A Traffic Flow Approach to Early Detection of Gathering Events: Comprehensive Results

AMIN VAHEDIAN KHEZERLOU, The University of Iowa
XUN ZHOU, The University of Iowa
LUFAN LI, The University of Iowa
ZUBAIR SHAFIQ, The University of Iowa
ALEX X. LIU, Michigan State University
FAN ZHANG, SIAT, Chinese Academy of Sciences, Shenzhen, China

Given a spatial field and the traffic flow between neighboring locations, the early detection of gathering events (EDGE) problem aims to discover and localize a set of most likely gathering events. It is important for city planners to identify emerging gathering events which might cause public safety or sustainability concerns. However, it is challenging to solve the EDGE problem due to numerous candidate gathering footprints in a spatial field and the non-trivial task to balance pattern quality and computational efficiency. Prior solutions to model the EDGE problem lack the ability to describe the dynamic flow of traffic and the potential gathering destinations because they rely on static or undirected footprints. In our recent work, we modeled the footprint of a gathering event as a Gathering Graph (G-Graph), where the root of the directed acyclic G-Graph is the potential destination and the directed edges represent the most likely paths traffic takes to move towards the destination. We also proposed an efficient algorithm called SmartEdge to discover the most likely non-overlapping G-Graphs in the given spatial field. However, it is challenging to perform a systematic performance study of the proposed algorithm, due to unavailability of the ground truth of gathering events. In this paper, we introduce an event simulation mechanism, which makes it possible to conduct a comprehensive performance study of the SmartEdge algorithm. We measure the quality of the detected patterns, in a systematic way, in terms of timeliness and location accuracy. The results show that, on average, the SmartEdge algorithm is able to detect patterns within a grid cell away (less than 500 meters) of the simulated events and detect patterns of the simulated events as early as 10 minutes prior to the first arrival to the gathering event.

CCS Concepts: • **Information systems → Geographic information systems;**

Additional Key Words and Phrases: Gathering Event; Early Detection; Spatial Data Mining

ACM Reference Format:
Amin Vahedian Khezerlou, Xun Zhou, Lufan Li, Zubair Shafiq, Alex X. Liu, and Fan Zhang, 2016. A Traffic Flow Approach to Early Detection of Gathering Events. *ACM Trans. Intell. Syst. Technol.* 9, 4, Article 39

This work is partially supported by the National Science Foundation under Grant Numbers 1566386, CNS-1318563, CNS-1524698, CNS-1421407, and IIP-1632051, the National Natural Science Foundation of China under Grant Numbers 61472184 and 61321491, the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program, the China National Basic Research Program (973 Program) under Grant 2015CB352400, the Research Program of Shenzhen under Grants JSGG20150512145714248, KQCX2015040111035011 and CYZZ20150403111012661, and the Obermann Center for Advanced Studies Interdisciplinary Research Grant at the University of Iowa.
Authors' addresses: A. V. Khezerlou and X. Zhou, Department of Management Sciences, The University of Iowa; L. Li and Z. Shafiq, Computer Science Department, The University of Iowa; emails:{aminvahediankhezerlou, xun-zhou, lufan-li, zubair-shafiq}@uiowa.edu; A. X. Liu, Department of Computer Science and Engineering, Michigan State University; email: alexliu@cse.msu.edu; F. Zhang, SIAT, Chinese Academy of Sciences, Shenzhen, China; email: zhangfan@siat.ac.cn

DOI: 0000001.0000001

(January 2017), 24 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Background & Motivation. A gathering event is the process where a large number of moving objects (e.g., taxi cabs, pedestrians) arrive at a specific destination during a time period via different paths. Typical examples of gathering events include but are not limited to: (1) a traffic congestion where more-than-usual number of vehicles arrive at a specific road segment or intersection and are not able to leave during the same time period, (2) fans arriving at the vicinity of a stadium before a sports event starts, and (3) protesters gathering at a destination (e.g., a park) in a pre-planned or unplanned social protest. Large-scale gathering events have a significant impact on urban planning and public safety. For example, traffic congestion leads to extra gas emissions and low transportation efficiency. As another example, social activities such as sports events may put a strain on public resources (e.g., parking spaces, cellular network capacity) and may even become a threat to public safety. Although many gathering events are predictable and occur regularly (e.g., football games, concerts), some others are rare or unexpected. Even for planned gathering events, the traffic volume may far exceed the expectation. For example, the stampede in Shanghai in 2014 was a result of larger-than-expected crowd gathered for new year celebrations [Wikipedia 2016]. Therefore, it is important for city planners and other stakeholders to have the ability to identify gathering events as early as possible.

Problem Statement. This paper investigates a computational solution to detecting gathering events based on human mobility data. Specifically, it focuses on detecting the footprint of gathering events where the total amount of moving objects is much higher than expected. Given a spatial framework that partitions the space into a grid, the *Early Detection of Gathering Event* (EDGE) problem aims to discover top-k most likely gathering events, their destinations, and the most likely routes along which moving objects gather at the destination. The EDGE problem is challenging to solve because the total number of possible event footprints in a spatial framework grow exponentially as a function of the number of spatial grids in the study area, and it is non-trivial to balance the quality of results (e.g., statistical significance) and computational efficiency.

Limitations of Prior Art. Prior work on event detection can be broadly categorized into two groups. The first group of methods are limited to identifying events with regularly-shaped footprints [Kulldorff et al. 1998][Kulldorff 2001][Neill 2009][Neill and Moore 2004] (e.g., circular, rectangular). The second group of methods are limited to finding undirected graph footprints (e.g., black holes or volcanoes [Hong et al. 2015][Li et al. 2010][Li et al. 2012]). *The key limitation of these two broad categories is that they lack the ability to capture how moving objects gather towards a specific destination*. In our recent work [Zhou et al. 2016], we introduced the concept of Gathering Graphs (G-Graph) to model gathering events. A G-Graph is a directed acyclic graph, the root node of which models the location of the gathering event, and the edges model the paths that moving objects take to the event location.

Contributions of Our Prior Paper. In our prior paper [Zhou et al. 2016], we made the following contributions: (1) We formulated the problem as a computational problem by proposing the concept of a Gathering Graph (G-Graph), which models the event location and the paths to the event. Then we proposed Gathering Score (GScore) as a metric to quantify the likelihood of event location and paths specified by each G-Graph. (2) We proposed the SmartEdge algorithm to efficiently discover the top-k G-Graphs. (3) We performed a case study on real taxi trajectory data from Shenzhen, China.

The algorithm was able to detect a gathering event, which corresponded to a large concert in Shenzhen Stadium. In addition, we conducted experimental evaluations, which showed that the SmartEdge reduced the computation time by 50% as compared to the baseline method [Zhou et al. 2016].

New Contributions in This Paper. This paper is a significant extension to our recent work [Zhou et al. 2016]. In this paper, we propose a systematic approach to quantitatively study the performance of the SmartEdge algorithm, i.e. the quality of the detected patterns in terms of their **timeliness** and **location accuracy**. Evaluating the quality of the discovered patterns of gathering events is challenging because the ground truth of the real-world events is not systematically available. To address this challenge, we present **the following contributions**: **(1)** We propose a framework to simulate gathering events based on real-world traffic patterns. **(2)** We quantify the location accuracy of the detected patterns by defining an error function called *destination error*. **(3)** We conduct extensive simulations with various parameter settings using the proposed simulation framework and the error function. The simulation results show that the SmartEdge algorithm, on average, is able to detect the event location with an error less than one grid cell (500 meters) and as early as 10 minutes prior to the arrival of the first vehicle.

The paper is organized as follows. Section 2 presents the concepts and definitions, followed by a problem statement. Computational Solutions to the EDGE problem are discussed in Section 3. Section 4 presents the comprehensive evaluation of our solution in the previous work as well as the case study. Experimental evaluation on computational efficiency are presented in Section 5. Related work is discussed in Section 6. Finally Section 7 concludes the paper.

2. PROBLEM FORMULATION

2.1. Overview of Gathering Events Detection

The EDGE problem formulation is based on the following traffic monitoring workflow: the real-time traffic flow of the entire region is monitored. High-volume traffic flows crossing the boundaries of adjacent grid cells are identified. Then an algorithm finds the most likely destination of the traffic (if any) as well as a flow map of the traffic to the destination. For example, when there is an event at some location, the flows on the paths leading to the event must increase significantly prior to the event. The algorithm finds the most likely destination of the high flows and the paths the flow will take to arrive at the event location. The model also works with a sample of the traffic flow data, given that the sample follows the same distribution and covers the entire region.

2.2. Concepts and Definitions

A *spatial field* S is a two-dimensional region (e.g., a city) partitioned into grid cells s_1, s_2, ... s_n. Given a spatial field, the location of a moving object (e.g., taxi) at a certain time thus could be mapped to one of the grids.

A directed edge $e = (s_i, s_j)$ can be defined between each pair of adjacent grids s_i and s_j. Given a certain time interval (e.g., 19:40-19:50, August 1, 2013), the **observed traffic flow** along e (denoted as C_e) is a non-negative integer that measures the number of moving objects (e.g., taxis) going from s_i to s_j. In reality, traffic flows can be obtained from sensing devices such as loop detectors or by processing real-time GPS locations of moving objects.

Next we propose a mechanism to quantify the abnormality of traffic flow observed on each edge. Poisson distribution is commonly used to model the number of discrete events such as the total number of car arriving at an intersection during a time interval. For each edge, we use a Chi-square test to fit a Poisson distribution on its historical

traffic flow data. Results show that the observed traffic flows along an edge in the same time interval of day can be well approximated with a Poisson distribution with mean equals the average of the observed traffic. We call this mean the baseline traffic flow of edge e (denoted as B_e).

Various tests have been developed to identify statistically significant anomalous patterns. For example, Kulldroff's Spatial Scan Statistics [Kulldroff 1997] is a classic model for data with Poisson distributions, where the likelihood ratio between alternative hypothesis H_1 (risk inside a region is higher than outside) and the null hypothesis H_0 (risk is the same everywhere) is maximized over all the possible regions. Here we employ the idea used by Neill [Neill 2009], which simplified Kulldroff's spatial scan statistic model. Neill et al proposed an Expectation-based Poisson (EBP) model, which compares the observed value of a region with its own historical average instead of the counts outside. We employ the idea of EBP in our problem and propose the following hypothesis testing mechanism.

Assume C_e is the observed traffic flow along e in a time interval t, and B_e is the baseline traffic flow in the same time interval of day. Under the null hypothesis H_0, the observed flow C_e is drawn from Poisson(B_e). The alternative hypothesis H_1 is: the observed flow C_e is drawn from a different Poisson distribution with an elevated mean value qB_e where $q \geq 1$. The EBP test maximizes the likelihood ratio between H_1 and H_0 ($\frac{Pr(C_e|H_1)}{Pr(C_e|H_0)}$) when $q = \frac{C_e}{B_e}$ ($C_e \geq B_e$). When $C_e < B_e$ (i.e., the observed flow is lower than the expected), the score is set to 0. Note when $B_e = 0$ (e.g., no road or inside a park), the corresponding edge is removed from the spatial field thus not included in any calculation. The log likelihood ratio of the given observed flow thus can be expressed as follows:

$$LLR(e) = \begin{cases} C_e \log \frac{C_e}{B_e} + (B_e - C_e) & \text{if } C_e \geq B_e \\ 0 & otherwise \end{cases} \quad (1)$$

Definition 2.1. A *Significant Flow* in spatial field S in a time interval t is an edge $e = (s_i, s_j)$ such that LLR(e)¿0 and LLR(e) is statistically significant at α level.

The statistical significance of a LLR score is typically assessed via Monte-Carlo simulations to filter results generated by random chance. Each trial of the simulation will generate a random observation C_{rand} under null hypothesis H_0 and calculates the corresponding score $LLR_{rand}(e)$. The actual score LLR(e) is significant at α level if no more than $\alpha * 100\%$ of all the simulated score LLR$_{rand}(e)$ are as high as LLR(e). However, Monte-Carlo simulation is very computationally expensive. Due to the properties of the log likelihood ratio score, a closed form of the p-value can be derived thus avoiding Monte-Carlo simulations.

LEMMA 2.2. *Given two edges e_1 and e_2 with the same baseline flow B and different observed flows C_{e_1} and C_{e_2} ($C_{e_1} \geq B$ and $C_{e_2} \geq B$), respectively, LLR(e_1) $\geq$ LLR(e_2) $\iff$ $C_{e_1} \geq C_{e_2}$.*

PROOF. The partial derivative $\frac{dLLR(e)}{dC_e} = log\frac{C_e}{B_e} > 0$ shows that the LLR(e) function is monotonically increasing with increasing C_e and fixed B_e when $C_e \geq B_e$. Thus the lemma is proved. □

LLR(e) is significant at α level means: the chance that the score $LLR_{rand}(e)$ calculated based on randomly generated observation C_{rand} under H_0 is no less than the real LLR(e) is at most α. According to Lemma 2.2, $LLR_{rand}(e) \geq LLR(e)$ when and only when the $C_{rand} \geq C_e$. In other words, for a LLR(e) to be significant at 0.005 level, the actual C_e must be no lower than at least 99.5% of the random C_{rand} drawn from

Poisson(B_e). This can be quickly tested in constant time by comparing 1-$Pr(X < C_e)$ with α. LLR(e) being tested is significant at α level when 1-$Pr(X < C_e) \leq \alpha$.

Given a significant flow e and a potential destination r, we evaluate how likely it is to find a path connecting e and r such that the traffic along this path is higher than normal.

Definition 2.3. Given a grid r and a significant flow $e_{in} = (s_i, s_j)$, an *incoming path* p_{in} of r from e_{in} is a sequence of directed edges $e_{in}, e_1, e_2, ..., e_n$such that p_{in} is a shortest path in Manhattan distance from s_j to r. e_{in} is called an incoming significant flow of r. $dist(r, e_{in}) = dist(r, s_j)$.

The shortest path constraint is added based on the assumption that most of the moving objects and vehicles should take the shortest path when gathering towards a destination. Note sometimes there might not exist a shortest path between a significant flow and a destination. For example in Figure 1(a) there is no incoming path from e_2 to r. In such cases we may define an outgoing path along which traffic goes out of r to e_{sig}.

Definition 2.4. Given a grid r and a significant flow $e_{out} = (s_i, s_j)$, an *outgoing path* p_{out} of r to e_{out} is a sequence of directed edges $e_1, e_2, ..., e_{out}$ such that p_{out} is a shortest path in Manhattan distance from r to s_i. e_{out} is called an outgoing significant flow of r. $dist(r, e_{out}) = dist(r, s_i)$.

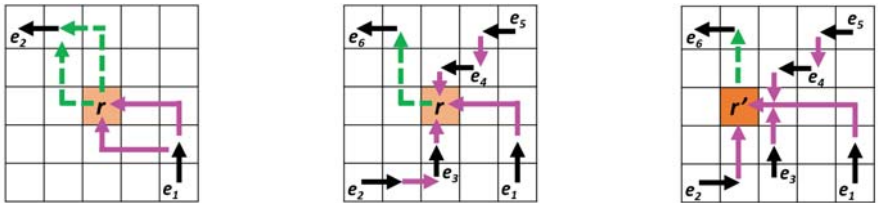
Note that a significant flow cannot be both incoming and outgoing for the same grid r since the directed shortest path between them is unique. Figure 1(a) shows an example of incoming paths and outgoing paths. e_1 and e_2 are two significant flows. There exist two possible incoming path from e_1 to r (solid, pink arrows) and two possible outgoing paths from r to e_2 (dashed, green arrows).

Next we quantify the likelihood that a path has a high traffic. One possible way might be to employ the idea used in the Spatial Scan Statistic and EBP. They assumed that all the locations (edges in our case) inside the event footprint (a path in our case) have a **uniform elevation** q. Then the sum of baseline $\sum B_e$ and the sum of observed counts $\sum C_e$ are used to calculate the LLR score of a path using Equation 1. This idea, however, does not work well in our case. Even in the same gathering event, different edges may have different degrees of traffic increase due to merging and splitting of traffic. Assuming same elevation may significantly over-estimate the likelihood. Alternatively, we choose the below score definition, which maximizes the log-likelihood ratio for each edge separately to allow them to have different traffic elevation q. The score tests how likely every single flow along a path p is higher than their respective baseline values. Naturally, we could multiply the likelihood ratio score of each flow along p, which is equivalent to the sum of the log-likelihood ratio of every flow along p. Formally, it can be expressed as follows:

$$LLR(P) = \sum_{e \in P} LLR(e) \tag{2}$$

Definition 2.5. The most likely incoming (outgoing) path $P^*_{in}(e, r)$ ($P^*_{out}(e, r)$) between r and e is the incoming (outgoing) path with the maximum LLR score.

A grid location is likely to be the destination of a gathering event when the likelihood of every edge along its incoming paths having an elevated traffic volume is high, while the likelihood of every edge along outgoing paths having an elevated traffic volume is low. This is to ensure that we do not find false alarms such as an intersection where both incoming and outgoing paths have elevated traffic amount. Note that not all the significant flows in the study area should be linked with every potential destination. High traffic volume in a region is very unlikely to be part of a gathering event occurring



(a) Incoming and outgoing paths (b) A G-Graph rooted at r (c) A G-Graph rooted at r'

Fig. 1. Examples of Paths and G-Graphs

20 kilometers away in an urban environment. Here we define a maximum distance d, where only significant flows within distance d to the grid should be included.

The gathering score (GScore) of a grid is thus calculated by summing the log likelihood ratio of each distinct flow along incoming paths with length d or shorter, and subtracting the log likelihood ratio of each distinct flow along the outgoing paths with length d or shorter. Formally, the GScore is defined as follows:

$$GScore(r) = \sum_{e \in P^*_{in}(e_{in}, r)} LLR(e) - \sum_{e \in P^*_{out}(e_{out}, r)} LLR(e) \tag{3}$$

Definition 2.6. *Gathering Graph (G-Graph).* Given a root grid $r \in S$, and all the significant flows E_{sig} such that $dist(e_{sig}, r) \leq d$, $e_{sig} \in E_{sig}$ at time interval t, a Gathering Graph (G-Graph) rooted at r is a directed acyclic graph $G(r)$ whose vertices are the grids and edges are the flows, and $G = \bigcup_{e \in Esig} P^*_{in}(r, e)$. $G.GScore = GScore(r)$.

Given the above definitions, one may identify very similar G-Graphs rooted at grids close to each other which overlap heavily. Figure 1(b) and Figure 1(c) show two different G-Graphs with the same set of significant flows. Note the outgoing paths (green) are not part of the G-Graphs. These G-Graphs usually represent the same gathering event and may not provide much additional useful information. Also if two G-Graphs share a flow, it is hard to tell which root is the actual destination of this flow. To this end, we only find G-Graphs with the highest score among all the G-Graphs within $2d$ distance so that there is absolutely no overlapping among the G-Graphs.

Definition 2.7. Given two G-Graphs $G_1(r)$ and $G_2(r')$ with depth d, G_1 dominates G_2 if $G_1.GScore > G_2.GScore$ and $dist(r, r') \leq 2d$.

Finally, we would like to find a set of G-Graphs that are not dominated by others and have the highest scores among all the candidates. Hereby we define the dominant G-Graph set as follows:

Definition 2.8. A k-dominant G-Graph set $\hat{G}_k$ is a set of no more than k G-Graphs such that none of them dominate each other, and for any G-Graph $G' \notin \hat{G}_k$, one of the following conditions hold: (1) $G'.GScore < Min_{G \in \hat{G}_k}\{G.GScore\}$, or (2) $\exists G \in \hat{G}_k$ such that G dominates G'.

2.3. Problem Statement

Given the above definitions, the EDGE problem could be formulated as follows:

Given:

- A spatial field S with observed and baseline flows C, B during time interval t
- Maximum distance threshold d
- p-value threshold α
- Size of result k

Find:

- The k-dominant G-Graph set for time interval t

Objective:

- Reduce Computational Cost

Constraints:

- All the distances are measured in Manhattan distance
- Correctness and Completeness

3. COMPUTATIONAL SOLUTION

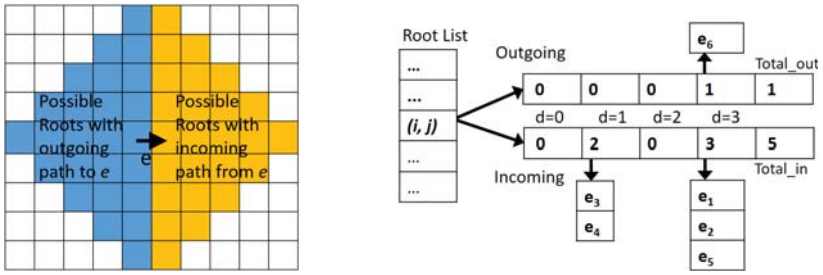
In this section we present the SmartEdge algorithm as a solution to the EDGE problem, which we proposed in [Zhou et al. 2016]. We also developed a brute-force algorithm to solve the EDGE problem, the details of which can be found in appendix A. Here, we first give an overview of the SmartEdge algorithm, then we explain each of the three design decisions we made, to design the algorithm. See appendix B For a theoretical analysis of the complexity of the SmartEdge algorithm.

3.1. The Smart Edge Algorithm

There are several computation bottlenecks in the Brute-Force algorithm. (1) A grid cell may be the root of a G-Graph only when there is at least one significant flow within distance d. All other cells can be ignored. (2) It is costly to exhaustively search for the most likely path of each significant flow to the root. (3) The algorithm does not have any ability to prune candidate G-Graphs. Since we are only interested in the top-k dominant G-Graphs, candidates with very low GScores or dominated by others should not be generated. To address the above computational bottlenecks, we present a new algorithm SmartEdge with three design decisions for better computational efficiency.

3.1.1. Candidate Root Filter. As previously mentioned, locations with no significant flows within distance d cannot be the root of a G-Graph. A location r is a possible root of the G-Graph that includes significant flow e_{sig} only if dist(r, e_{sig})$\leq$ d. Figure 2(a) shows an example where the yellow area are the possible root locations with an incoming path from significant flow e, and the blue area are the possible root locations with an outgoing path to e (assume $d = 4$). We create a data structure called the Candidate Root Index (CRI) with a hash table to store candidate roots with at least one significant flow around. Each root element is also linked with two vectors that store all the incoming and outgoing significant flows within distance d to the root. The time to locate significant flows when generating G-Graphs is reduced to constant. Since we will need the number of significant flows at each distance to the root in later calculations, we store the significant flows separately in $d + 1$ bins based on their distance to the root. When a significant flow e_{sig} is identified, the algorithm finds all the locations that could be the root of e_{sig}. For each of these candidates, the algorithm calculates its distance from e_{sig} and inserts e_{sig} into the corresponding bin. Locations with no significant flow around will not exist in the CRI thus won't be evaluated. Figure 2(b) shows the structure of the CRI.

3.1.2. Building G-Graph with Dynamic Programming. Due to the way the LLR score of a path is defined, the most likely path between a candidate root grid r and a significant flow $e = (s_a, s_b)$ could be calculated using a dynamic programming approach. The following optimal substructure can be observed. Assuming $r.x > s_b.x$, $r.y > s_b.y$, the most likely path P^* from grid s_b to root $r = (r.x, r.y)$ can be calculated by comparing the best paths to r via $(r.x-1, r.y)$ and $(r.x, r.y-1)$. This optimal substructure can be recursively used until the best path is found. Figure 3(a) shows an example where the best path connecting the significant flow (black arrow) and the root (red grid) is chosen between the two pink routes, either via cell s_1 or via s_2. Best paths from s_1 to r and from s_2



(a) Possible root locations of a significant flow (d=4)

(b) Candidate Root Index

Fig. 2. Candidate Root Filter Examples (best viewed in color).

ALGORITHM 1: Procedure: Build_Graph_DP(r, CRI[r])

Input: root grid r, Candidate Root Index entry CRI[r]
Output: G-Graph for r and its score

1 $G \leftarrow [\,]$; $GScore \leftarrow 0$; $P[2d+1][2d+1] \leftarrow \emptyset$
2 **for** $e_{sig} = (s_i, s_j)$ *in CRI*[r].*Sig_Flows* **do**
3 **for** $level$ = *1 to* $dist(s_j, r)$ **do**
4 **for** *grids* s *where* $dist(s, r) = level$ **do**
5 P[$s.x$][$s.y$].score$\leftarrow$ the max total score at s
6 P[$s.x$][$s.y$].next$\leftarrow$ the next grid in best path
7 $s_{now} = s_j$
8 **do**
9 $s_{next} \leftarrow$ P[$s_{now}.x$][$s_{now}.y$].next
10 **if** (s_{now}, s_{next}) *not in* G **then**
11 **if** e_{sig} *is incoming* **then**
12 $G = G \bigcup ((s_{now}, s_{next}))$
13 $GScore = GScore + LLR(s_{now}, s_{next})$
14 **else**
15 $GScore = GScore - LLR(s_{next}, s_{now})$
16 $s_{now} = s_{next}$
17 **while** (s_{now}*!=r)*
18 Return(G, GScore)

to r can be calculated in the same way recursively. Algorithm 1 presents the pseudo code of the new procedure Build_GGrap_DP. For a given root r and a list of significant flows nearby, the algorithm picks each significant flow and traverses all the cells in the rectangular area bounded by the root and this significant flow in a breadth-first manner, starting from the root (Line 2-7). The most likely path to the root from every grid cell along the way is calculated until the significant flow is reached. After finding the most likely path, all the flows along this path will be added to the G-Graph (Line 8-14). Figure 3(b) shows the breadth-first traversal order to calculate the best path to/from r.

3.1.3. G-Graph Pruning: GScore Upper-Bound. Finally we discuss how to efficiently prune candidate G-Graphs without generating them to save computation time. The G-Graph generation step takes up to $O(|E_{sig}|d^2)$ for each root even with the dynamic programming design decision, where $|E_{sig}|$ is the number of significant flows near the current

(a) An example of DP G-Graph generation optimal sub-structure

(b) Traversal order to generate best path to root for each location

Fig. 3. G-Graph Generation using Dynamic Programming.

root. In fact, most G-Graphs can be pruned because they are either dominated by a nearby G-Graph or have lower scores than all the top-k ones. Therefore, we propose two ideas (1) a GScore upper-bound for candidate roots, which is easy to compute, and (2) a pruning strategy for candidate G-Graphs. The upper-bound of the $GScore$ of $G(r)$ can be calculated as follows:

$$\widehat{GScore}(r) = N_e(r) \times \widehat{LLR(e_{ins})} + \sum_{e_{sig} \in E_{sig}} LLR(e_{sig}) \tag{4}$$

$N_e(r)$ is the upper-bound of the number of insignificant flows in G(r). $\widehat{LLR(e_{ins})}$ is the maximum possible LLR score of these insignificant flows.

Calculating $\widehat{LLR(e_{ins})}$: We design a zone-index to keep track of the maximum LLR(e) of insignificant flows. The entire spatial field $|S|$ is partitioned into zones of size $(2d+1) \times (2d+1)$. This size allows a G-Graph to fit in a single zone in the best case, while covers at most 4 zones in the worst case. Each zone keeps the maximum LLR of all insignificant flows inside it. Figure 4 shows an example of the zone-index with $d = 2$. There are four zones and the maximum LLR of insignificant flows, in each zone, are listed on the right. For example, the maximum LLR of insignificant flows pointing to right in zone 1 is 4.00. These numbers are obtained at the beginning of the algorithm when the LLR score of each directed edge is calculated. To calculate $\widehat{LLR(e_{ins})}$ for a candidate root r, we pull all the incoming significant flows near r from the CRI. For each significant flow e_{sig}, we check the directions and zones of insignificant flows needed to connect e_{sig} to r. Then the maximum LLR of these zones and directions are fetched. For example, e_1 of $G(r_1)$ can be connected to r_1 via insignificant flows pointing right or down in zone 1. The max LLR along $P^*_{in}(e_1, r_1)$ is thus $max\{2.50, 4.00\} = 4.00$. The same is done for all the significant flows near r_1, and $\widehat{LLR(e_{ins})}$ is the max of all the records pulled from the zone-index. It is possible that records in multiple zones are pulled for the same candidate root. In Figure 4, records in green and blue show the records fetched to calculate $\widehat{LLR(e_{ins})}$ of r_1 and r_2, respectively. The final results are 4.00 and 4.50 as they are the maximum records fetched for r_1 and r_2.

Calculating N_e: For every significant flow $e_{sig} = (s_i, s_j)$ within distance d to root r, the number of edges needed to connect it to r is $dist(s_j, r)$. Hence, a loose upper bound for N_e is the sum of the distance between each significant flow and r, i.e., $N_e \leq \sum_{e_{sig} \in E_{sig}} dist(s_j, r)$. However, this upper bound can be tightened since the most likely paths between the significant flows and the root may overlap thus reducing the possible number of distinct flows in G-Graph.

Let $N_e(i)$ be the maximum number of distinct flows of any G-Graph at distance i from the root r ($0 \leq i \leq d$). Due to the optimal substructure discussed previously, the

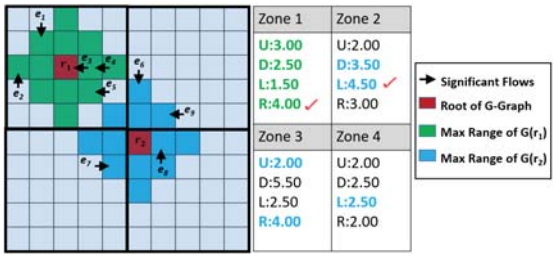


Fig. 4. $\widehat{LLR(e_{ins})}$ calculation with zone-index (d=2)

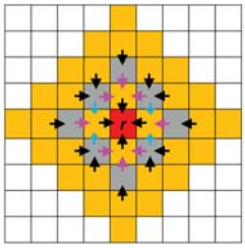


Fig. 5. Maximum number of edges, at distances $d = 0$ to $d = 2$

most likely path from each grid cell to the root r is unique. Thus the total number of distinct flows at distance i to r equals the number of grids at distance $i + 1$. Hence, $N_e(i) = 4d + 4$. Figure 3.1.3 shows the maximum possible number of edges at $d = 3$. There are 4, 8, and 12 possible edges within distance 0, 1, and 2 to r.

If the total number of significant flows that are i or further from r is more than $N_e(i)$, then only $N_e(i)$ flows can exist in the final G-Graph. Otherwise, the maximum number of flows at distance i equals the number of significant flows at least i away from r (denoted as $N_{sig}(i)$). Formally, $N_e = \sum_{i=0}^{d} Min\{N_e(i), N_{sig}(i)\}$. Each $N_{sig}(i)$ can be calculated by a linear scan of the bin sizes in the Candidate Root Index entry of root r in descending order at cost $O(d)$.

3.1.4. Candidate G-Graph Pruning Strategy. Based on the GScore upper-bound discussed above, we show how to prune G-Graphs as early as possible. In general, a candidate root is likely to have higher GScore and dominate G-Graphs around it, if there are more incoming significant flows near it. Thus we sort the candidate root index discussed in Section 3.1.1 based on the total number of incoming significant flows in descending order to increase the chance of visiting the dominating G-Graphs earlier. For each candidate root r in the candidate root index (CRI), a procedure *G_Prune* is called to decide if r should be added to the top-k list or pruned. Algorithm 2 presents its pseudo code. The priority queue Q, keeps the current top-k G-Graphs. The upper-bound $\widehat{GScore}(r)$, if not calculated yet, will be calculated and compared with the minimum GScore in Q (Line 3). If the upper-bound is lower, then r will be pruned (Line 4-6). Otherwise the actual G-Graph rooted at r will be generated using the *Build_Graph_DP* procedure and the actual GScore(r) will be compared with the minimum GScore in Q again. If GScore(r) is lower, then r will be pruned (Line 7-11). Otherwise, we may consider adding $G(r)$ into Q and pop the current k-th G-Graph. Before adding $G(r)$ to Q, there are some additional issues to consider. If one of r's neighbors (i.e., within distance $2d$) r' has a higher GScore, it means $G(r')$ dominates $G(r)$. Then, $G(r')$ should be added to Q and $G(r)$ will need to be removed from Q. Same for r'. If there is another G-Graph $G(r'')$ such that $G(r'')$ dominates $G(r')$ but does not dominate $G(r)$ then $G(r'')$ will be added to Q and $G(r')$ will need to be removed. In addition, $G(r)$ should be added back to Q now that its dominating G-Graph $G(r')$ is pruned. This may result in a long chain of G-Graphs with such dominating relationship from one end to the other. Adding and removing G-Graphs to and from Q takes substantial amount of computation time. To resolve this issue, we do not push $G(r)$ into Q until we can verify that $G(r)$ is not dominated by any other G-Graphs that are either already in Q or will be pushed into Q. Specifically, we examine every candidate root r' within $2d$ distance to r and get their GScore upper-bounds. If the upper-bounds of all these candidate roots is lower than $GScore(r)$, we prune all of them and push $G(r)$ into Q and pop the current k-th G-

ALGORITHM 2: Procedure: G_Prune

Input: root r, candidate root list CRI, priority queue Q
Output: Update Q and CRI

```
1  if r Not in CRI then
2      Return;
3  if GScore_up(r) <Min{Q.GScore} then
4      CRI.delete(r)
5      Return;
6  else
7      Build G(r) and calculate GScore(r)
8      if GScore(r) <Min{Q.GScore} then
9          CRI.delete(r)
10         Return;
11     else
12         for r' in CRI where dist(r, r') ≤ 2d do
13             if GScore(r') > GScore(r) then
14                 Build G(r') and calculate GScore(r')
15                 if GScore(r') > GScore(r) then
16                     M.push_back(r')
17         sort(M) on GScore(r') DESC
18         for r' in M do
19             G_Prune(r', CRI, Q)
20         if r in CRI then
21             Q.push_back([r, G(r), GScore(r)])
22             delete all r' where dist(r, r') ≤ 2d including r
23 Return;
```

Graph. If the upper-bound of $GScore(r')$ is higher than $GScore(r)$ then $G(r')$ is built and r' is pushed into a list M (Line 12-14). Then the procedure recursively calls itself to handle every r' in M. After all the roots in M are examined if r is still not pruned, we can be sure that $G(r)$ is not dominated by any other G-Graphs and thus can safely be added to Q (Line 17-18). The procedure *G_Prune* exists the recursion when the given root r is pruned or added to Q.

The full SmartEdge algorithm is presented in Algorithm 3. The CRI index and the zone-index are created and updated when the flows are examined to identify the significant ones (Line 2). Then the CRI is sorted based on total incoming significant flows in descending order (Line 3). Finally, each candidate root in CRI will be handled by the G_Prune procedure to generate the final results. There is no post-processing step needed for SmartEdge.

4. A COMPREHENSIVE EVALUATION OF THE PATTERN QUALITY

In this section we perform a comprehensive evaluation on the quality of the patterns discovered by our proposed method [Zhou et al. 2016]. First, we present a case study from our previous paper which used real trajectory data, to examine whether the algorithm is capable of detecting real-world gathering events. Second, we evaluate the performance of the algorithm in a more rigorous way, by developing an event simulator, which adds simulated events to the data. Simulation enables us to rigorously

ALGORITHM 3: Algorithm: SmartEdge

Input: Spatial field S, baseline and observed flows B, C, thresholds d, k, α
Output: The d-dominate G-Graph set of size k

```
Priority Queue Q[k]← 0
[CRI, Zone_Index]←find_sig_flow(B, C, α)
Sort CRI on number of incoming significant flows DESC
for r in CRI do
    G_Prune(r, Q, CRI, Zone_Index)
Output(Q)
```

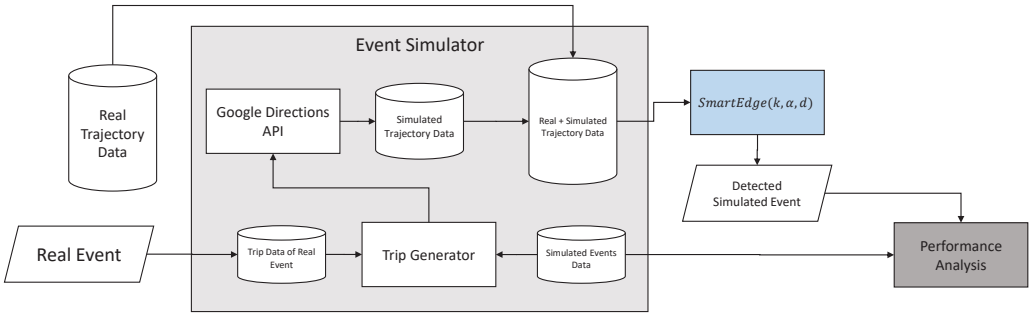
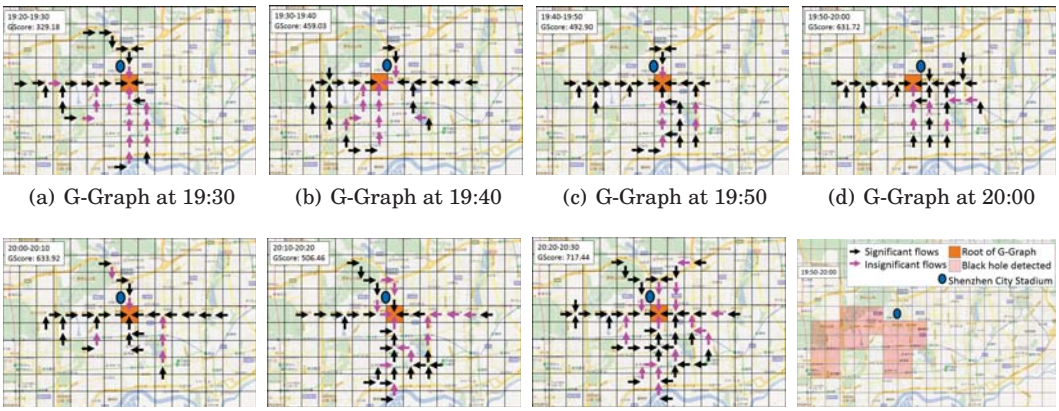


Fig. 6. Pattern Quality Evaluation with Event Simulator

analyze the performance of the algorithm, because it makes the ground truth of the gathering events available to us. Figure 6 shows how the simulator is used to conduct the performance analysis. The simulator takes the characteristics of a real event as input, such as travel patterns of vehicles to the event and number of arrivals. Then these characteristics are used to generate simulated trips to the specified time and location of a simulated event. This way, we make the characteristics of the simulated events consistent with a verified real gathering event. Then the trajectories of these trips, obtained from Google Directions API, are added to the real trajectory data and given to the SmartEdge algorithm as input. Then performance of SmartEdge is analyzed by evaluating the output of the algorithm.

4.1. The Dataset

Due to availability of advanced sensing technologies a verity of urban *big data* can now be recorded [Zheng et al. 2014]. The dataset we use contains detailed trajectories of over 10,000 taxis in Shenzhen, China, during the month of August, 2013. We partition the city into 500 meter by 500 meter grid cells. The whole city is thus partitioned into a 128 by 64 grid. Choosing smaller grid size will make it hard to track the taxis moving between adjacent grid cells due to the low GPS sampling rate (40 seconds). Then we count the total number of occupied taxis crossing each grid boundary during every 10-minute time windows and generated the flow function used in our problem. If the same taxi have two consecutive GPS points in two neighboring grid cells, we add 1 to the boundary flow count. The baseline flows (B) are generated by averaging the monthly average traffic flow crossing the same boundary during the same time of day. Since weekends and weekdays usually have different traffic patterns, we created separate baselines for Saturdays, Sundays, and other weekdays.



(a) G-Graph at 19:30 (b) G-Graph at 19:40 (c) G-Graph at 19:50 (d) G-Graph at 20:00

(e) G-Graph at 20:10 (f) G-Graph at 20:20 (g) G-Graph at 20:30 (h) Output of Related Work [Hong et al. 2015]

Fig. 7. Evolving of the Gathering Event near the Stadium

4.2. Gathering Event On Real Trajectory Data

We run the SmartEdge algorithm on the whole month's data and identify the top 5 most likely gathering events for every 10-minute time interval. The most interesting ones are verified by news and reported in this paper. The maximum distance threshold d is set to 10 grids (5km), and the statistical significance threshold α=0.005. On August 16 (Friday) at 8PM, there was a big charity pop concert held in the Shenzhen City Stadium. The stadium has a capacity of 35,000 and was almost filled up despite of the heavy rain that night [Sina Entertainment News 2013]. Several famous Chinese pop stars performed in the concert. Figure 7 shows the result of our algorithm on the same day in 7 consecutive 10-minute time intervals between 19:20 and 20:30. The most likely gathering destination (orange grid) is very close to the stadium (blue oval). The black arrows and pink arrows represent the identified significant flows and insignificant flows, respectively. The G-Graph near the stadium remains the most likely gathering event with highest GScore in all the 7 time intervals. The footprints clearly showed that big waves of audiences started to arrive half an hour before the concert started. Most traffic gathered towards the destination along the east-west road (Sungang Road) before 8PM. More traffic started to emerge from the South and North after the concert began. The GScore increase from 329.18 at 19:30 to 633.92 at 20:00 when the concert began, then dropped to 506.46 at 20:10 since a big wave of audiences had arrived before the concert began. Then it raised to 717.44 again, suggesting that another big wave of audiences are arriving at the stadium. The root moves to the east after 20:30 and the G-Graph vanished after 20:40.

We also implemented the method from a related work [Hong et al. 2015] and run it on the data for the same event to compare results. Since their black hole detection algorithm is designed for spatial networks, we treat each grid in our data as a road segment. The actual flow threshold is set to 50% of the actual flow of the grid we identified as the root between 19:50 and 20:00. Figure 7(h) show that their method discovered a black hole area with high net incoming traffic. However there is no directional information and no gathering destination can be identified.

Figure 8 shows a big picture with the top-5 most likely gathering destinations and their G-Graphs on August 16, 2013 between 7:30PM and 7:40PM. Grids of each G-Graph are highlighted using different colors. Besides No.1 (the stadium), No.5 is close to Shenzhen Railway Station, No.4 is a port from Shenzhen to Hong Kong. No.2 and

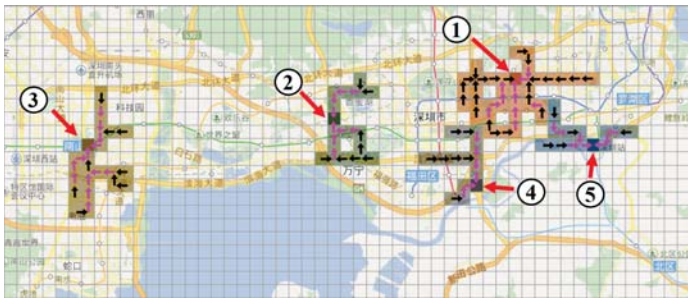


Fig. 8. Top-5 G-Graphs, 19:30-19:40, 8/16/2013

No.3 are close to major highway ramps and subway stations where congestion are likely to occur.

4.3. Detecting Simulated Gathering Events

The validity of the detected gathering events can sometimes be verified by looking into public records. For instance, for the gathering event detected in section 4.2, we were able to find news items that reported the actual event. However, this verification is not always possible. Measuring the accuracy of event detection methods is challenging, because the ground truth is not available most of the time [Zheng et al. 2015]. To tackle this challenge, we propose creating simulated gathering events and detecting them using the SmartEdge algorithm to quantify its performance in terms of accuracy and timeliness.

4.3.1. Simulation Setup. To understand the travel pattern of the taxis involved in a gathering event, we further analyze the concert event in Shenzhen Stadium, which we detected in section 4.2. We will use the results of this analysis to generate similar events at other locations and time slots. The grid cells inside the white box in figure 9 are considered as the event location in this analysis. Our analysis shows that most of the taxis arrived at the location one hour before and one hour after the start time of the concert. Figure 10 shows the scatter plot of the travel distance to the event based on arrival time and the cumulative distribution of number of arrivals based on arrival time to the event location. Each point refers to a taxi that started its trip from a grid cell with Manhattan distance specified on the y-axis, and arrived at the event location at the time specified on the x-axis. Event start time is denoted by 0. The total number of taxis that arrived at this location during this two-hour period is 1612. The average number of arrivals at this location during the same period is 442. Therefore, we observe an increase of 1170 in the number of arrivals.

To simulate gathering events, we specify five event locations. For each location, we specify two time-slots, one at noon on August 7, 2013 and one at 7 PM on August 6, 2013, resulting in simulation of 10 events in total. Figure 11 shows the geographical locations of the selected grid cells in the study area along with event numbers at each location. Then, for each event at location (a, b) and time t_e we add u taxis to u trip origins and have them travel to the specified event location. To specify each origin, we randomly select a point (t_a, l) from figure 10, then set the location of the origin to a random grid cell at Manhattan distance l from (a, b) and set its arrival time to $t_e + t_a$. To obtain the path from each origin to its destination, we use Google Maps Directions API, which returns the path as a sequence of latitude and longitude pairs along with the trip time. Then we convert the returned sequence into a sequence of grid cells, which in turn is converted into a sequence of edges between the grid cells. For each

Fig. 9. The analyzed event. Area inside the white box is considered the location of the event.

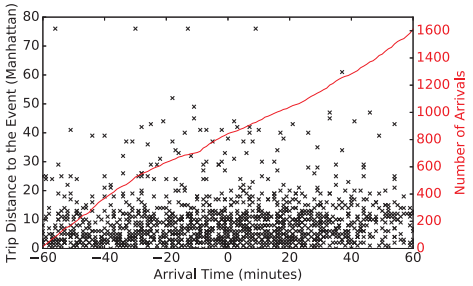


Fig. 10. trip distance based on arrival time and cumulative distribution of arrivals for the event detected in case study.

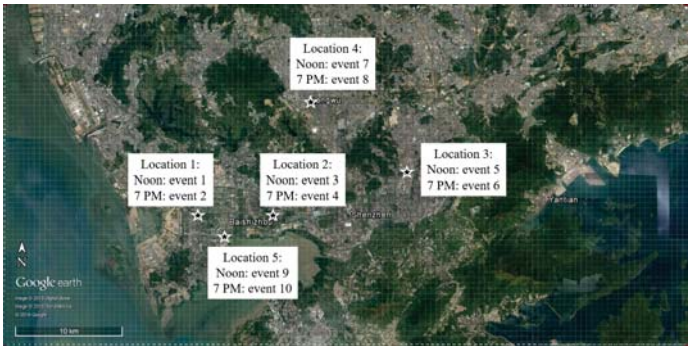


Fig. 11. Selected locations for the simulated events, and event numbers.

edge in the sequence, we increment its observed count C by 1, meaning that a vehicle traveled along these edges.

By performing this simulation we wish to measure: (1) how *accurately* the algorithm detects the event location, (2) how *early* the algorithm detects the event. Given parameters k as the results set size, and t as time, each run of the simulation returns a set of G-Graphs $Result(k,t)$. The *destination error* of a single run is defined as the Manhattan distance between the simulated event's location R, and the root of the closest G-Graph to R in $Result(k,t)$. Formally, the error is defined in equation 5.

$$e_{dest}(k,t) = min\{dist_M(R,R_k)|R_k \in Result(k,t)\} \tag{5}$$

In equation 5, $dist_M(R,R_k)$ is the Manhattan distance between grid cells R and R_k. Based on our analysis of the concert event, we set u to 1170 by default. Later we will also examine the effect of changing u. The default values of the rest of the parameters are $k = 10$, $d = 5$, and $\alpha = 0.005$.

4.3.2. Results with Default Parameters. First, for each event, we perform the simulation with default settings, 100 times, to evaluate how early the SmartEdge algorithm can detect events. Figure 12 shows how the average destination error of all events changes two hours before and two hours after the event. The error starts dropping at around 100 minutes before the event, meaning that the algorithm starts detecting G-Graphs related to the simulated event. 70 minutes before the event, when the vehicles start to arrive, the average error is very low (2.1). This means that on average, the algorithm

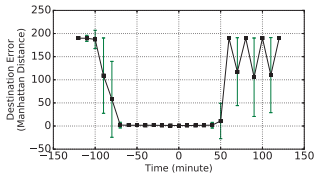
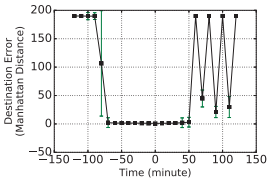
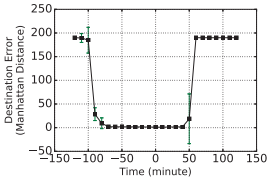


Fig. 12. Average destination error over time.

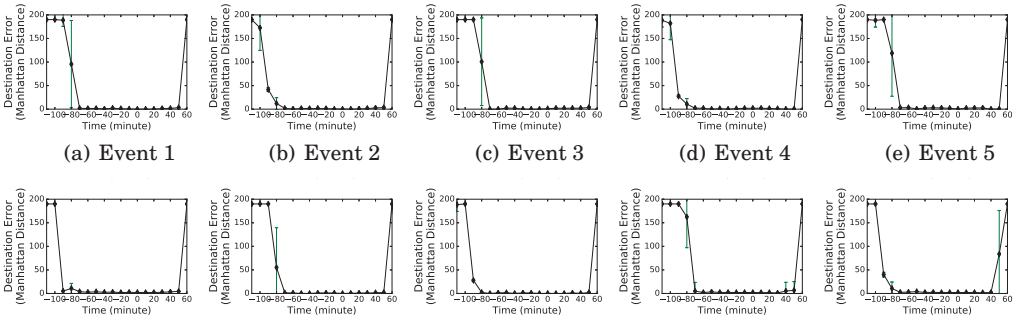


(a) Average error of events at noon



(b) Average error of events at 7 PM

Fig. 13. Average destination error over time, (a) events at noon, (b) events at 7 PM.



(a) Event 1 (b) Event 2 (c) Event 3 (d) Event 4 (e) Event 5

(f) Event 6 (g) Event 7 (h) Event 8 (i) Event 9 (j) Event 10

Fig. 14. Average destination error over time.

detects the event location around 2 grid cells away, even 10 minutes before the first vehicle arrives. The error remains below 2 until 50 minutes after the event.

Figure 13(b) shows the same plot for events simulated at noon and figure 13(a) shows it for simulated events at 7 PM. In both figures, we can see that the value of the error function starts dropping 100 minutes before the event.

Figure 14 (a) - (j) shows the value of the error function for each individual event. Each plot is the average of 100 simulations. For all of the events, destination error shows similar behavior by dropping sharply prior to the time of the event and staying low for the whole period in which simulated flow exists. The minimum average destination error is 0 with standard deviation 0 for events 1, 3, 7 and 8. These values mean that the algorithm was able to detect the exact location of these events all the time. The average error of all events reaches the minimum of 0.96 with standard deviation of 0.99. This value means that, on average, the algorithm detects the gathering events 0.96 grid cells away from their actual location.

4.3.3. Results of Varying The Added Traffic Amount. We also analyze the performance of the SmartEdge algorithm based on the amount of simulated flow added for each event. We repeated the above simulations by varying the number of vehicles added (u). Each time, we set u to 20%, 40%, 60% and 80% of the default number of vehicles, 1170. Thus we repeat the simulation with following values: 234, 468, 702 and 936. Figure 15 shows the destination error for each added count. We observe that the more simulated vehicles added for each event, the more accurate and timely the algorithm performs. In case of 20%, the algorithm detects G-Graphs related to the event with minimum error of 20.6, suggesting that the traffic is not enough to detect the location precisely. With 40%, the error drops sharply at -70, which is 10 minutes before the arrival of the first vehicle

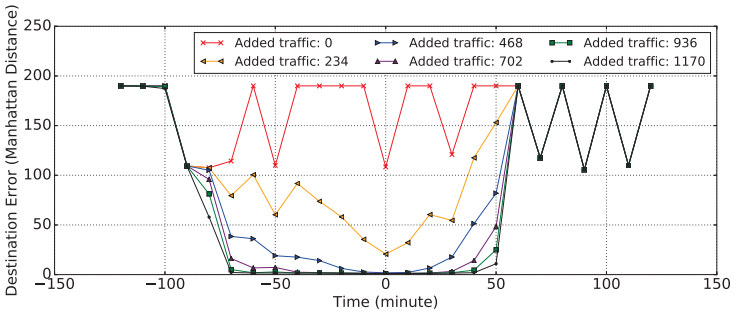
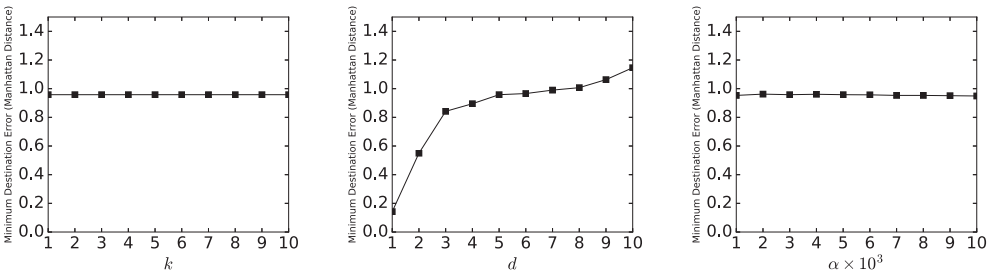


Fig. 15. Destination error through time, with different amount of simulated traffic



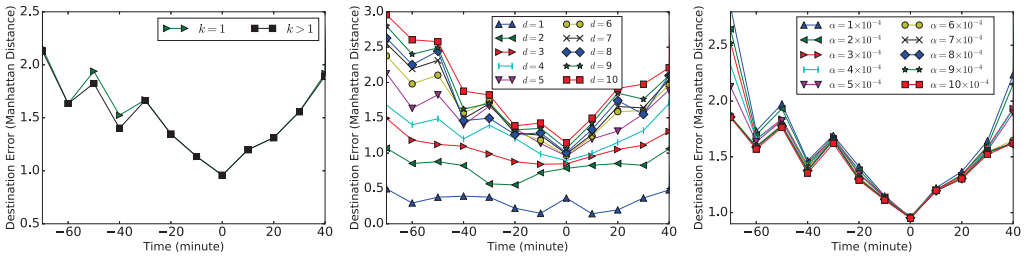
(a) Varying result size. (b) Varying maximum distance. (c) Varying significance threshold.

Fig. 16. Minimum average destination error by parameters.

to the event. The error reaches a minimum of 1.78 at time 0. With larger added counts (60% and 80%), we can also see a sharp drop at -70 minutes before the event. For 100%, the sharp drop happens even earlier, at -80, i.e. 20 minutes before the first arrival. The minimum error for 60% is 1.06, for 80%, it is 0.95, and for 100% it is 0.96, all of which happened at time 0.

4.3.4. Results of Varying SmartEdge Parameters. To understand the impact of varying parameters on the pattern quality, we perform the simulation with different parameter settings. Each time we vary one of the k, d and α parameters and fix others to their default values. Figure 16 shows the minimum average error for all events at each setting. Figure 16(a) shows the error with varying size of the top k results returned at each time step. The error remains constant and equals to 0.96, the same value obtained from the default settings. This means that the simulated events always resulted in G-Graphs that ranked as the top dominating G-Graph. Figure 16(b) shows the average minimum error by varying maximum distance d. With larger values of d the detected G-Graph will contain more information about the paths that lead the traffic to the event location. Figure 16(b) shows that the error increases slightly by increasing d, suggesting a trade-off between location accuracy and more paths information. However, the algorithm demonstrates robustness by keeping the error as low as 1.15 with $d = 10$. Figure 16(c) shows that varying α, the significance threshold, does not impact the ability of the algorithm to detect the simulated gathering events.

Figure 17 shows the error through time for different parameter settings. In figure 17(a), in time-steps -50 and -40 the error is slightly lower for $k > 1$. At all other times, the error is the same, meaning the performance of the algorithm, in terms of



(a) Varying result size. (b) Varying maximum distance. (c) Varying significance threshold.

Fig. 17. Destination error through time, with varying parameters.

timeliness, is not impacted by varying k. In figure 17(b) we are interested in the value of the error 70 minutes before the event, i.e. when the event location is first detected. With increasing d, the error at this time increases slightly. However, the algorithm is able to keep the error as low as 2.96 with $d = 10$, as early as 10 minutes before the first arrival to the event. Therefore, similar to the location accuracy, the algorithm is robust to varying d in terms of timeliness too. The impact of varying the significance threshold is shown in figure 17(c). The lower the threshold is, the later the error starts dropping. It means that the SmartEdge algorithm performs better, in terms of timeliness, if we have a looser significance threshold when we are identifying the significant flows.

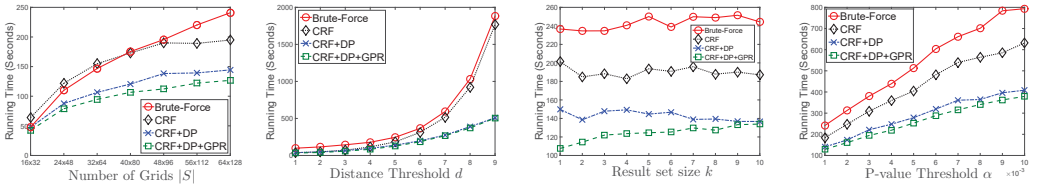
4.3.5. Summary. We found that the algorithm is able to specify the location of gathering events as early as 70 minutes before the event, i.e. 10 minutes before the arrival of the first vehicle. Also, the average error when detecting the event location is 0.96 grid cells. Moreover, the ability of the algorithm in terms of accuracy and timeliness increases when the traffic flow to the gathering event is larger. Parameter k does not impact the performance of the algorithm. The algorithm demonstrates robustness in terms of location accuracy and timeliness with varying d. With $d = 10$, the error remains as low as 1.15, and the location is still detected 10 minutes before the first arrival, while the pattern includes the information of paths coming from 10 grid cells away to the event location. Finally, with lower significance thresholds α, the SmartEdge algorithm performs better in terms of timeliness.

5. EVALUATION ON THE COMPUTATIONAL EFFICIENCY

This section presents our experimental evaluations on the efficiency of the proposed algorithms.

5.1. Experiment Settings

The dataset used is the same as the one in section 4. We compare the time cost of a Brute-Force algorithm (see appendix A) with three variants of the SmartEdge algorithm: (1) only with candidate root filtering (CRF), (2) CRF and G-Graph building with Dynamic Programming (CRF+DP), and (3) CRF, DP and the G-Graph Pruning (CRF+DP+GPR) which is the full SmartEdge algorithm. This way we examine how each design decision impacts the running time. The algorithms are run on the whole month's data. We filter time intervals with less than 5 G-Graphs and use the rest 1400 time intervals. In each case we run the same experiment 4 times and report the average CPU time. The experiment is run on a Linux with a single Intel Xeon E5-2650 2.00GHz CPU of a cluster with 32 identical CPUs, 20MB Cache for each CPU, and 32GB shared memory. Through the experiments we hope to answer the following questions: (1) How will the computation time change when varying the number



(a) Runtime with varying $|S|$ (b) Runtime with varying d (c) Runtime with varying k (d) Runtime with varying α

Fig. 18. Runtime Evaluation Results

of grids $|S|$? (2) How will the computation time change when varying the statistical significance level α? (3)How will the computation time change when varying k? The default parameters are: $|S| = 64 \times 128$, $d = 5$, $k = 5$, $\alpha = 0.01\% = 0.0001$.

5.2. Experiment Results

First we test the impact of varying the number of grids in the study area. We take a sub-area with the same side ratio from the center of S and grow it to fully cover S. The number of grids on the short side $L = \sqrt{|S|/2}$ is changed from 32 to 64 with a step of 8. The total number of cells changes from 32×64 to 64×128. Figure 18(a) shows that the Brute-Force algorithm runtime increases at linear speed while others increase at a sub-linear speed. This is because SmartEdge filtered impossible roots and G-Graphs but Brute-Force algorithm still examines all the locations in $|S|$. The full SmartEdge can achieve as much as 50% time savings over the Brute-Force algorithm.

We test the impact of varying d from 1 (500m) to 9 (4.5km). Increasing d will increase the time to generate G-Graphs. Figure 18(b) shows that both Brute-Force and CRF has an exponential increase speed since they enumerate all the possible paths when generating G-Graphs. The SmartEdge with DP G-Graph Building reduced the cost to super-linear on average case. SmartEdge can achieve on average 49% time savings over the Brute-Force algorithm. The time savings from GPR is 10% compared to the version without it.

Next we test the impact of varying the result set size k from 1 to 10. Increasing k will only impact the last algorithm (CPF+DP+GPR) since the other three don't use the top-k list to do any pruning. Figure 18(c) shows that SmartEdge with G-Graph Pruning (green line) has the best performance with smaller k. The savings is between 44% and 50% over the Brute-Force algorithm and 15% over the version without GPR. The gap between CRF+DP+ GPR (green) and CRF+DP (blue) becomes smaller when k increase as the minimum GScore in the top-k list is smaller thus reducing SmartEdge's pruning power.

Finally we test the impact of varying the p-value threshold α for significant flows from 0.0001 (0.01%) to 0.001 (0.1%) with a step 0.0001. Increasing α will increase the total number of significant flows N in the complexity. Figure 18(d) shows that the running time for all the four algorithms increase at a linear speed. However, CRF+DP and CRF+DP+GPR grow at a much slower speed. SmartEdge can achieve as much as 52% time savings over the Brute-Force algorithm and the GPR design decision provided 11% further improvement over the version without it.

6. RELATED WORK

Related work on event detection has focused on representing the footprint of events using regular shapes such as rectangles, circles, or undirected graphs. Based on how events are defined, these work can be further classified into two groups.

6.1. Count-Based Event Detection Methods

This group of related work identify regions where the total count of objects or instances (e.g., disease cases, vehicles) is higher than expected. Martin Kulldroff's spatial scan statistic [Kulldroff 1997] is widely used in epidemiology and many other areas. It detects circular areas in space, which have significantly high number of counts (e.g. disease cases) compared to the rest of the study area. Kulldroff extended his statistic to spatio-temporal setting [Kulldorff 2001; Kulldorff et al. 1998; Kulldorff et al. 2005], where cylinder-shaped clusters instead of circular clusters are used to account for the time span of the outbreaks. Neil et al modified Kulldorff's framework and proposed a statistic which detected emerging clusters, in which the risk inside cluster increases monotonically with time [Neill et al. 2005]. Neil also proposed an Expectation-based Poisson (EBP) model [Neill 2009]. Instead of comparing the risk inside a region against outside, EBP compares the observed count of a region with its own historical average. Neil et al. also proposed a Bayesian version of the spatial scan statistics [Neill and Cooper 2010; Neill et al. 2006] and later extended it to detect irregularly-shaped clusters [McFowland et al. 2013]. Zheng et al proposed a Latent Dirichilet Allocation topic modeling framework to detect spatiotemporal clusters while combining different types of data [Zheng et al. 2015] (e.g. bike rental data, emergency calls data, etc.). Their method solves the problem of sparse data by combining these different datasets.

Other count-based event detection methods include the papers using counts of geotagged tweets [Abdelhaq et al. 2013; Krumm and Horvitz 2015; Chen and Neill 2014; Zhao et al. 2015; Zhang et al. 2016b], counts from GPS records [Cao et al. 2010; Huang and Powell 2012; Pan et al. 2013] and records of bike sharing systems [Li et al. 2015]. Additionally, Lukasczyk et al [Lukasczyk et al. 2015] propose a novel visualization of hotspots' evolution over time using Reeb Graphs.

Additionally, some work focused on speeding up Kulldroff's algorithm. Neill et al. [Neill and Moore 2004] proposed an algorithm that divided rectangular areas into overlapping sub-rectangles and calculated an upper bound of the score in each sub-region for quick pruning. Later Neill used a subset scan technique to scan the spatial region in a time efficient way which worked with Kulldorff's statistic among others [Neill 2012]. Agarwal et al. [Agarwal et al. 2006] proposed a heuristic to approximate the discrepancy function assuming it is convex. Wu et al. [Wu et al. 2009] proposed a likelihood ratio test framework to find the most likely cluster in a grid with much lower computational cost than exhaustive search.

6.2. Volume-Based Event Detection Methods

The second group of related work focused on identifying regions where the incoming traffic is higher than outgoing traffic, or vice versa. Such regions are also called Blackholes and Volcanoes. Li et al. [Li et al. 2010; Li et al. 2012] proposed a framework to model blackholes and an algorithm to discover top-k blackhole subgraphs and applied the method to financial data in a purely spatial setting. Hong et al. [Hong et al. 2015] applied the same idea to an urban setting. Their work identifies sub-graphs with net traffic (incoming minus outgoing) higher than pre-defined thresholds. There are a few issues that were not addressed by these work: (1) they do not compare the observed traffic with any baseline thus may find trivial events such as morning rush hour congestion, (2) using the same net traffic threshold for the entire region and time period is inappropriate since traffic density are heterogeneous over space and time, and (3) the results of these methods do not reflect the direction and path of the gathering traffic.

All of the above related work find undirected footprints of events. While some of them can reflect the impacted area of an event (e.g., black holes), it is hard to tell how traffic flows and gathers inside the impacted region. Our work, by contrast, identify

the most likely gathering destination and the path along which moving objects gather towards the destination.

Other work includes modeling mobility networks such as [Zhang et al. 2015] which models the movement of objects in space using mobility graphs by determining the flow (counts) of moving objects in a network structure of places. [Zhang et al. 2016a] determines the flows using deep learning techniques. These works are not directly related to this paper because their goal is to determine the flows between places using different sources of data, instead of detecting events based on flows. Another type of work focuses on destination prediction based on sub-trajectories such as [Xue et al. 2013; Krumm 2006; Hendawi et al. 2015] which predicts the destination of an incomplete trajectory. Zheng presents a survey of trajectory mining techniques [Zheng 2015]. This work solves a different problem than this paper and also uses different type of data, i.e. individual trajectories rather than flows of moving objects. Works such as [Zheng et al. 2013; Zheng et al. 2014; Vieira et al. 2009; Gudmundsson and van Kreveld 2006] detect moving clusters, flocks, or dynamic groups of objects. They focus on mining the relationship among moving objects instead of footprints of events. Therefore, they are not directly related to the topic of this paper.

7. CONCLUSIONS

This paper investigated the problem of early detection of gathering events (EDGE). The EDGE problem is important to a broad range of applications in public safety and transportation management, yet it is computationally challenging. Related work did not consider the direction of gathering traffic flow thus can't precisely show the destination of the gathering. In our recent work [Zhou et al. 2016] we formulated the footprint of a gathering event as a Gathering Directed Acyclic Graph (G-Graph). An efficient algorithm SmartEdge was proposed to efficiently identify top-k non-overlapping G-Graphs for each time interval. In this paper, we propose a novel simulation-based evaluation approach to study the performance of the SmartEdge algorithm, i.e. the quality of the detected patterns in terms of timeliness and location accuracy. The results show that, the algorithm was able to detect patterns within one grid cell away (500 meters) from the simulated events on average, and the patterns were being detected as early as 10 minutes prior to the arrival of the first vehicle.

In the future, we plan to (1) apply the model in a spatial network settings, (2) consider the trajectories of individual vehicles instead of their aggregated flow, and (3) develop methods to predict gathering events, instead of early detection.

APPENDIX

A. A BRUTE-FORCE ALGORITHM TO SOLVE THE EDGE PROBLEM

The EDGE problem can be solved using straightforward ideas following the workflow discussed in Section 2.1. Algorithm 4 presents the pseudo code of a brute-force algorithm. First, each edge is examined and significant flows are identified (Line 1). Then the algorithm constructs the G-Graph rooted at each grid. For each candidate root grid r, all the significant flows within distance d are fetched. Then an exhaustive search is performed to find the most likely paths to/from r. The corresponding G-score is also calculated (Line 2-14). Finally the G-Graphs generated are sorted and scanned. Only the top k G-Graphs whose are not d-dominated by any others are reported (Line 15-20).

B. THEORETICAL ANALYSIS OF THE SMARTEDGE ALGORITHM

Both the Brute-Force algorithm and the SmartEdge algorithm need to go through each edge to identify significant flows, which takes $O(|S|)$ time. To build a G-Graph, the brute-force algorithm enumerates all the possible paths from each significant flow to

ALGORITHM 4: Brute-Force Edge Algorithm

Input: Spatial field S, baseline flows B and observed flows C, thresholds d, k, α
Output: The k-dominant G-Graph set

```
Scan all the edges to find significant flows sig_flow
for each r ∈ S do
    GScore ←0; visited[ ] ←[FALSE]; G_in ← [ ]
    for each e_sig in sig_flow where dist(r, e_sig) ≤ d do
        Enumerate all the path to find P*(r, e_sig)
        for each edge e ∈ P*(r, e_sig) do
            if !visited[e] then
                if P*(r, e_sig) is an incoming path then
                    GScore = GScore+LLR(e)
                    G_in.push_back(e)
                else
                    GScore = GScore−LLR(e)
                visited[e] ←TRUE
    Result.push_back([r, G_in, GScore])
sort Result on GScore DESC for i = 1 to k do
    for j = i + 1 to Result.end() do
        if Result[j] is d-dominated by Results[i] then
            Result.delete(j)
Output(Result)
```

the current root. There are up to $\binom{d}{d/2}$ routes and the total cost for this step is up to $O(Nd(d)^{d/2})$, where N is the number of significant flows near each root. This is done for all the $|S|$ grids. Then it sorts the G-Graphs based on their GScores and goes through the sorted list to find the k-dominant set, leading to a post-processing overhead of $O(k \cdot |S| + |S|log|S|)$. The overall time complexity is $O(|S|(Nd \cdot (d^{d/2})) + k \cdot |S| + |S|log|S|)$

The SmartEdge Algorithm only examines the possible roots of G-Graphs. We assume at least k G-Graphs can be built. Building the candidate root index will take $O(|S|)$ time. Using the dynamic programming design decision, a G-Graph can be built with $O(N + d)$ time (N for checking the zone-index to find $\widehat{LLR(e_{ins})}$, d for calculating N_e). In the best case, the first k G-Graphs found are actually the final results. All the other G-Graphs can be pruned without being generated. The total number of G-Graphs built is thus k. The total number of upper-bound calculated is $X - k$ where X is the total number of possible roots, $X < |S|$. In the worst case, the GScore upper-bound of every G-Graph need to be calculated but none of them could be pruned based on the upper-bounds. As a result, all the G-Graphs will have to be built. The total upper-bound calculation and G-Graph generation are both X. However, the SmartEdge algorithm does not have any post-processing overhead cost. Table I shows the comparison between the two algorithms in different scenarios.

REFERENCES

Hamed Abdelhaq, Michael Gertz, and Christian Sengstock. 2013. Spatio-temporal characteristics of bursty words in Twitter streams. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 194–203.

Deepak Agarwal, Andrew McGregor, Jeff M Phillips, Suresh Venkatasubramanian, and Zhengyuan Zhu. 2006. Spatial scan statistics: approximations and performance study. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 24–33.

Table I. Time Cost Comparison for the Algorithms

Steps	**Brute-force**	**SmartEdge best**	**SmartEdge worst**
G-Graphs Built	$\|S\|$	k	X
G-Graph Build cost	$O(Nd \cdot (d)^{d/2})$	$O(Nd^2)$	$O(Nd^2)$
Upperbounds Calculated	N/A	$X - k$	X
Upperbound calculation cost	N/A	$O(N + d)$	$O(N + d)$
Post-processing	$O(\|S\|log\|S\| + k \cdot \|S\|)$	None	None
Total	$O(\|S\|(log\|S\| + k + N(d)^{d/2})$	$O(\|S\| + kNd^2 + X(N + d))$	$O(\|S\| + XNd^2)$

Xin Cao, Gao Cong, and Christian S Jensen. 2010. Mining significant semantic locations from GPS data. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 1009–1020.

Feng Chen and Daniel B Neill. 2014. Non-parametric scan statistics for event detection and forecasting in heterogeneous social media graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1166–1175.

Joachim Gudmundsson and Marc van Kreveld. 2006. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*. ACM, 35–42.

Abdeltawab M Hendawi, Jie Bao, Mohamed F Mokbel, and Mohamed Ali. 2015. Predictive tree: An efficient index for predictive queries on road networks. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 1215–1226.

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.

Yan Huang and Jason W Powell. 2012. Detecting regions of disequilibrium in taxi services under uncertainty. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 139–148.

John Krumm. 2006. *Real time destination prediction based on efficient routes*. Technical Report. SAE Technical Paper.

John Krumm and Eric Horvitz. 2015. Eyewitness: Identifying local events via space-time signals in twitter feeds. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 20.

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.

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.

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.

Martin Kulldroff. 1997. A Spatial Scan Statistic. *Communications in Statistics - Theory and Methods* 26, 6 (1997), 1481–1496.

Yexin Li, Yu Zheng, Huichu Zhang, and Lei Chen. 2015. Traffic prediction in a bike-sharing system. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 33.

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.

Zhongmou Li, Hui Xiong, Yanchi Liu, and Aoying Zhou. 2010. Detecting blackhole and volcano patterns in directed networks. In *2010 IEEE International Conference on Data Mining*. IEEE, 294–303.

Jonas Lukasczyk, Ross Maciejewski, Christoph Garth, and Hans Hagen. 2015. Understanding hotspots: a topological visual analytics approach. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 36.

Edward McFowland, Skyler Speakman, and Daniel B Neill. 2013. Fast generalized subset scan for anomalous pattern detection. *Journal of Machine Learning Research* 14, 1 (2013), 1533–1561.

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

Daniel B Neill. 2012. Fast subset scan for spatial pattern detection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 74, 2 (2012), 337–360.

Daniel B Neill and Gregory F Cooper. 2010. A multivariate Bayesian scan statistic for early event detection and characterization. *Machine learning* 79, 3 (2010), 261–282.

Daniel B Neill and Andrew W Moore. 2004. Rapid detection of significant spatial clusters. In *Proc. Tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 256–265.

Daniel B Neill, Andrew W Moore, and Gregory F Cooper. 2006. A Bayesian spatial scan statistic. *Advances in neural information processing systems* 18 (2006), 1003.

Daniel B. Neill, Andrew W. Moore, Maheshkumar Sabhnani, and Kenny Daniel. 2005. Detection of Emerging Space-time Clusters. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*. ACM, New York, NY, USA, 218–227. DOI:http://dx.doi.org/10.1145/1081870.1081897

Bei Pan, Yu Zheng, David Wilkie, and Cyrus Shahabi. 2013. Crowd sensing of traffic anomalies based on human mobility and social media. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 344–353.

Sina Entertainment News. 2013. Shenzhen, A World Concert in China. (2013). http://ent.sina.com.cn/y/2013-08-20/15363991794.shtml.

Marcos R Vieira, Petko Bakalov, and Vassilis J Tsotras. 2009. On-line discovery of flock patterns in spatio-temporal data. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 286–295.

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].

Mingxi Wu, Xiuyao Song, Chris Jermaine, Sanjay Ranka, and John Gums. 2009. A lrt framework for fast spatial anomaly detection. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 887–896.

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.

Chao Zhang, Guangyu Zhou, Quan Yuan, Honglei Zhuang, Yu Zheng, Lance M. Kaplan, Shaowen Wang, and Jiawei Han. 2016b. GeoBurst: Real-Time Local Event Detection in Geo-Tagged Tweet Streams. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17-21, 2016*. 513–522. DOI:http://dx.doi.org/10.1145/2911451.2911519

Desheng Zhang, Juanjuan Zhao, Fan Zhang, and Tian He. 2015. coMobile: real-time human mobility modeling at urban scale using multi-view learning. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 40.

Junbo Zhang, Yu Zheng, and Dekang Qi. 2016a. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. *arXiv preprint arXiv:1610.00081* (2016).

Liang Zhao, Qian Sun, Jieping Ye, Feng Chen, Chang-Tien Lu, and Naren Ramakrishnan. 2015. Multi-task learning for spatio-temporal event forecasting. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1503–1512.

Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 242–253.

Kai Zheng, Yu Zheng, Nicholas J Yuan, Shuo Shang, and Xiaofang Zhou. 2014. Online discovery of gathering patterns over trajectories. *IEEE Transactions on Knowledge and Data Engineering* 26, 8 (2014), 1974–1988.

Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, 3 (2015), 29.

Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 3 (2014), 38.

Yu Zheng, Huichu Zhang, and Yong Yu. 2015. Detecting collective anomalies from multiple spatio-temporal datasets across different domains. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2.

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 (GIS '16)*. ACM, New York, NY, USA, Article 4, 10 pages. DOI:http://dx.doi.org/10.1145/2996913.2996998

Received January 2017; revised January 2017; accepted January 2017