-tong}@uiowa.edu

Yanhua Li
Worcester Polytechnic Institute
yli15@wpi.edu

Jun Luo
SIAT, Chinese Academy of Sciences
jun.luo@siat.ac.cn

ABSTRACT

Urban gathering events such as social protests, sport games, and traffic congestion bring significant challenges to urban management. Identifying gathering events timely is thus an important problem for city administrators and stakeholders. Previous techniques on gathering event detection are mostly descriptive, i.e., using real-time on-site observations (e.g., taxi drop-offs, traffic volume) to detect the gathering events that have already emerged. In this paper we propose a predictive approach to identify future gathering events through destination prediction of incomplete trajectories. Our approach consists of two parts, i.e., destination prediction and event forecasting. For the destination prediction part, we relax the Markov property assumed in most of the related work and address the consequent high-memory-cost challenge by proposing a novel Via Location Grouping (VIGO) approach for destination prediction. For the event forecasting part, we design an online prediction mechanism that learns from both historical and recent trajectories to address the non-stationarity of urban trip patterns. Gathering events are predicted based on projected arrivals in each location and time. A case study on real taxi data in Shenzhen, China shows that our proposed approach can correctly and timely predict gathering events. Extensive experiments show that the proposed VIGO approach achieves higher accuracy than related work for destination prediction and has orders of magnitude less memory cost than a baseline approach. The event forecasting based on VIGO is effective and fast enough for continuous event forecasting.

CCS CONCEPTS

• **Information systems → Geographic information systems**;

KEYWORDS

Gathering Events, Destination Prediction, Trajectory Mining

ACM Reference format:
Amin Vahedian, Xun Zhou, Ling Tong, Yanhua Li, and Jun Luo. 2017. Forecasting Gathering Events through Continuous Destination Prediction on Big Trajectory Data. In *Proceedings of ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Redono Beach, California, USA, November 2017 (SIGSPATIAL'17),* 10 pages.
https://doi.org/10.475/123_4

SIGSPATIAL'17, November 2017, Redono Beach, California, USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

Gathering events is the scenario where an unexpectedly-large number of moving objects (pedestrians, vehicle, etc.) arrive at the same region during a short period of time. Gathering events in urban areas pose serious challenges for city management as more-than-ordinary resources will be required and public safety concerns will be raised. Example consequences may include traffic jams and high risk of injury, crimes, and terror-attacks. Shanghai's 2014 new year's eve stampede is a tragic example [20]. If given timely warning of future gathering events, city officials will have the opportunity to react to these situations in a timely manner, e.g., re-routing usual traffic, adopting necessary provisions, etc.

State-of-the-art techniques on urban event detection are mostly descriptive, i.e., the region and time of events are detected based on available on-site observations such as taxi drop-offs or traffic volume around the venue. These methods lack the ability to forecast future events before the gathering becomes significant.

In this paper we investigate a gathering event forecasting approach through trajectory destination prediction. The approach works in two steps. First, a spatio-temporal destination prediction model is learned from historical trajectories of moving objects (e.g., taxis). Second, we use this model to continuously predict the destination and arrival time distribution of incomplete trajectories, and identify future spatio-temporal regions with high projected arrivals as gathering events. To our best knowledge, this is the first work on gathering event forecasting through trajectory prediction.

Designing the aforementioned approach for event forecasting is very challenging. First of all, a common way in the literature for trajectory destination prediction is to assume Markov property for the trips [12, 22, 23]. This assumption is unrealistic and adversely affects the prediction accuracy. Relaxing this assumption, however, result in prohibitive memory cost due to the huge number of source, via location and destination combinations. Second, urban trips tend to deviate from historical distribution in case of rare gathering events. A global destination prediction model generates poor results in such cases and affect event forecasting effectiveness.

To address these challenges, this paper proposes two novel techniques. First, we relax the Markov assumption and predict the destination of a trajectory based on the source and the current location. To address the memory cost challenge, we design a scalable Via Location Grouping (VIGO) approach to effectively reduce memory cost. The VIGO approach reduces the memory cost of a baseline approach by 80%. Second, we propose an online learning mechanism to address the challenge posed by temporal non-stationarity of the trips. We perform a case study on real taxi trajectory data to demonstrate the effectiveness of the proposed solutions. We show through experiments that the events would not have been

predicted without the proposed online mechanism. Also, the proposed event forecasting algorithm takes less than 0.4 milliseconds per trip, which makes continuous online event forecasting feasible.

Specifically our main contributions in this paper are summarized as follows:

- We design a novel Via Location Grouping (VIGO) algorithm to learn destination probabilities with efficient memory cost, while relaxing the Markov property assumption.

- We design an online learning mechanism using VIGO to address the temporal non-stationarity of the trips and to do real-time gathering event forecasting.

- We do a case study using a real taxi trajectory dataset to validate the effectiveness of the proposed approach for event forecasting.

- Extensive experimental evaluations using real data demonstrate the time and memory efficiency of the proposed solutions.

The reminder of the paper is organized as follows. In Section 2, we discuss the related work. In Section 3 the destination prediction and event forecasting problems are formulated as computational problems. Sections 4 and 5 discuss the proposed solutions. Section 6 presents our evaluation and section 7 concludes the paper.

2 RELATED WORK

To the best of our knowledge, destination prediction has not been used in the context of event forecasting before. The works of Martin Kulldorff and Neill [8–11] and other recent works on event detection [5, 13] are based on *detecting* events using already observed counts at locations. The works of Zhou et al. and Vahedian et al. [18, 26] are based on real-time monitoring of significantly high flows in space and are categorized as early-detection and not forecasting. In this paper, we use destination prediction of incomplete trips to predict the number of arrivals at each location *ahead of the time*, which enables us to forecast unexpected events at a future time.

The literature of **destination prediction** problem can be organized into two broad categories based on the data used: (1) using context-related and personal trip data and (2) only using anonymous trip data (e.g., no traveler information). In the first category, personalized trip data is one type of context-related data used by researchers to predict the destinations of incomplete trips of re-identifiable individuals [3, 4, 6, 17, 23, 25]. Krumm et al. and Yamaguchi et al. [7, 24] used spatial region data and personalized trips to predict the destinations of incomplete trips. Alvarez et al. [2] use similar information to do the predictions by proposing a novel method to partition the space. Xue et al. [23] use social network related data to predict destinations of individuals by developing a set of learning trajectories for groups of people who share network or behavioral similarities. However, in this paper, we only use the anonymous trajectory data to approach this problem.

In the second category, an important challenge is the complex dependencies among the segments of of an urban trip. To address this challenge, most researchers have adopted a Markov model-based approach. In this approach, the trip is decomposed into a sequence of transitions between locations in space. These transitions are modeled by low-order Markov chains, hence facilitating the calculation of the probability of an incomplete trip. Xue et al. [21, 22] use this technique to calculate the destination probabilities using the Bayes rule. However, using a Markov model results in a memory-less model, meaning that future movements along a trip are independent of the past movements. However, this Markov property assumption is frequently violated in real-world scenarios and adversely affects the prediction accuracy. Li et al. [12] use a similar approach but distinguish between transitions among via points and transitions from via points to destinations in their calculations of the transition probabilities. This approach still does not properly address the independence issue in the Markov model-based approaches. Wang et al. [19] propose to condition the destination probability on the start location. They learn three transition probability matrices: source to destination, via points to destination and via point to via point. They also define a direction concept called Mobility Gradient for each sub-trajectory, which is used together with the three transition matrices to calculate the destination probabilities. However, in each of the transition matrices, the probabilities are calculated based on the assumption that they are independent of the other locations in the trip, which also imposes limitations on the accuracy of the predictions.

In this paper, we propose a destination prediction method that calculates the destination probabilities conditioned on all the three locations: source, current location and destination by proposing an efficient learning algorithm that allows real-time predictions with efficient use of memory.

3 OVERVIEW

The Gathering Event Forecasting through Trajectory Destination Prediction problem can be solved in two steps: (1) Build a model for trajectory destination prediction. (2) Use the model to predict the destination probability distributions of incomplete trajectories and identify potential events based on the predicted arrivals in every location and time slot. In this section we define basic concepts and formulate these two steps as two sub-problems. Then we discuss the challenges in solving these problems as well as an overview of our proposed solution.

3.1 Concepts and Definitions

A **spatial field** $\mathcal{S}$ is a two-dimensional geographical region partitioned into a grid. Grid cells $\mathbf{l_1}, \mathbf{l_2}, ..., \mathbf{l_n}$ represent distinct locations in the field. Given a spatial field, the location of any moving object at a certain time can be mapped to a grid cell. For example, locations of a moving taxi in a trip can be represented by a sequence of grid cells, paired with the corresponding time.

Definition 3.1. A **trajectory** $Y = \{(s, t_s), (v_1, t_{v_1}), (v_2, t_{v_2}), ..., (v_n, t_{v_n}), (d, t_d) | s, v_i, d \in \mathcal{S}\}$ represents a trip of a moving object with a sequence of location and time pairs. s and t_s are the *source* location and start time of the trip, while d and t_d are the *destination* location and arrival time of the trip. The locations of the rest of the points v_i in the trajectory are called *via locations* of Y.

Taxi trips are examples of trajectories. The pick-up location is the source of the trajectory and the drop-off location is the destination. Based on definition 3.1, we define sub-trajectories to represent the incomplete trips.

Definition 3.2. A **sub-trajectory** $Y_c = \{(s, t_s), (v_1, t_1), ..., (c, t_c)\}$ is the first few elements of trajectory Y, where $c \in \{v_i\}$.

The first record of a sub-trajectory still represents the source of the trip but the last element of a sub-trajectory is a via-point instead of the destination of the trip.

Definition 3.3. A **spatio-temporal region** $R = (S_R, T_R)$ is a pair of spatial region S_R and a time window T_R, where S_R is a rectangular sub-region of the spatial field S.

In this paper we follow the definition of events in prior work [26]. For a spatio-temporal region R, we calculate the average number of trips ending in S_R during the same time of day as T_R, denoted as B_R or the baseline. The predicted number of trips ending in S_R during time T_R is denoted as C_R. We employ a Expectation-Based Poisson Model [14] to calculate the log likelihood ratio between the hypothesis that there will be an elevation of arrivals in R versus the hypothesis that the predicted arrival count is normal. Specifically, the log-likelihood ratio is calculated as follows:

$$LLR(R) = \begin{cases} C_R \log \frac{C_R}{B_R} + (B_R - C_R) & \text{if } C_R \geq B_R \\ 0 & otherwise \end{cases} \quad (1)$$

As proved in a prior paper [26], the $LLR(R)$ score is statistically significant at α level if 1-$Pr(X < C_R) \leq \alpha$, where $X \sim Po(B_R)$.

Definition 3.4. A **gathering event** is a spatio-temporal region R such that $LLR(R)$ is statistically significant at α level.

3.2 Problem Statements

We formulate the two steps of our approach into two sub-problems, namely, the destination prediction problem and the event forecasting problem.

Sub-Problem 1: Destination Prediction. *Given:* a spatial field $\mathcal{S}$, a set of historical trajectories X, and a sub-trajectory Y_c, *Find:* the probability of each location in $\mathcal{S}$ to be the destination of Y_c as well as the probability distribution of the arrival time.

Sub-Problem 2: Event Forecasting. *Given:* a spatial field S, a set of historical trajectories X_h, a list of sub-trajectory U at current time, a target time-window t, and statistical significance threshold α, *Find:* Top-k gathering events at time t with the highest LLR scores. The *Objectives* of both sub-problems are to reduce computation cost while improving the accuracy of results.

3.3 Challenges and Solution Overview

Two challenges arise when designing the computational solutions to our proposed problem. We illustrate them with examples to motivate our solutions.

First, it is challenging to handle the trade-off between destination prediction accuracy and computational cost. Prior research have assumed the urban trips have low-order Markov property [22], i.e. the movement at each stage of the trip is only dependent on the current location but independent of previous steps. This assumption, although helpful in reducing computational cost, is unrealistic and limiting, and might lead to lower accuracy. Figure 1 shows a counterexample of this assumption. A quiet two-way street (light-shaded, top-down) overpasses a busy one-way express way (darkly shaded, left to right) at c, where the traffic volume on the latter is 9 times of that on the former. A moving object started a trip at location s and currently at location c will be predicted to go right with 90% probability, if assuming first-order Markov property. However, considering the source of the trip, the probability of moving downwards (100%) is much higher than moving to the right (0%).

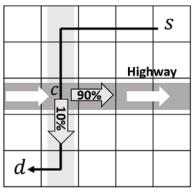


Figure 1: Example of an error caused by Markov property assumption

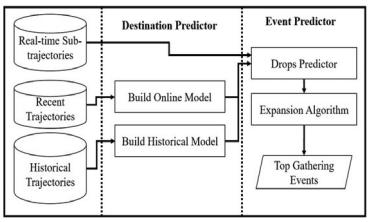


Figure 2: High-level workflow of the event forecasting framework.

In our approach, we relax the Markov assumption to predict the destination of a moving object based on the current location **and** the start location. However, doing so significantly increases the memory cost required to store all the combinations of source, via location, and destination. To address this challenge, we propose a Via Location Group (VIGO) approach that combines via locations with the same destination distributions to effectively reduce memory cost.

Another challenge is the temporal non-stationarity of urban trip patterns. Popular destinations of trips from the same source might deviate significantly from the overall historical statistics in case of rare events. For example, generally taxis taking passengers from a hotel zone are more likely to end up at the airport. However, during a big event such as a sports game or a concert, the most likely destination might be a stadium instead. This phenomenon is particularly challenging to handle for gathering event forecasting since a global destination prediction model might not be able to accurately predict destinations for trips going to these events.

We address this challenge by proposing an online prediction mechanism for destination prediction. A historical model learned from all the historical trajectories is combined with an online model learned from recent trajectories to improve the event forecasting effectiveness.

Figure 2 demonstrates the high-level overview of the solution framework proposed in this paper and how the two components, destination prediction and event forecasting, interact with each other. A destination predictor is built using complete trajectories, then the event forecaster takes real-time sub-trajectories as input and uses the output of the destination predictor to forecast top gathering events.

4 TRAJECTORY DESTINATION PREDICTION: COMPUTATIONAL SOLUTIONS

4.1 A Simple Classification Model

The destination prediction problem can be defined as a classification problem, i.e. every location is treated as a class. To classify each sub-trajectory, the Bayes classifier is commonly used to compute the probability of a location as the destination, conditioned on the observation of a sub-trajectory Y_c. Based on the definition of the

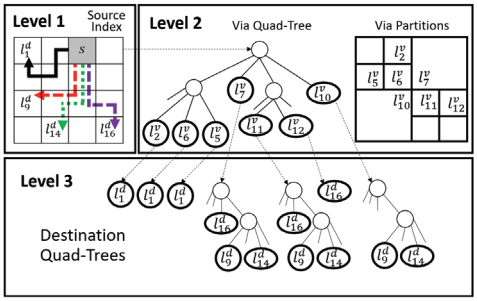


Figure 3: The Nested Quad-Tree Structure. l_i^v means a via location at l_i and l_j^d means a destination at l_j

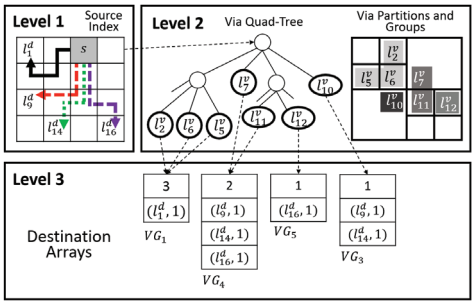


Figure 4: The proposed VIGO index structure.

conditional probability and Bayes' Theorem, we have:

$$p(d|Y_c) = \frac{p(d \cap Y_c)}{p(Y_c)} = \frac{p(Y_c|d) \times p(d)}{p(Y_c)} \quad (2)$$

In this paper, by probability of a location, we mean the probability of the moving object being at that location. Therefore, $p(d)$ in equation 2 means probability of being at d. The approach of equation 2 involves calculating $p(Y_c|d)$. Commonly, related work (e.g., [22]) solve the problem based on the Markov property assumption. That means, the probability of a sub-trajectory $p(Y_c)$ is the product of the probabilities of all the pair-wise transitions. As previously illustrated in Section 3.3, this assumption is not realistic and may give poor results particularly when predicting destinations of trajectories along unpopular routes.

To address this limitation, we relax the Markov property assumption by using the combination of source and current location (s, c) to replace the entire partial trajectory Y_c. This suggests that the destination is dependent on the combination of source and the current location. We argue that this is a realistic yet computation-friendly simplification of Equation 2, which achieves higher accuracy. We rewrite Equation 2 in the following way:

$$p(d|s \cap c) = \frac{p(s \cap c \cap d)}{p(s \cap c)} = \frac{dest(s, c, d)}{via(s, c)} \quad (3)$$

In Equation 3, s is the source of Y_c and c is its current location. $via(s, c)$ is the total number of trajectories with s as the source and c as a via location. $dest(s, c, d)$ is the total number of such trajectories that end at d in the data. A naive approach to learn the prediction model of Equation 3 is to store the counts $via(s, c)$ and $dest(s, c, d)$ of every combination of s, c and d in S. In such a case, if S is a 128×64 grid, we will need to store $(128 \times 64)^3 \approx 5.5 \times 10^{11}$ counts. With a 4-byte data-type, we will need 2 TB of memory to learn and apply the model of equation 3. Considering the hardware capabilities of an average machine, this approach is infeasible. To address this challenge, we propose a simple quad-tree based approach as a baseline solution.

4.2 Baseline: A Nested Quad-Tree Approach

The 3D array described above to store the destination counts is very sparse. Many (s, c, d) combinations do not exist in the real data. Thereby, we could simply use a spatial tree index structure to store the via and destination locations and the corresponding counts to save memory cost.

Here we propose a simple nested quad-tree (NesQ) as a baseline solution. NesQ consists of three levels, where the top level is a an index of each source location and the other levels both use a variation of the quad-tree discussed in [16]. Figure 3 shows an example of the NesQ data structure, in which four trajectories, starting from s, have been learned.

The first level is a two dimensional grid indexing all the possible sources. Each location s points to a via quad-tree at the second level, which stores the counts of via locations c for source s. Also stored in each leaf node is a pointer to a quad-tree in level three, which stores the counts of the destinations of trips that start from s and pass c. The counts stored in the second and third levels are used to calculate destination probabilities of sub-trajectories by using equation 3. These counts can be learned by going through all the complete trajectories once.

In Figure 3, one can observe that many destination quad-trees are identical, which could be stored only once. we use this observation to propose a novel and efficient model in the next section.

4.3 VIGO: A Scalable Via-Location Grouping Approach for Destination Prediction

The many identical destination quad-trees in Figure 3 suggest that many via locations of the same source share exactly the same destination distributions. This is particularly true for locations along major roads with high traffic volume. For instance, imagine a sequence of locations on a major expressway between two exits. These locations will for sure have the same destination probabilty distributions for a particular s. Based on the above observations, we propose a scalable Via-Location Grouping (VIGO) approach to efficiently reduce the memory cost of NesQ.

4.3.1 The VIGO Index Data Structure. First we introduce the concept of a "via group", which is a key idea in our proposed VIGO Index structure.

Definition 4.1. A **via group** of a source location s, denoted as $VG_i(s)$ is a set of via locations l_j^v where for every $l_j^v \in VG_i(s)$ we have $via(s, l_j^v) > 0$ and $\forall l_j^v, l_k^v \in VG_i(s)$ and $d \in S$, $dest(s, l_j^v, d) = dest(s, l_k^v, d)$. Each via location of the same source could belong to only one via group.

In Figure 3, the destination quad-trees of via nodes l_2^v, l_5^v, and l_6^v are exactly the same. They should form a via group of s. We only need to store one copy of the destination counts for the two of them. Our proposed VIGO Index Structure is based on the above definition. Figure 4 demonstrates the VIGO index for a particular

source location s. The structure for other sources are similar. The top two levels are similar to the NesQ structure. It has a source grid index, where each source location points to a quad-tree to store the counts of via locations. However, for each via quad-tree leaf node, we add a pointer to the destination list of a "via group". Each destination list has an array storing destinations and counts. Via locations within the same via group will have pointers to the same list, because they have the same destination count distribution. In level 2 of Figure 4 the partitioning of the via tree is shown on the right. Each shaded area in the partitions represents a unique via group. There are four via groups in total in this example.

For each destination list, we track the number of via locations in the group. For instance, . This count is followed by an array of destinations and their counts. For example, VG_4 has 2 via locations and it has 3 possible destinations: l_9^d, l_{14}^d, and l_{16}^d, all with count=1.

4.3.2 Learning of the VIGO Model. The above data structure can save quite much memory by reducing the number of destination counts stored. However, we also need to design an efficient and correct learning algorithm to build this model. To learn the VIGO model, we still read each historical trajectory at a time and scan all the points from the source to the destination.

Algorithm 1 shows the VIGO-Learner algorithm based on the proposed VIGO index structure. The input is a set of trajectories and the output is the updated VIGO Index. The algorithm takes one trajectory at a time, and updates the underlying VIGO index. For a trajectory Y with source s and destination d, we fetch the corresponding via quad-tree $Q_v(s)$. Then we scan each via location v in Y sequentially (Line 6) and update the VIGO index M according to the following rules.

(1) If a via location $l^v \in Y$ was not in Q_v, we insert it into Q_v and set $via(s, l^v) = 1$ (line 5-7).
(2) All the "new" via locations l_i^v along Y that are newly inserted into Q_v should be assigned to the same new group. When scanning the first such via location, we create a new group VG_new and add d as the only destination, with count = 1 (Line 8-10). For all the following new via locations, we assign all of them to this group (Line 11).
(3) All the "old" via locations l_j^v along Y that were already in Q_v also need to be moved to new groups, because a new destination d could potentially change the destination distribution of these via locations. Thus we create a new group to hold these via locations. Suppose l_j^v was in *old_group* before Y was learned (Line 15-16). If l_j^v is the first via location in *old_group* to be processed, then we create a new via group *new_group* by copying the destination array of *old_group*. If d is already in the array then we increment the count. Otherwise we append d at the end with count = 1. Then l_j^v's via group pointer will change to *new_group*(Line 17-19). To avoid creating multiple *new_group* for future via locations in *old_group*, we thereby put a pointer *map* in *old_group* to *new_group* to record this group transfer (Line 20). For future l_k^v in *old_group*, we use this pointer to find *new_group* and move l_k^v over (Line 22-23). Also, when a new trajectory comes in, all the group mapping information will be reset. We simply use another variable *cur_trj* in each destination array to make sure that old mappings are not used when learning the next trajectory (Line 21).

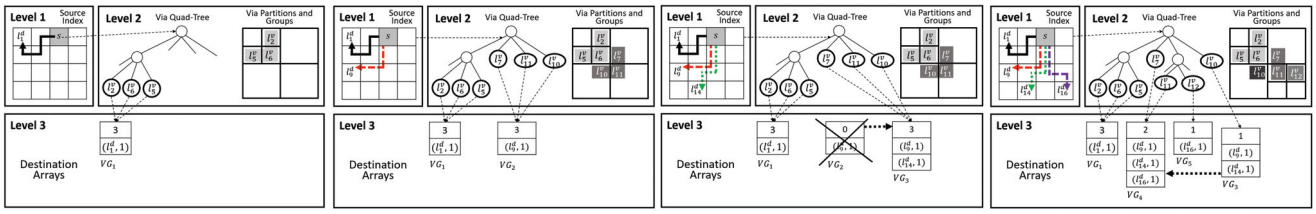
Algorithm 1: The *VIGO Learner* Procedure

Input: List of all trajectories (X)
Output: A VIGO Index (M)

```
1  cur_trj ← 0; M[] ←NULL
2  for each Y ∈ X do
3      Q_v ← M[Y.s]
4      for each via location v ∈ Y do
5          if v not in Q_v then
6              via_node ← Q_v.insert(v)
7              via_node.count = 1
8              if VG_new == NULL then
9                  VG_new ← Create new via group
10                 VG_new.dst_array[0]←(Y.d, 1)
11             via_node.group ← VG_new
12         else
13             via_node ← Q_v.get_node(v)
14             via_node.count + +
15             old_group ← via_node.group
16             new_group ← old_group.map
17             if new_group == NULL || old_group.trj ≠ cur_trj
               then
18                 new_group ← Create a copy of old_group
19                 new_group.increment_count(Y.d)
20                 old_group.map ← new_group
21                 old_group.trj ← cur_trj
22             via_node.group ← new_group
23             new_group.v_count + +
24             old_group.v_count − −
25             if (old_group.v_count) == 0 then
26                 Delete old_group
27     VG_new ← NULL; cur_trj + +
28 return M
```

(4) After each via location is moved to a new via group, we check the number of via locations remaining in the *old_group* and delete it if it is empty. This avoids unnecessary memory cost (Line 24-26).

An illustrative example: Figure 5 shows an example of how the trajectories are learned into the VIGO structure. First, $Y_1 = \{s, l_2^v, l_5^v, l_6^v, l_1^d\}$ is fed to the learner (Figure 5 (a)), via locations l_2^v, l_6^v and l_5^v do not exist in the via tree of s. Therefore, a new via group is created $VG_1(s) = \{l_2^v, l_5^v, l_6^v\}$, which has only one destination l_1^d. When $Y_2 = \{s, l_7^v, l_{11}^v, l_{10}^v, l_9^d\}$ is fed to the learner (Figure 5 (b)), same scenario happens and group $VG_2 = \{l_7^v, l_{10}^v, l_{11}^v\}$ is created. When $Y_3 = \{s, l_7^v, l_{11}^v, l_{10}^v, l_{14}^d\}$ is fed to the learner (Figure 5 (c)), it copies VG_2 to create VG_3 and appends l_{14}^d. Then VG_2 gets deleted because no via locations point to it anymore. When $Y_4 = \{s, l_7^v, l_{11}^v, l_{12}^v, l_{16}^d\}$ is fed to the learner (Figure 5 (d)), VG_3 is copied to create VG_4. Then l_{16}^d is appended to VG_4. When Y_4 visits l_{12}^v, similar to Y_1 and Y_2 where the via point was being visited for



(a) Learning Y_1 (b) Learning Y_2 (c) Learning Y_3 (d) Learning Y_4

Figure 5: An example of learning the VIGO structure.

the first time, a new group is created and l_{16}^d is added to it. However, VG_3 does not get deleted this time, because it has one remaining member l_{10}^v after losing l_7^v and l_{11}^v. Finally the VIGO index has four destination arrays, one for each via group.

We show, through the following lemma and theorem that the VIGO Learn algorithm can correctly build the VIGO Index with all the necessary counts.

Lemma 4.2. *Given a set of via locations $l_1^v, l_2^v, ...l_k^v$ that belong to the same via group $VG_i(s)$. If an incoming trajectory Y traverses all of them, then after Y is learned, these via locations will still belong to the same via group in VIGO.*

Proof. Since $l_1^v, l_2^v, ...l_k^v$ were in the same via group $VG_i(s)$, they share the same destination array $Dest(s, VG_i(s))$. Since each trajectory only has one unique destination, the new destination count distributions of v_i, $i = 1...k$ after learning Y are identically $Dest(s, VG_i(s)) \bigcup (d, 1)$. Per the definition of a via group, $l_1^v, l_2^v, ...l_k^v$ still belong to the same via group. □

Theorem 4.3. *The VIGO Learner algorithm is correct, i.e., the via locations assigned in the same via group in a learned VIGO Index always have the same destination distribution.*

Proof. Algorithm 4.2 always moves the via locations that were in the same old via group and passed by the same trajectory to the same new via group. According to Lemma 4.2 and Definition 4.1, the VIGO Learner algorithm is correct. □

4.4 Spatio-Temporal Destination Prediction

To complete the entire trajectory destination prediction component presented in Figure 2, we need to predict not only the destination location, but also the arrival time. However, equation 3 only predicts the location of the destination. To predict the destination time, we calculate travel time probability distribution between pairs of via and destination locations, defined as $p(\Delta t|c, d)$, where d is destination and c is a via location $\Delta t = t_d - t_c$. Therefore, we compute the probability of spatio-temporal destination $\{d, t_d\}$ for sub-trajectory Y_c with source s and current location c using the following equation:

$$p(\{d, \Delta t\}|s \cap c) = p(d|s \cap c) \times p(\Delta t|c, d) \quad (4)$$

Equation 4 is obtained based on the assumption that the travel time between two given points c and d is independent of the points the trajectory visited before c. In other words, the travel time between two points in space only depends on the two points themselves. We argue that this assumption is reasonable because unlike destination, travel time is only determined by the route that will be taken rather than determined by a travel plan made ahead of time.

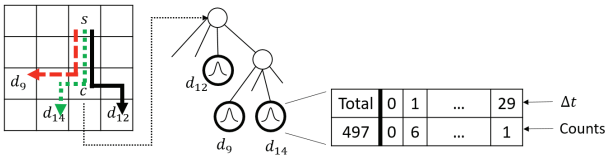


Figure 6: Travel time distributions.

To save memory cost, we use a data structure similar to the top two levels of the VIGO Index to store the travel time distributions. Figure 6 shows an example of this data structure. The top level is a two dimensional grid index for the current location c. Each grid points to a quad-tree of possible destinations from c. Each leaf node of this quad-tree contains the destination d as well as an array of possible travel time and the corresponding counts. Through the analysis of our data we find that more than 93% of all the trips are shorter than 30 minutes. Therefore we limit the size of the array to 30. The travel time distribution can be learned simultaneously with the VIGO Index structure. We integrate this process with the VIGO Learner algorithm and design a *VIGO_ST* algorithm. The pseudo code is presented in Algorithm 2. The algorithm takes one trajectory at a time and scans all the via locations sequentially. A via location v is used to update the VIGO index first (Line 5), and then the count for travel time $d.t - v.t$ is incremented (Line 6-9). The output is an integrated ST destination predictor M_{ST}. Compared to the learning phase, destination prediction using the ST destination predictor is relatively simple. Given a partial trajectory Y_c with source s and current location c, we first predict the destination location probability. This can be done by finding $via(s, c)$ at the via node of c in VIGO index. Then we also find $desc(s, c, d_i)$ for all the possible destinations d_i by scanning every entry in the destination array of c's via group. The destination probability is calculated according to Equation 3. Then we find all the possible travel time values between c and d_i and their probability. Finally we use Equation 4 to calculate the ST destination probabilities.

5 GATHERING EVENT FORECASTING

This section presents the Event Forecasting component of our proposed solution. First we develop an online model for continuous prediction of arrival counts in every location. Then based on the predicted arrivals we identify the spatio-temporal footprint of the gathering events.

Algorithm 2: VIGO_ST Learner

Input: List of all trajectories (X)
Output: A Spatio-Temporal Destination Predictor M_{ST}

```
1  cur_trj ← 0; M ← NULL; L ← NULL
2  for each Y ∈ X do
3      Q_v ← M[Y.s]
4      for each via location v ∈ Y do
5          Same as Line 5-26 in VIGO Learner
6          if L[v].get_node()==NULL then
7              node ← L[v].insert(d)
8          node.count[t_d - t_v]++
9          node.total++
10     Same as Line 27 in VIGO Learner
11 return M_ST=(M,L)
```

Algorithm 3: VIGO_ST Predictor

Input: A VIGO_ST Model $M_{ST} = (M, L)$, Sub-trajectory Y_c
Output: Destination probability D at each location and time

```
1  D ← 0
2  via_node ← M[Y_c.s].get_node(c)
3  via_count ← via_node.count
4  dst_array ← via_node.group->dst_array
5  for each d in dst_array do
6      prob_d = dst_array[d]/via_count
7      for each Δ_t in L[c].get_node(d) do
8          prob_t = (L[c].get_node(d))[d]
9          D[d][t_c + Δ_t] ← prob_l × prob_t
10 return D
```

5.1 Online Arrival Count Prediction

As mentioned in Section 3.3, due to the temporal non-stationarity in the urban trips, one global prediction model may not make accurate predictions at all time, especially for trips to rare gathering events. Instead, recent trajectories may better reflect short-term changes of trip patterns. Therefore we propose an online destination prediction model, which consists of a historical model learned offline using long-term historical data, and a recent model dynamically built only based on recently observed trajectories. The final destination probability of each location and arrival time is calculated as a weighted average of the results from these two models. We update the online model for every time window to continuously predict the arrival count at each location and time slot.

First we split historical trajectories X into two subsets, X_τ which contains the completed trajectories within the last τ time-steps, and X_h which contains all the earlier trajectories. Then we train a historical model M_h using only X_h, and an online model M_o using X_τ. (Line 1-2). Then we predict the destinations of each sub-trajectory at time t_g. When doing destination prediction, we feed a sub-trajectory to both M_h and M_o. The final destination probability is calculated as a weighted average as shown in Equation 5. β is a weight between 0 and 1 to adjust how much we trust the M_h versus M_o. $p_h(\{d, \Delta\}|s, c)$ and $p_o(\{d, \Delta\}|s, c)$ are the destination probabilities of d, after Δt time-steps given by M_h and M_o respectively, while $\Delta t = t_d - t_c$. Note that due to the limited amount of data used in the online model, it is possible that the (s, c) combination is not found in the online model. In such cases we use the result of historical model.

$$p(\{d, \Delta t\}|s \cap c) = (1 - \beta) \times p_h(\{d, \Delta t\}|s \cap c) + \beta \times p_o(\{d, \Delta t\}|s \cap c) \quad (5)$$

The arrival count is predicted as the expectation of trips ending at each location and time slot, given the list of sub-trajectories at t_c. We calculate the predicted arrival count of each location at target time t_g as follows:

$$A(d, t_c, t_d) = \sum_{Y_c \in U(t_c)} p(\{d, t_d - t_c\}|s \cap c) \quad (6)$$

Where $U(t_c)$ is the list of sub-trajectories at time t_c, s is the source and c is the current location of Y_c.

The above procedure is summarized in the line 1 to 5 in Algorithm 4. After the prediction is made, we discard the online model for time t_c and rebuild a new online model in the next time slot $t_c + 1$. Meanwhile, the all the trajectories completed in time t_c are used to update the historical model M_h (Line 24).

5.2 Event Forecasting

After obtaining the predicted counts at each location at time t_g, we find the top-k ST regions with statistically significant arrival counts based on Definition 3.4. Scalable algorithms have been proposed to identify regions of statistically significant hotspots and events [14, 15]. These algorithms find the most likely event by searching all the possible spatio-temporal regions with pruning strategies. However, finding the exact solution is computationally costly and is inapplicable in the context of online event monitoring. Thereby we use a heuristic algorithm to identify k events that are statistically significant.

Given the predicted arrival count for each location at time t_g, we first find the grid locations with predicted arrival counts significantly higher than their respective baselines. We feed them as seeds to an area expansion algorithm to summarize the footprints of potential events. Algorithm 4 shows how the online procedure and the proposed expansion algorithm work. First, each location is tested using Definition 3.4 to filter those with significantly high counts, which are added to a seed list E_s (lines 7 to 9). Then we sort E_s based on their LLR score and pick each seed location $E_i \in E_s$ to expand. For every iteration, we expand E_i by moving its boundary on one of the four directions further by one grid. Whichever direction results in the highest new LLR score for E_i is chosen. If expansion from none of the directions results in a significant count for E_i, the expansion stops (lines 12 to 18). The result is a rectangularly-shaped area, which is added to the priority queue E as a predicted event. Any other significant location that are included in E_i as a result of expansion, are removed from the original seed list E_0. Finally, the top k elements of E are returned. To continue the real-time prediction, X_τ, U are updated and the trajectories that completed at t_c are fed to M_h to be learned.

Algorithm 4: The event forecasting procedure

Input: Historical trajectories (X_h), Recent trajectories (X_τ), Sub-trajectories (U), current time t_c, target time t_g, baseline arrival count at each location at t_g (B), k, α
Output: k significant gathering events

$M_h \leftarrow VIGO\_ST\_Learner(X)$; $A \leftarrow 0$
while *program not terminated* **do**
  $M_o \leftarrow VIGO\_ST\_Learner(X_\tau)$
  for *each* Y_c *in* U **do**
    $A \leftarrow A + VIGO\_ST\_Predictor(Y_c, M_h) \times (1-\beta) + VIGO\_ST\_Predictor(Y_c, M_o) \times \beta$
  $E, E_0 \leftarrow \emptyset$
  for *all locations* d **do**
    if *is_significant*$(A[d][t], B[d][t], \alpha)$ **then**
      $E_0 \leftarrow E_0 \bigcup d$
  Sort E_0 on $LLR(d)$ in descending order
  for *each* $d \in E_0$ **do**
    $G \leftarrow d$
    do
      for *directions* $dir \in \{top, right, bottom, left\}$ **do**
        $G\_next[dir] \leftarrow$ expand G along dir
        $score[dir] \leftarrow LLR(G\_next[dir])$
      $dir\_exp = argmax_{dir}\{score[dir]\}$
      if *is_significant*$(G\_next[dir\_exp])$ **then**
        $G \leftarrow G\_next[dir\_exp]$
    while G *expanded*
    Add G to E
    $E_0 \leftarrow E_0 - (E_0 \cap G)$
  Sort E on $LLR(G)$ in descending order
  output E.top(k)
  Update X_τ, U, M_h; $t_0 = t_0 + 1$

6 EVALUATIONS

6.1 The Dataset

The dataset we use contains the GPS records taxis operating in Shenzhen, China. The data is recorded during the month of November 2014. The location of each taxi is recorded periodically in short intervals. In addition to location, each record includes a taxi identifier, a time-stamp and a field that indicates if the taxi has a passenger or not. We map every record into a 128×64 grid with cells of size 500×500 meters. Also, we map the records into one-minute time intervals. We extract the taxi trips from this dataset by excluding the records in which the taxi is unoccupied which results in around 14 million trips. Because the longest distance in study area in Shenzhen can be traveled in less than two hours, we exclude all the trajectories that have trip duration longer than 120 minutes as not being purpose-driven trips.

6.2 Case Study

To demonstrate the effectiveness of the event forecasting framework, we apply our proposed method to the real-world dataset described above. We use a default setting for the parameters: $\beta = 0.9$, $\tau = 30$. We train the VIGO model for all days in the month, excluding the day on which we are predicting events. Moreover, we trained different models for weekdays, Saturdays and Sundays. Then we ran algorithm 4 for every minute of the day. Figure 7 shows a predicted event on November 21^{st}, 2014. The black dot in figure 7 (a) to (c) and the black arrow in (d) and (e) are the location of Shenzhen Bay Sports Center. The red box in figure 7 (a)-(b) is the area reported by algorithm 4 at $\alpha = 0.01\%$. After we observed this output, we looked into public records and found that it corresponds to a real event, i.e. a concert that started at 20:00 with nearly 30,000 attendees [1]. Figure 7 (a) shows the forecast 10 minutes before the event (i.e. 19:50). Figure 7 shows the prediction 5 minutes before the event (i.e. 19:55). Figure 7 (c) shows footprint of the event obtained by applying the area expansion algorithm to true arrival counts. Figure 7 (d) shows a heat-map of the predicted arrival counts of the trips that are going to end at event location using only the historical model. We can see the drops are predicted to be spread in a wider region around the stadium rather than the stadium itself. This is caused by the temporal non-stationarity of the urban trips, i.e. historically, similar trajectories often end in other locations rather than the stadium. Figure 7 (e) shows the predictions of the same trajectories using equation 5, i.e. using both historical and online models. This time the predicted counts are correctly concentrated at the stadium because the online model has captured the recent behavior of the recent urban trips and is able to predict the event.

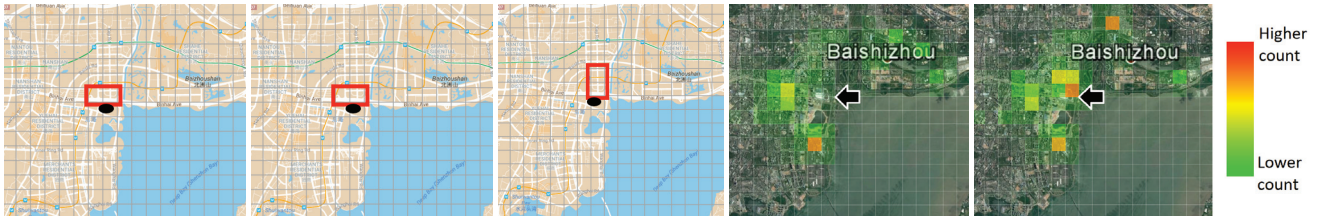
Figure 8 (a) and (b) show the predicted arrival count error. The counts are predicted 10 and 5 minutes before the target time. Negative values mean underestimation and 0 is the time of the event. It is clear that the historical model consistently underestimates the arrival counts while incorporating online model reduces the error effectively. Figure 8 (c) shows the predicted counts as we get closer to the event time, i.e. target time is fixed to the event time (20:00). We can see that the counts predicted by the historical model never reach the significance threshold, thus making it impossible to forecast the event, while incorporating the online model allows us to forecast the event 11 minutes in advance.

Figure 9 shows the result of continuous forecasting from one hour before the event. The x-axis is the target time of forecast, while the y-axis is the earliest time when a significant event at each target time is forecast. The dashed line along the diagonal represents real-time detection, i.e. no forecasting. The curve being located well below the dashed line indicates that our proposed approach consistently forecast the event ahead of the time. The average forecast time is 10 minutes before the target time.

6.3 Experiments

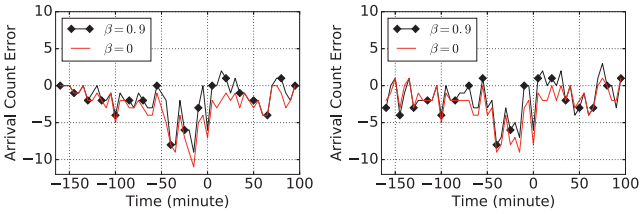
In this section we conduct experiments to evaluate the accuracy and scalability of the proposed solutions.

Destination Prediction Accuracy. In this experiment, we measure the prediction error of the VIGO approach and compare it with related work, to show how relaxing the Markov property assumption impacts prediction accuracy. To do this, we implemented the



(a) 10 minutes ahead of the time. (b) 5 minutes ahead of the time. (c) Footprint based on true arrivals. (d) Historical model. (e) Historical and online model.

Figure 7: Event predicted on day 21 (best viewed in color).



(a) 10 minutes before. (b) 5 minutes before. (c) Predicted counts with $\tau = 30$.

Figure 8: Effect of incorporating the online model in arrival count prediction

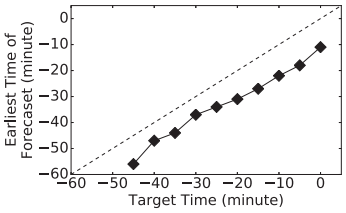
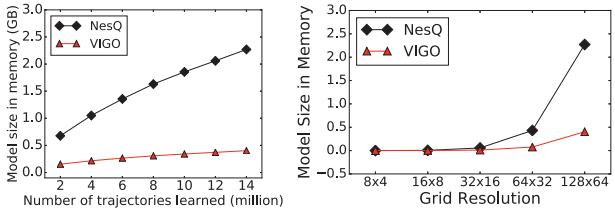


Figure 9: Earliest time of event forecast.

Table 1: Destination location prediction error for VIGO algorithm vs. Sub-Syn[22] measured in Manhattan Distance.

Completion	k	VIGO	Sub-Syn
30%	1	10.55 (grid cells)	12.78 (grid cells)
70%	1	6.06 (grid cells)	8.01 (grid cells)
30%	5	7.91 (grid cells)	8.24 (grid cells)
70%	5	3.76 (grid cells)	4.75 (grid cells)
Prediction Rate	-	99.76%	96.69%



(a) By increasing number of trajectories learned. (b) Varying grid size.

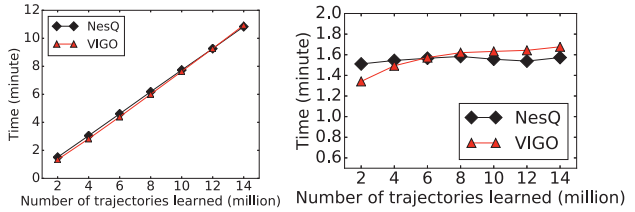
Figure 10: Model size in memory.

method proposed by Xue et al. [22] and ran it on our dataset with one day's data held out of training for testing. We used the default settings presented in section 6.2. We use sub-trajectories that are 30% and 70% completed, and measure the Manhattan Distance between the true and the predicted destinations. We use the closest predicted destination among the top-k given by each method as the predicted destination, with $k = 1$ and $k = 5$. The results in table 1 show that the VIGO approach performs consistently better than the competitor. Also we are able to give predictions on more test sub-trajectories (99.76% vs. 96.69%).

Memory Cost Evaluation. In this experiment, we examine the memory cost of the VIGO Index and the NesQ approach by varying number of trajectories learned and varying grid size. For this experiment, we run the algorithm for the entire dataset with default settings, i.e. we train the model of Equation 3 using all the 14 million trajectories in the dataset. We measure the size of the model in memory as we train every 2 million trajectories. Figure 10 (a) shows that the growth of VIGO's size in memory is orders of magnitude slower than NesQ. The final size of the model learned by VIGO is 412 MB vs. 2.27 GB of NesQ, yielding a memory cost saving as high as 82%.

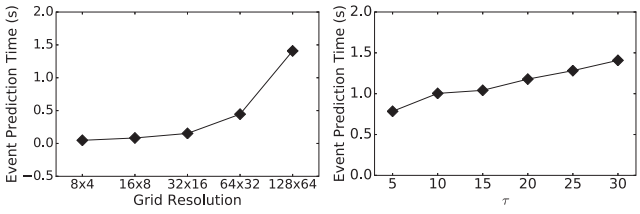
Then, we measure the size of the model when using VIGO and NesQ structures by varying the grid size. Figure 10 (b) shows that size of the model in memory increases by increasing the number of grid cells in both structures. However, the growth is much faster when using NesQ.

Running Time Evaluation. In this experiment, we evaluate the running time of the proposed solutions. First, we evaluate the training time. We train both NesQ and VIGO using the entire dataset and measure the total training time by increasing number of trajectories learned and the time spent to learn every 2 million trajectory as the model gets larger. Figure 11 (a) shows the training time as the model learns more trajectories. This figure shows that both structures can be learned efficiently with almost the same training time. Figure 11 (b) shows the time spent to learn every 2 million trajectories by increasing number of trajectories learned. Although VIGO shows minimal increase in training time as the model gets larger, both NesQ and VIGO have stable learning times at any stage of training. Finally, we evaluate the event forecasting processing time. Fast processing is important for continuous event forecasting. Event forecasting include online model training, calculating destination probabilities for all destinations and running the area



(a) Total learning time with growing number of trajectories. (b) Learning time of 0.2 million trajectories by increasing number of trajectories learned.

Figure 11: Training time evaluation.



(a) Varying grid size. (b) Varying τ

Figure 12: Event Forecasting time.

expansion algorithm. In this experiment we use one-minute time slots. Figure 12 shows the time cost of doing forecasting for one target time by varying grid size and τ. Event forecasting time increases by increasing both parameters. In Figure 12 (b) the event prediction time increases since more trajectories are used to train the online model. The results show that even with finest grid resolution and largest τ the forecasting time cost is less than 1.5 seconds. This level of performance makes it possible for real-time forecasting of gathering events at 1-minute level.

7 CONCLUSIONS

In this paper, we addressed the gathering event forecasting problem through destination prediction of incomplete trips. Event forecasting in urban setting is important to traffic management and public safety. Prior event detection techniques are mostly descriptive, which only reply on on-site observations such as taxi drop-offs therefore lacking the ability to make forecasts ahead of the time. Our work, for the first time, solved the gathering event forecasting problem through trajectory destination prediction. We relaxed the Markov property commonly assumed by related work, and addressed the consequent memory cost challenge through a novel Via Location Group (VIGO) approach. We also addressed the temporal non-stationarity of urban trip patterns through an online prediction mechanism. A case study and experiments showed that our proposed approach could effectively and timely predict gathering events ahead of time with orders of magnitude less memory cost than baseline solutions.

REFERENCES

[1] 2014. 10th Anniversary of the Mixc - Super Stars Concert. http://news.ifeng.com/a/20141128/42597678_0.shtml. (2014). Accessed: 2017-06-17.

[2] Juan Antonio Alvarez-Garcia, Juan Antonio Ortega, Luis Gonzalez-Abril, and Francisco Velasco. 2010. Trip destination prediction based on past GPS log using a Hidden Markov Model. *Expert Systems with Applications* 37, 12 (2010), 8166–8171.

[3] Ling Chen, Mingqi Lv, and Gencai Chen. 2010. A system for destination and future route prediction based on trajectory mining. *Pervasive and Mobile Computing* 6, 6 (2010), 657–676.

[4] Ling Chen, Mingqi Lv, Qian Ye, Gencai Chen, and John Woodward. 2011. A personal route prediction system based on trajectory data mining. *Information Sciences* 181, 7 (2011), 1264–1284.

[5] Liang Hong, Yu Zheng, Duncan Yung, Jingbo Shang, and Lei Zou. 2015. Detecting urban black holes based on human mobility data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 35.

[6] John Krumm, Robert Gruen, and Daniel Delling. 2013. From destination prediction to route prediction. *Journal of Location Based Services* 7, 2 (2013), 98–120.

[7] John Krumm and Eric Horvitz. 2006. Predestination: Inferring destinations from partial trajectories. In *International Conference on Ubiquitous Computing*. Springer, 243–260.

[8] Martin Kulldorff. 1997. A spatial scan statistic. *Communications in Statistics-Theory and methods* 26, 6 (1997), 1481–1496.

[9] Martin Kulldorff. 2001. Prospective time periodic geographical disease surveillance using a scan statistic. *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 164, 1 (2001), 61–72.

[10] Martin Kulldorff, William F Athas, Eric J Feurer, Barry A Miller, and Charles R Key. 1998. Evaluating cluster alarms: a space-time scan statistic and brain cancer in Los Alamos, New Mexico. *American journal of public health* 88, 9 (1998), 1377–1380.

[11] Martin Kulldorff, Richard Heffernan, Jessica Hartman, Renato Assunçao, and Farzad Mostashari. 2005. A space-time permutation scan statistic for disease outbreak detection. *PLoS medicine* 2, 3 (2005), 216.

[12] Xiang Li, Mengting Li, Yue-Jiao Gong, Xing-Lin Zhang, and Jian Yin. 2016. T-desp: Destination prediction based on big trajectory data. *IEEE Transactions on Intelligent Transportation Systems* 17, 8 (2016), 2344–2354.

[13] Zhongmou Li, Hui Xiong, and Yanchi Liu. 2012. Mining blackhole and volcano patterns in directed graphs: a general approach. *Data Mining and Knowledge Discovery* 25, 3 (2012), 577–602.

[14] Daniel B Neill. 2009. Expectation-based scan statistics for monitoring spatial time series data. *International Journal of Forecasting* 25, 3 (2009), 498–517.

[15] Daniel B Neill and Andrew W Moore. 2004. Rapid detection of significant spatial clusters. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 256–265.

[16] Hanan Samet. 1990. *The design and analysis of spatial data structures*. Vol. 199. Addison-Wesley Reading, MA.

[17] Kohei Tanaka, Yasue Kishino, Tsutomu Terada, and Shojiro Nishio. 2009. A destination prediction method using driving contexts and trajectory for car navigation systems. In *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 190–195.

[18] Amin Vahedian Khezerlou, Zhou Xun, Lufan Li, Zubair Shafiq, Alex X. Liu, and Fan Zhang. 2017. A Traffic Flow Approach to Early Detection of Gathering Events: Comprehensive Results. *ACM Transactions on Intelligent Systems and Technology* (2017).

[19] Liang Wang, Zhiwen Yu, Bin Guo, Tao Ku, and Fei Yi. 2017. Moving Destination Prediction Using Sparse Dataset: A Mobility Gradient Descent Approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 11, 3 (2017), 37.

[20] Wikipedia. 2016. 2014 Shanghai stampede — Wikipedia, The Free Encyclopedia. (2016). https://en.wikipedia.org/w/index.php?title=2014_Shanghai_stampede&oldid=701733900 [Online; accessed 28-June-2016].

[21] Andy Yuan Xue, Jianzhong Qi, Xing Xie, Rui Zhang, Jin Huang, and Yuan Li. 2015. Solving the data sparsity problem in destination prediction. *The VLDB Journal* 24, 2 (2015), 219–243.

[22] Andy Yuan Xue, Rui Zhang, Yu Zheng, Xing Xie, Jin Huang, and Zhenghua Xu. 2013. Destination prediction by sub-trajectory synthesis and privacy protection against such prediction. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 254–265.

[23] Di Xue, Li-Fa Wu, Hua-Bo Li, Zheng Hong, and Zhen-Ji Zhou. 2017. A novel destination prediction attack and corresponding location privacy protection method in geo-social networks. *International Journal of Distributed Sensor Networks* 13, 1 (2017), 1550147716685421.

[24] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. 2011. Who are you with and where are you going?. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 1345–1352.

[25] Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. 2011. Semantic trajectory mining for location prediction. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 34–43.

[26] Xun Zhou, Amin Vahedian Khezerlou, Alex Liu, Zubair Shafiq, and Fan Zhang. 2016. A traffic flow approach to early detection of gathering events. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 4.