Scalable Evaluation of Local K-Function for Radius-Accurate Hotspot Detection in Spatial Networks

Yongyi Liu, Yunfan Kang*
University of California, Riverside
Riverside, California
{yliu786,ykang040}@ucr.edu

Ahmed R. Mahmood
Google LLC.
Mountain View, California
amahmoo@google.com

Amr Magdy†
University of California, Riverside
Riverside, California
amr@cs.ucr.edu

Abstract

The widespread of geotagged data combined with modern map services allows for the accurate attachment of data to spatial networks. Applying statistical analysis, such as hotspot detection, over spatial networks is very important for precise quantification and patterns analysis, which empowers effective decision-making in various important applications. Existing hotspot detection algorithms on spatial networks either lack statistical evidence on detected hotspots, such as clustering, or they provide statistical evidence at a prohibitive computational overhead. In this paper, we propose efficient algorithms for detecting hotspots based on the network local K-function for predefined and unknown hotspot radii. The network local K-function is a widely adopted statistical approach for network pattern analysis that enables the understanding of the density and distribution of activities and events in the spatial network. However, its practical application has been limited due to the inefficiency of existing algorithms, particularly for large-sized networks. Extensive experimental evaluation using real and synthetic datasets shows that our algorithms are up to 28 times faster than the state-of-the-art algorithms in computing hotspots with a predefined radius and up to more than four orders of magnitude faster in identifying hotspots without a predefined radius.

CCS Concepts

• **Information systems** → **Geographic information systems**; • **Location based services**;

Keywords

Hotspot Detection, K-function, Spatial Network

ACM Reference Format:
Yongyi Liu, Yunfan Kang, Ahmed R. Mahmood, and Amr Magdy. 2023. Scalable Evaluation of Local K-Function for Radius-Accurate Hotspot Detection in Spatial Networks. In *The 31st ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '23), November 13–16, 2023, Hamburg, Germany.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3589132.3625646

*First two authors contributed equally to this research.
†This work is partially supported by the National Science Foundation, USA, under grants IIS-2237348, SES-1831615, and CNS-2031418, and the Google-CAHSI research grant.

SIGSPATIAL '23, November 13–16, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0168-9/23/11.
https://doi.org/10.1145/3589132.3625646

1 Introduction

The rapid growth of spatial data has sparked considerable interest in spatial analysis [2, 22, 38]. Spatial data is usually associated with spatial networks [5, 13, 27, 28, 31, 33, 37]. Spatial networks significantly influence various aspects of society, from daily commuting patterns [25] to the distribution of goods and services [10]. Hotspot detection along the spatial network is an important problem that identifies regions where objects are significantly concentrated. Compared to Euclidean space, hotspot detection proves to be more precise along spatial networks, given the fact that most human activities are centered around these networks [22]. Hotspot detection in spatial networks is applied in diverse fields such as traffic management [27], epidemiology control [28], and crime analysis [13]. In some cases, practitioners having specific domain knowledge aim to identify hotspots within predetermined radii. For example, in environmental studies [12], certain pollutants tend to affect a specific radius around the point of release, so, fixed-radius hotspots could be established at this distance to pinpoint high-risk areas. However, in many cases, the hotspot radius may not be known, e.g., identifying crime regions and traffic congestion. Therefore, most hotspot detection algorithms used in spatial networks automatically determine the appropriate radius through enumeration [32–34, 39].

Existing hotspot detection methods along spatial networks fall into two categories: clustering-based methods [6, 29, 31, 37, 41] and statistical-based methods [2, 22, 32–34, 39]. While clustering methods have efficient runtime, they might produce false-positive results [38]. In contrast, statistical-based methods offer a thorough statistical examination of the detected hotspots, typically by using a scoring function for quantification, e.g., log-likelihood score [33], density score [32, 34], to filter candidates. However, the scoring function doesn't directly give a statistical significance guarantee because it doesn't take into account the variability that could arise from random sampling. Consequently, randomization techniques such as Monte Carlo Simulation are followed to further validate the statistical significance. However, performing such simulations is expensive and can lead to performance issues.

The network local K-function [22, 40] is a statistical function for the analysis of the distribution of objects in the spatial network. For a given center, it computes the sum of edge weights and the number of reachable objects in the network within a specific distance from the center and derives a distribution function. Consequently, the network local K-function can be used directly to compute a measure of statistical significance due to its incorporation of the distribution function, without using randomization techniques for statistical validation. This makes its computation efficient and solution quality statistically supported. Consequently, the network local K-function can efficiently and effectively detect hotspots in spatial networks.

Detecting hotspots in spatial networks is challenging for several reasons. Firstly, modern network datasets are extensive, including up to millions of objects [24], whereas existing statistical methods for network hotspot detection are typically designed to handle smaller datasets, usually thousands of objects [32–34]. Second, the distances between objects are constrained by the network topology, where edges in the network are intricate and interconnected, making the distance computation more complicated compared to that in Euclidean space. Third, determining the range of each hotspot is computationally expensive. Existing statistical-based methods for hotspot detection are mainly based on enumeration [15, 32–34], which does not scale due to the vast candidate size involved.

In this paper, we address the limitations of the current hotspot detection algorithms in spatial networks based on the network local K-function. Our contribution can be summarized as follows:

- We propose two problems on spatial networks: Hotspot Detection with Predefined Radius (HDPR) and Hotspot Detection Without Predefined Radius (HDWPR) based on the network local K-function.
- We propose efficient algorithms to solve the HDPR problem.
- We define optimal hotspots radii and propose an approximate algorithm to solve the HDWPR problem.
- We present complexity analysis and correctness proof for the proposed algorithms.
- We conduct extensive experimental evaluations on both real and synthetic spatial network datasets to demonstrate the efficiency of our methods.

The rest of the paper is organized as follows: Section 2 presents the related work. Section 3 defines the problems. Section 4 and Section 5 detail our proposed algorithms. Section 6 presents the experimental evaluation and Section 7 concludes the paper.

2 Related Work

In this section, we discuss the work related to hotspot detection in spatial networks and the evaluation of the network K-function.

Hotspot detection: Spatial hotspot detection is an important problem with numerous applications, e.g., identifying regions with high occurrences of diseases in the study of epidemiology [17, 30], pinpointing locations with a high concentration of accidents or congestion in traffic management [18, 27], and revealing areas with high crime rates in public safety [1]. Hotspot detection has been studied extensively in the Euclidean space [38]. However, in this paper we focus on hotspot detection in spatial networks as it provides accurate insights based on actual edge distances.

Consider a set of events happening on a spatial network, a *hotspot* [33] in a spatial network denotes a region in the network that exhibits a high density of events compared to the mean number of events. There are two main approaches for identifying hotspots in spatial networks: (1) *clustering-based hotspot detection*, and (2) *statistical analysis-based hotspot detection*. Clustering-based hotspot detection [6, 29, 31, 37, 41] aims to identify groups of events that are close to each other in the network space. Extensions of traditional clustering approaches have been developed to support spatial networks, e.g., partition-based clustering [16, 41], DBSCAN-based [8, 37, 41], and hierarchical-based clustering [41]. While clustering-based hotspot detection algorithms are considered computationally efficient, they lack strong statistical robustness, i.e., they can result in biased or misleading results [38]. For example, partition-based clustering [41] and DBSCAN-based clustering [37] algorithms will always return clusters even if the points are randomly distributed. These clusters are not statistically robust and are considered false positives for statistical methods.

Statistical-based hotspot detection [2, 22, 32–34, 39] defines hotspots based on statistical models. These models identify regions where the density of events significantly exceeds the expected density assuming a specific event distribution. Khalid et al. [13] use the NetKDE [18] to detect crime hotspots and the Getis-Ord GI* [9] is adopted as the statistical evidence of hotspot properties. However, this method is rather computationally expensive for large networks as it needs to compute the pairwise correlation between events. Tang et al. [33], propose NPP, which is the most relevant statistical method for object-centered hotspot detection in spatial networks. NPP assigns a log-likelihood score for all possible radii of hotspots, and uses network partitioning to prune spatial objects that do not contribute to hotspots. Candidate hotspots are validated using Monte Carlo simulation trials. While NPP provides strong statistical evidence for detected hotspots, NPP is computationally expensive because of the large number of Monte Carlo simulation trails and the scoring of multiple radii per hotspot.

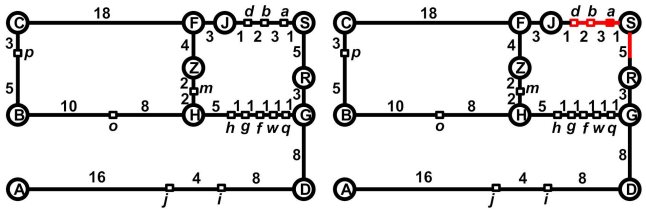
The **network K-function:** [22] is proposed as a robust measure for density and distribution of spatial patterns in the network space. It is divided into two categories: the *network global K-function* [5, 21, 22] and the *network local K-function* [22, 40]. Chan et al. [5] introduced the state-of-the-art algorithm for the evaluation of the network global K-function, which counts the object pairs within a specific distance threshold. It speeds up the processing by sharing the processing of nearby objects, i.e., *neighborhood sharing*. One important limitation of the network global K-function is its inability to identify objects that are part of dense clusters.

Recently, the network local K-function [22, 40] has been introduced to better analyze the densities and distribution of spatial objects surrounding a center object. It calculates the total number of spatial objects that are within a specific network distance from the center object as well as the sum of network distances between those spatial objects and the center object. The main advantage of the network local K-function is that it provides strong statistical evidence on the densities of spatial objects in the network.

In this paper, we address the problem of efficient hotpot identification with strong statistical evidence using the network local K-function. Notice that, existing statistical-based hotspot detection algorithms [32–34] enumerate and score a large number of hotspot radii. This is computationally expensive, even with the proposal of pruning algorithms [33] that reduce the search space of hotspot radii. On the contrary, we identify hotspots with and without a predefined radius while avoiding the expensive enumeration. This significantly improves efficiency without affecting utility.

3 Preliminaries and Problem Definition

In this section, we formally define the terminologies used and the problems addressed in the paper.



(a) Spatial Network (b) Isodistance sub-network

Figure 1: Spatial Network Example

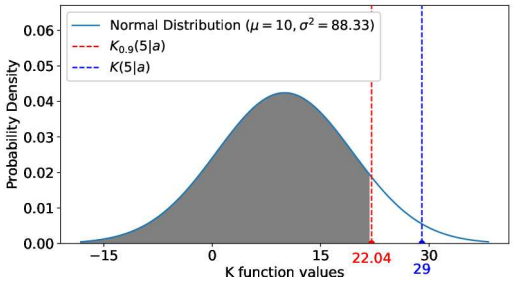


Figure 2: Probability Distribution Function

A ***spatial network*** [22], denoted as $G = (V, E)$, is an undirected weighted graph where V is a set of vertices and E is a set of edges. V has two types of vertices: (1) ***location vertices*** that represent physical locations, e.g., road intersections and turning points, and (2) ***event vertices*** that represent spatial events, e.g., vehicle locations, 911 calls, or traffic collisions. Practically, the set V is the union of two sets: a set of location vertices and a set of event vertices. The spatial events are overlaid over the physical straight lines that connect physical locations. E represents the set of connections among graph vertices. $w(u, v)$ represents the weight of the connection (u, v) and it is the network travel distance between vertices u and v. Every connection $e \in E$ is either: (1) an ***edge***, or (2) a ***segment***. An edge is a direct connection between two location vertices. A segment is a direct connection that connects an event vertex to another event vertex or to a location vertex. Each segment is overlaid on an edge and each edge is associated with zero or more overlaid segments. Figure 1a gives a spatial network where location vertices and event vertices are denoted by uppercase and lowercase letters, respectively. In Figure 1a, Edge (F, J) is an edge associated with zero segments, while Edge (J, S) is an edge associated with four segments (J, d), (d, b), (b, a), and (a, S). E maintains both edges and their overlaid segments as graph connections in one set differentiated with a flag attribute indicating their type. We denote the shortest path distance between any pair of vertices x and y, as $dist(x, y)$, where x and y can be location or event vertices. In Figure 1a, $w(J, S) = 7$, and event vertex o is overlaid on Edge (B, H) with $dist(o, H) = 8$, $dist(o, B) = 10$, and $dist(o, m) = 10$.

The number of overlaid event vertices along an Edge e in G is denoted as $|e|$. The number of location vertices and event vertices in G are denoted as $|V_l|$ and $|V_e|$, respectively, so $|V| = |V_l| + |V_e|$. The number of edges and segments are denoted as $|E_e|$ and $|E_s|$, respectively, so $|E| = |E_e| + |E_s|$.

An ***isodistance sub-network*** [33] $I(t|\vartheta)$ of an event Vertex ϑ with a given distance t is a network that consists of vertices, edges, and segments that are reachable from ϑ within a distance t. Edges and segments could be fully or partially covered by $I(t|\vartheta)$.

$L(t|\vartheta)$ is the sum of weights of all the edges and segments in $I(t|\vartheta)$ and $L(G)$ is the total weight of edges in G. If an edge or a segment is partially covered by $I(t|\vartheta)$, only the covered parts are considered in $L(t|\vartheta)$. The weights of edges and their overlaid segments are not added twice. The number of event vertices in $I(t|\vartheta)$ is denoted as $N(t|\vartheta)$. Figure 1b shows an example of $I(5|a)$ where all the edges, segments, and the event vertices encountered in $L(5|a)$ are highlighted in red. In this example, $L(5|a) = 10$ and $N(5|a) = 3$ (counting event vertices a, b, d). In Figure 1a, $L(G) = 116$. Whenever there is no ambiguity, $L(t|\vartheta)$ is abbreviated as L and $N(t|\vartheta)$ is abbreviated as N.

The ***network local K-function*** [22, 40] is a statistical function to analyze the density and distribution of events surrounding a center vertex. This function takes two inputs, an event Vertex ϑ and a network distance t. Formally,

$$K(t|\vartheta) = \frac{N(t|\vartheta)}{\rho}$$

where $\rho = \frac{|V_e|-1}{L(G)}$ represents the overall density of event vertices within the entire network. In Figure 1a, $\rho = \frac{13-1}{116} = 0.103$, and $K(10|a) = \frac{3}{0.103} = 29$.

Complete spatial randomness (CSR) hypothesis [22] is a fundamental spatial statistical hypothesis that describes random occurrence of spatial events. CSR states that random locations of spatial events follow a homogeneous binomial point process [22]. We use the CSR hypothesis to represent a hypothetical distribution of spatial events surrounding a center vertex without hotspots. The expectation μ and variance σ^2 of the hypothetical spatial distribution surrounding ϑ are calculated as follows:

$$\mu(K(t|\vartheta)) = L(t|\vartheta)$$

$$\sigma^2(K(t|\vartheta)) = \frac{1}{|V_e| - 1} * L(G) * L(t|\vartheta) * (1 - \frac{L(t|\vartheta)}{L(G)})$$

In Figure 1b, $\mu(K(5|a)) = L(5|a) = 10$, and $\sigma^2(K(10|a)) = \frac{1}{13-1} * 116 * 10 * (1 - \frac{10}{116}) = 88.33$.

For a specific radius t, an event Vertex ϑ is a **hotspot** when the actual density of events within t distance from ϑ exceeds the expected density. We find the quantile of the actual local K-function $K(t|\vartheta)$ compared to the hypothetical distribution. If it is above a preset threshold, e.g., 90%, ϑ is considered a hotspot [22]. Formally, ϑ is a hotspot when

$$K(t|\vartheta) > K_\alpha(t|\vartheta)$$

where α is a preset quantile threshold that serves as the statistical evidence of the identified hotspot. $K_\alpha(t|\vartheta)$ represents the α quantile of the $K(t|\vartheta)$ distribution function.

Figure 2 represents the CSR-based probability distribution of events surrounding Vertex a in Figure 1b and radius $t = 5$, given $\mu(K(5|a)) = 10$, and $\sigma^2(K(5|a)) = 88.33$. The 22.04 is the 90% quantile of the hypothetical local K-function. The actual value of the local K-function, i.e., $K(5|a)$ is 29. Hence a is considered a hotspot with confidence exceeding 90%. The quantile of a is 97.83% which represents an even higher statistical significance of the hotspot a. Higher

quantile values (>90%) make hotspot detection more stringent [40], which improves the confidence in the detected hotspots and avoids false positives [38]. Based on the above discussion, we define the following problems for hotspot detection in spatial networks.

Problem 1. ***Hotspot Detection with Predefined Radius (HDPR):*** *Given a spatial network $G = (V, E)$, a minimum statistical significance threshold α, and a distance value t, find the set of event vertices V_H, such that $\vartheta \in V_H$ if ϑ satisfies $K(t|\vartheta) > K_\alpha(t|\vartheta)$.*

In HDPR, hotspot detection is based on a predefined network distance, t, and a statistical significance threshold, α. Any event Vertex ϑ is identified as a hotspot if $K(t|\vartheta) > K_\alpha(t|\vartheta)$. However, in some situations, users may not have a specific idea about the best radius to choose for hotspots. Hence, we define the following problem for hotspot detection that returns hotspots alongside their radii. Instead of a predetermined radius, each hotspot is characterized by an optimal radius discovered by the algorithm.

Problem 2. ***Hotspot Detection Without Predefined Radius (HDWPR):*** *Given a spatial network $G = (V, E)$ and a minimum statistical significance threshold α_{min}, find a set of hotspots H, each hotspot $h \in H$ is defined with a triple $(\vartheta_h, t_h, \alpha_h)$ where $\vartheta_h \in V$ is an event vertex with radius t_h and statistical significance α_h, where $K(t_h|\vartheta_h) > K_{\alpha_{min}}(t_h|\vartheta_h)$ and $\alpha_h \geq \alpha_{min}$. ϑ_h is considered the center of a hotspot with radius t_h and statistical significance α_h.*

4 Hotspot Detection with Predefined Radius (HDPR)

In this section, we present our solution to HDPR problem, i.e., hotspot detection with predefined radius. The key idea is to efficiently use the network local K-function at a specific distance and statistical significance to detect hotspots. This requires efficient computation of N and L for every vertex. For Vertex ϑ with network distance t, $N(t|\vartheta)$ represents the number of event vertices reachable from ϑ within distance t, and $L(t|\vartheta)$ represents the sum of edge weights within distance t. There are three main components of our algorithms for HDPR: (1) *Graph Traversal*, (2) *Batch-Based Traversal*, and (3) *Incremental Batched Traversal*.

4.1 Graph Traversal (GT)

GT is the core technique for computing N and L values for any event vertex. GT initiates a Dijkstra shortest path-based traversal [7] that originates from an event node, say ϑ. The objective is to count events that are within a shortest path distance t and use this count to compute $N(t|\vartheta)$ and $L(t|\vartheta)$ and hence the network local K-function value for ϑ. GT starts with setting $N(t|\vartheta)$ and $L(t|\vartheta)$ to 0. Initially, the isodistance sub-network of ϑ is not discovered. GT incrementally explores edges and segments and keeps track of all vertices (event and location) that are reachable from ϑ sorted based on their shortest path distances. Then, GT visits Vertex u that is closest to ϑ. If u is an event vertex, then $N(t|\vartheta)$ is incremented by 1. To update $L(t|\vartheta)$, GT checks the weight u can contribute to the neighboring Edge or Segment (u, v), which is $min(t - dist(\vartheta, u), w(u, v))$. If $t - dist(\vartheta, u) \geq w(u, v)$ then (u, v) is *fully traversed*. Otherwise, (u, v) is *partially traversed*. This means Edge/Segment (u, v) partially contributes to $L(t|\vartheta)$. We keep track of the partial contribution of (u, v) to $L(t|\vartheta)$, i.e., $P(u, v) = t - dist(\vartheta, u)$. GT keeps iterating until all vertices within the distance t from ϑ are visited.

Running Example: In Figure 1, we want to check if i is a hotspot with $t = 10$ and significance level $\alpha = 90\%$. To compute $N(t|i)$ and $L(t|i)$, we traverse the graph starting from i. j is the first vertex visited and $N(t|i)$ is incremented to 2 (from Events i, and j). $L(t|i) = 10$ because (i, j) is fully traversed and Segment (j, A) is partially traversed with $P(j, A) = 6$. Then, Location Vertex D is visited, with $L(t|i) = 20$. This is because Segment (i, D) is fully traversed and Edge (D, G) is partially traversed with $P(D, G) = 10 - 8 = 2$. Hence $K(10, i) = \frac{N(t|i)}{\rho} = \frac{2}{0.103} = 19.33$ and that is a significance of level of 47.9%. Recall that $\rho = \frac{|V_e|-1}{L(G)} = \frac{12}{116} = 0.103$. This indicates that i does not qualify as a hotspot.

4.2 Batch-Based Traversal (BBT)

While GT accurately computes N and L for each event vertex, GT is inefficient as events are processed individually. We speed up GT by adding batch processing of events, i.e., Batched-Based Traversal (BBT). BBT computes N and L values of events that belong to a single Edge e in a single batch. The key idea here is that events on the same edge are close to each other and have the same neighboring events. This allows sharing the computation among events on the same edge. The objective in BBT is to compute N and L for the events along e in a single batch. This is done by traversing each edge in G and evaluating the edge's contribution to N and L of events along e. To speed up the processing of event batches within an edge, events within an edge are sorted according to their distances to one of the edge's endpoints, and the list of sorted events on e is stored as S_e. In the following, we discuss how to batch compute the N and L of events along Edge $e = (P, Q)$.

First, we evaluate the contribution of events of Edge e to their own N and L values. If $t > w(e)$, then we increment N by $|e|$, i.e., the number of events along e, and L by $w(e)$, i.e., the weight of Edge e, for all events. The reason is that all events on e are within distance t of each other. Otherwise, we apply a sliding-window-based approach that incrementally computes the N and L for each event. The length of the window is $2t$ to account for the situation where an event is in the middle of Edge e and there are events on both sides of this event on e. Initially, the sliding window begins at the first event in S_e, say ϑ. $N(t|\vartheta)$ and $L(t|\vartheta)$ are updated according to the vertices, edges, and segments covered in the window. When the processing of ϑ is complete, the window slides towards the next event in S_e. This results in the addition of new segments to the window and the removal of other segments. The N and L of the new event can be updated according to the N, L of ϑ, and the difference in the window segments. The window keeps sliding until all events along e are processed. Figure 3 shows an example.

Then, we categorize any Edge, say e_1, from graph G into the following categories:

(1) **Category1:** *All* events in e_1 are within t distance from all events in e. N and L for events on e are batch increased by $|e_1|$ and $w(e_1)$, respectively.
(2) **Category2:** *No* events in e_1 are within t distance from events in e. In this case, e_1 does not affect the values of N and L for events on e.

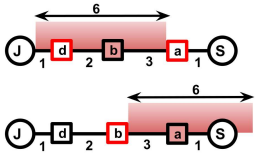


Figure 3: Sliding Window

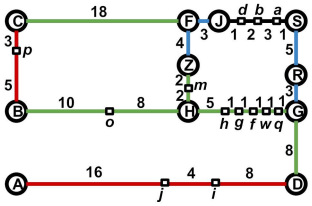


Figure 4: Edge Categories

(3) **Category3:** *Some* events in e_1 are within t distance from events in e.

The processing of Categories 1 and 2 has been described above. To present our solution to Category 3, we define the notions of maximum distance and minimum distance between edges. For an Edge $\tilde{e} = (\tilde{P}, \tilde{Q})$, we define the minimum distance between e and $\tilde{e}$, as the minimum distance between any endpoint in e and any endpoint in $\tilde{e}$. Formally,

$$D_{min}(e, \tilde{e}) = min\{dist(P, \tilde{P}), dist(P, \tilde{Q}), dist(Q, \tilde{P}), dist(Q, \tilde{Q})\}$$

The maximum distance between e and $\tilde{e}$ is the upper-bound on the shortest network travel distance between any event in e and any event in $\tilde{e}$. Formally,

$$D_{max}(e, \tilde{e}) = D_{min}(e, \tilde{e}) + w(e) + w(\tilde{e})$$

In Figure 4, $D_{min}((H, G), (A, D))$ is the distance between D and G, which equals to 8. $D_{max}((H, G), (A, D)) = D_{min}((H, G), (A, D)) + w(H, G) + w(A, D) = 8+10+28 = 46$. For Category 3, $D_{min}(e, \tilde{e}) \leq t$ and $D_{max}(e, \tilde{e}) > t$ and some events in $\tilde{e}$ may not be reachable from events on e. This requires events in e to be examined individually.

In this case, BBT sequentially visits the events along e, i.e., $S_e[i]$ for $i = 1, ..., |S_e|$. For each event, $S_e[i]$, along Edge e, we calculate the shortest distance from $S_e[i]$ to $\tilde{P}$, denoted as $dist(S_e[i], \tilde{P})$, where $dist(S_e[i], \tilde{P}) = min(dist(S_e[i], P) + dist(P, \tilde{P}), dist(S_e[i], Q) + dist(Q, \tilde{P}))$.

Then we compute $t - dist(S_e[i], \tilde{P})$, which is the remaining weight that $S_e[i]$ can disseminate along Edge $\tilde{e}$ from $\tilde{P}$ to $\tilde{Q}$. Similarly, we compute $t - dist(S_e[i], \tilde{Q})$, which is the remaining weight that $S_e[i]$ can disseminate along Edge $\tilde{e}$ from $\tilde{Q}$ to $\tilde{P}$.

Denote $r_i(\tilde{P}) = max(t - dist(S_e[i], \tilde{P}), 0)$ and $r_i(\tilde{Q}) = max(t - dist(S_e[i], \tilde{Q}), 0)$, we define the following scenarios based on the relationship between $r_i(\tilde{P}) + r_i(\tilde{Q})$ and $w(\tilde{e})$:

(1) **Case 1:** $r_i(\tilde{P}) + r_i(\tilde{Q}) \geq w(\tilde{e})$. In this case, $S_e[i]$ can reach any place on $\tilde{e}$ within t distance. $L(t|S_e[i])$ is incremented by $w(\tilde{e})$ and $N(t|S_e[i])$ is incremented by $|\tilde{e}|$.

(2) **Case 2:** $r_i(\tilde{P}) + r_i(\tilde{Q}) < w(\tilde{e})$. In this case, $L(t|S_e[i])$ is incremented by $r_i(\tilde{P}) + r_i(\tilde{Q})$. As for $N(t|S_e[i])$, since the events along each edge are sorted, we perform a binary search with key equals to $r_i(\tilde{P})$ to retrieve the number of events on $\tilde{e}$ reachable from $\tilde{P}$ to $\tilde{Q}$, and update $N(t|S_e[i])$. Similarly, we perform another binary search with key equals to $r_i(\tilde{Q})$ to retrieve the number of events reachable from $\tilde{Q}$ to $\tilde{P}$ and update $N(t|S_e[i])$.

BBT creates a set $E_{e,t}$ to store edges such that $\forall \tilde{e} \in E_{e,t}$, $D_{min}(e, \tilde{e}) \leq t$. Then, BBT computes the set of location vertices reachable from P or Q within t shortest distance, denoted as $D_{P,Q}$, using the Dijkstra single source shortest path algorithm [7]. Finally, BBT iterates over location vertices in $D_{P,Q}$. For each vertex in $D_{P,Q}$, denoted as $\tilde{P}$, we add every edge connected to $\tilde{P}$, say $\tilde{e} = (\tilde{P}, \tilde{Q})$, to the set $E_{e,t}$. Then, edges $\tilde{e}$ in $E_{e,t}$ are visited incrementally and assessed for the contribution of $\tilde{e}$ towards the N and L values of events along Edge e. BBT batch updates N and L for edges in Category 1 and applies a binary-search based method to update N and L for edges in Category 3. The time complexity of BBT is $O(|V_e||E_e| \log \frac{|V_e|}{|E_e|} + |E_e|^2 log|V_l| + |V_e| log|V_e|)$. The space complexity is $O(|V_e| + |V_l| + |E_e|)$. The proof is given in Appendix A.1.

Running Example: In Figure 4, when processing the events along Edge $e = (J, S)$, we have $D_{J,S} = \{F, J, S, Z, R, H, G\}$ when $t = 15$, and $E_{e,t}$ includes all edges except for (B, C) and (A, D). We first process e itself. Since $t = 15 > w(e) = 10$, we batch update $N = 3$ and $L = 7$ for all events. Then, other edges satisfying Category 1, 2, and 3 are colored blue, red, and green, respectively.

After processing the edge e itself and traversing all edges of Category 1: (F, J), (F, Z), (S, R), and (R, G), the N and L of events are batch updated to 3 and 22, respectively. Edges satisfying Category 2 are red-colored and pruned as they do not contribute to N and L of any event along e. Then we evaluate the edges in Category 3. Take (H, G) as an example, the minimum distance between e and (H, G) is 8, but the maximum distance is 25. Say $S_e = [d, b, a]$. We have $r_1(H) = max(t - dist(d, H), 0) = max(15 - 12, 0) = 3$, and $r_1(G) = max(t - dist(d, G), 0) = max(15 - 14, 0) = 1$. Since $r_1(H) + r_1(G) = 4 < 10$, this falls into Case 2. We invoke a binary search with a key equals to 3 from H to G to find that the number of reachable events is 0. Similarly, we invoke a binary search with a key equal to 1 from G to H and find that the number of reachable events is 1. Thus we increment $L(t|d)$ by 4 and $N(t|d)$ by 1. After batch evaluation of N and L, we have $N(15|a) = 9$, $N(15|b) = 7$, and $N(15|d) = 5$. Further, $L(15|a) = 42$, $L(15|b) = 43$, and $L(15|d) = 45$. Their respective K function values are $K(15|a) = \frac{N(15|a)}{\rho} = \frac{9}{0.103} = 87$, $K(15|b) = \frac{N(15|b)}{\rho} = \frac{7}{0.103} = 67.66$, and $K(15|d) = \frac{N(15|d)}{\rho} = \frac{5}{0.103} = 48.33$, where $\rho = \frac{|V_e|-1}{L(G)} = \frac{13-1}{116} = 0.103$. Finally, the statistical significance for a, b, d are 99.74%, 93.64%, 58.09%. Hence, a and b are hotspots with $t = 15$ and $\alpha = 90\%$.

4.3 Incremental Batched Traversal (IBT)

BBT requires a binary search for each event along Edge e to find the number of events reachable from both ends of Edge $\tilde{e}$. This operation is repeated for every event which can be expensive. We introduce Incremental Batched Traversal (IBT) to solve this problem. IBT employs incremental processing to compute N and L values for an event vertex based on the N and L values of its neighbor.

Similar to BBT, IBT also processes events along Edge e, represented as $S_e[i]$, in sequential order, for i ranging from 1 to $|S_e|$. For each event along e, i.e., $S_e[i]$, IBT calculates values $r_i(\tilde{P})$ and $r_i(\tilde{Q})$. The handling of Case 1 in Category 3 is identical to that of BBT. However, IBT handles Case 2 incrementally.

For the first event along Edge e, represented as $S_e[i]$ with $i = 1$, IBT initiates a binary search with key equals to $r_i(\tilde{P})$ along $S_{\tilde{e}}$. Then IBT maintains a pointer, denoted as $ptr_{\tilde{P}}$. This pointer points to the furthest vertex that $S_e[i]$ can reach from $\tilde{P}$ to $\tilde{Q}$. Likewise, another binary search with a key equal to $r_i(\tilde{Q})$ is invoked and

Algorithm 1: Incremental Batched Traversal (IBT)

Input: G : *The spatial network*
t : *The network traverse distance*
Output: N *and* L *for each event vertex*

```
Ns, Ls = { }, { }
for each Edge e = (P, Q) in G do
    D_{P,Q} = location vertices reachable from P or Q within t
    E_{e,t} = Set()
    for P̃ in D_{P,Q} do
        for Q̃ in P̃'s neighbors do
            ẽ = (P̃, Q̃)
            add ẽ to E_{e,t}
    for ẽ in E_{e,t} do
        //edges in E_e − E_{e,t} are in Category 2 and are pruned
        if e == ẽ then
            apply the Sliding Window approach
        else
            compute D_min(e, ẽ) and D_max(e, ẽ)
            if D_max(e, ẽ) ≥ t then
                update N,L based on Category 1
            else
                incrementally update N,L based on Category 3
    update Ns and Ls with the events along e
return Ns, Ls
```

IBT maintains another pointer, denoted as $ptr_{\tilde{Q}}$, that points to the furthest vertex reachable by $S_e[i]$ from $\tilde{Q}$ to $\tilde{P}$ along $\tilde{e}$.

Beginning from $i = 2$, when computing the number of events on $\tilde{e}$ that can be reached from $S_e[i]$ (either from $\tilde{Q}$ to $\tilde{P}$ or from $\tilde{P}$ to $\tilde{Q}$), IBT starts from the previous pointers, $ptr_{\tilde{Q}}$ and $ptr_{\tilde{P}}$. It then compares $r_i(\tilde{P})$ against $r_{i-1}(\tilde{P})$ and $r_i(\tilde{Q})$ against $r_{i-1}(\tilde{Q})$, and updates the positions of the two pointers, effectively tracking the current furthest event that can be reached from both ends and updating $N(t|S_e[i])$ and $L(t|S_e[i])$ accordingly. IBT continues until all the events along e have been processed. The time complexity of IBT is $O(|V_e||E_e| + |E_e|^2 log|V_l| + |V_e|log|V_e|)$. The space complexity is $O(|V_e| + |V_l| + |E_e|)$. The proof is presented in Appendix A.2.

Running Example: In Figure 4, when processing events along Edge $e = (J, S)$ with $t = 15$ regarding Edge (H, G), in the first iteration, $S_e[1] = d$, and we have $r_1(H) = 3$ and $r_1(G) = 1$, because $t - dist(d, H) = 3$ and $t - dist(d, G) = 1$. We have two pointers, ptr_H pointing to H, and ptr_G pointing to q. Then, in the second iteration, $S_e[2] = b$, and we have $r_2(H) = 1$ and $r_2(G) = 3$, because $t - dist(b, H) = 1$ and $t - dist(b, G) = 3$. Since $r_2(H) - r_1(H) = -2$, ptr_H still points to H. On the other hand, $r_2(G) - r_1(G) = 2$ and ptr_G now points to f. Algorithm 1 gives the pseudocode for IBT.

5 Hotspot Detection without Predefined Radius (HDWPR)

In this section, we present the solution for HDWPR problem, i.e., hotspot detection without a predefined radius. For each hotspot vertex, there exists an optimal radius that most accurately defines the precise locality of the hotspot. In the following, we discuss how to determine the optimal radius for a hotspot.

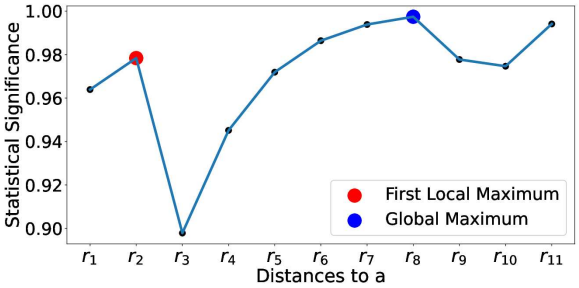
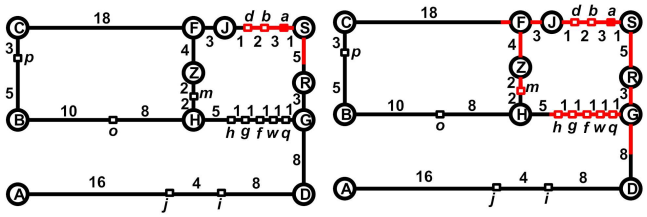


Figure 5: Statistical Significance under Different Radii



(a) Local Maximum **(b) Global Maximum**

Figure 6: Hotspot Detection Example

5.1 Optimal Hotspot Radius Determination

Consider a hotspot Vertex ϑ, there can be $O(|V_e|)$ distinct shortest distances between ϑ to other event vertices. The possible radii for ϑ come from distinct shortest distances between ϑ to all other event vertices [33]. For example, in Figure 1a, there are 11 distinct distances from Event a to other event vertices. Suppose that the distances are sorted as $(r_1, ...r_{11})$ where $r_i < r_{i+1}$. For $i \in [1, 11]$, we have $r_1 = dist(a, b) = 3$ and $r_{11} = dist(a, p) = 30$. Each r_i corresponds to a statistical significance level α_i. Figure 5 shows the statistical significance for each radius. From the figure, we see that when $t = 5, i.e., r_2$, we achieve the first local maximum significance of 97.83%. In this case, the isodistance subnetwork of a includes a, b, and d, as shown in Figure 6a. When $t = 15, i.e., r_8$, we achieve the global maximum statistical significance of 97.83%. In this case, the isodistance subnetwork of a includes $a, b, d, m, h, g, f, w, q$, as shown in Figure 6b. The events in the isodistancce subnetwork shown in Figure 6a are more localized compared to the events in the isodistance subnetwork shown in Figure 6b.

For a given event vertex, we aim to identify the radius t that yields the ***first local maximum statistical significance***. This ensures that the identified hotspot is as localized as possible. The radius that brings the first local maximum statistical significance reflects a region where event occurrences are truly dense and not affected by other nearby clusters. This approach ensures spatial locality in hotspot detection, maximizing the relevance and accuracy of detected hotspots. This conforms to Tobler's first law of geography [35], "Everything is related to everything else, but near things are more related than distant things".

In the search for the optimal hotspot radius, we start with an initial radius value. Then, we increment the radius until we hit the optimal radius value. Two questions arise here, (1) what is the

starting radius value, and (2) how to increment the radius. The starting radius of a hotspot needs to be neither too small nor large. If the starting radius is set too small, it could potentially lead to false positive detection of hotspots. Conversely, if we set the starting radius too large, we might overshoot and the large initial radius could encompass multiple event clusters that are not in the same spatial vicinity. Setting the starting radius to the distance between the center and the event closest to it can be problematic. The reason is that in most practical applications many events can be mapped to the same location. This is due to the low accuracy of event mapping or obfuscation of locations due to privacy constraints. In this case, the starting radius will be set to 0. This will result in a statistical significance of 1. This high statistical significance can lead to false detection of hotspots.

We set the starting radius to the weight of the edge on which the event is located. For instance, in Figure 6, we assign a starting radius of 7 to event a, which corresponds to $w(J, S)$. The rationale behind this choice stems from the observation that events situated on the same edge are inherently closer and more likely to be interconnected, thus making it a suitable initial range for hotspot analysis. After obtaining the initial radius, we proceed with incrementing the radius. In each iteration, we increase the radius to the minimum value required to include the next nearest event vertex from the source event vertex. This iterative process continues until we identify a radius that results in the first local maximum statistical significance. As an illustration, in Figure 6, during the second iteration, we set the radius for Event a to be 10, which corresponds to $dist(a, q)$.

An exact approach to determine the optimal hotspot radius is by utilizing the Graph Traversal Algorithm (GT), which is discussed in Section 4.1. GT traverses the graph incrementally, visiting vertices based on their distances from the source vertex. This incremental graph traversal of GT guarantees exact and optimal results. However, GT is computationally expensive as it requires performing $O(|V_e|)$ graph traversals for each event due to the existence of up to $O(|V_e|)$ distinct distances. For instance, in Figure 6, the number of traversals for each event vertex can reach up to 11. Consequently, for each event vertex, this process takes $O(|V_e|(|E_e| + |E_s|)\log(|V_e| + |V_l|))$ time, since executing GT with a specific distance t requires $O(|E_e| + |E_s|)\log(|V_e| + |V_l|))$ time and there can be $O(|V_e|)$ different distances before reaching the first local maximum.

Running Example: In Figure 6, the initial radius for a is set to 7 because $w(J, S) = 7$. We have $N(7|a) = 3, L(7|a) = 14$, and $K(7|a) = 29$ which corresponds to a statistical significance of 91.54%. The next closest event to a is q with $dist(a, q) = 10$. Consequently, in the next iteration, the radius of a is set to 10, and $N(10|a) = 4, L(10|a) = 22$, and $K(10|a) = 38.67$, which yields a statistical significance of 89.79%. Since 89.79% < 91.54%, we conclude that the optimal radius for event Vertex a is 7.

5.2 Approximate Hotspot Identification

To achieve a trade-off between scalability and solution quality, we propose an approximate algorithm named Approximate Hotspot Identification via Incremental Batched Traversal (AH-IBT). This algorithm is an adaptation of our most efficient algorithm for hotspot detection with a predefined radius, i.e., IBT.

Similar to IBT, AH-IBT computes the optimal t for the events along the same edge in a single batch. The process involves incrementally increasing the radius values but in a coarser granularity. We gradually increase the radius to cover the nearby location vertices rather than event vertices so that the nearby edges are incrementally traversed.

While determining the optimal radii for the events along $e = (P, Q)$, we store the distinct distances between P and other location vertices in the array D_P. The array is sorted such that $D_P[i] < D_P[i + 1]$ for all $1 \leq i < |D_P|$, with $D_P[1] = 0$. During the i^{th} iteration, we set the radius, t_i, for events along Edge e as $t_i = D_P[i] + w(e)$. This approach employs a coarser granularity in incrementing the radius compared to the ground truth computation, leading to enhanced computational efficiency. However, our experimental results given in Section 6.3 and Appendix C.2, show that this approximation maintains a high level of accuracy. It exhibits less than a 5% error in determining the optimal radius for hotspots, effectively striking a balance between efficiency and precision.

Algorithm 2 gives the outline of AH-IBT, during the first iteration, i.e., $i = 1$, of identifying optimal radii for events along e, AH-IBT applies IBT to compute the corresponding statistical significance with radius t_i for each event, C_e, where $C_e[j]$ represents the statistical significance for the j^{th} event along e. Meanwhile, we initialize a mask array M of the same size as C_e. If $C_e[j]$ surpasses the minimum statistical significance threshold in this first iteration, then $M[j]$ is assigned a true value, implying that the optimal t value for event $S_e[j]$ could be found at a larger t value. Conversely, if this threshold is not met, $M[j]$ is set to false, signifying that further processing of this event is unnecessary as it does not form a hotspot from its initial radius. The statistical significance from the initial radius should be larger than a predefined threshold, suggesting spatial locality of events is observed.

Starting from the second iteration, while recalculating the N and L values for S_e with the updated radius value, we restrict our processing to events whose mask values are set to true. Also, since the radius monotonically increases, we incrementally compute the new N and L values by evaluating only the edges that become reachable under the larger radius at each step to avoid redundant computation. Once the new statistical significance is obtained, we compare these with the previous statistical significance for events marked true in the mask array. If the new statistical significance exceeds the previous one, the mask remains true, and we update the optimal t value and the statistical significance for this event. Otherwise, the mask is set to false, and we determine the optimal t value for this event to be the last t value, corresponding to the local maximum statistical significance. This iterative process continues until all entries in the mask array are false. The time complexity of AH-IBT is $O(|V_e||E_e| + |E_e|^2 log|V_l| + |V_e|log|V_e|)$. The space complexity is $O(|V_e| + |V_l| + |E_e|)$. The proof is given in Appendix A.3.

Running Example: In Figure 6, when calculating the optimal radii for the events along Edge $e = (J, S)$, the initial radius in the first iteration is set to 7. The corresponding statistical significances for vertices d, b, and a are 84.45%, 89.36%, and 91.54%, respectively. In the second iteration, the radius is set to 10 because the closest location vertex to $e[0]$ is F, and we have $t = w(J, S) + w(F, J) = 10$. With the new radius, the statistical significances for vertices d, b,

Algorithm 2: Approximate Hotspot Identification

Input: *G : The spatial network*
MSS : The minimum statistical significance
Output: *optimal radius for each event vertex*

```
optimal_r = { }
for each Edge e = (P, Q) in G do
    mask_e = [True, True, ....] //mask array of events
    SS_e = [0, 0, ...] //statistical significance of events
    r_e = [0, 0, ...] //optimal radii of events
    t_e = w(P, Q) // starting radius
    D_P = sorted distances between P to other location vertices
    while mask.any() is True do
        //process only events whose mask values are True
        SS_new = IBT(e, t, mask)
        for i = 1 to |e| do
            //process each event along e
            if First Iteration then
                if SS_new[i] > MSS then
                    SS_e[i] = SS_new[i]
                else
                    mask_e[i] = False
            else
                if mask_e[i] then
                    if SS_new[i] > SS_e[i] then
                        SS_e[i] = SS_new[i]
                    else
                        r_e[i] = t
                        mask_e[i] = False
        incre = next smallest distance value in D_p
        t_e = w(P, Q) + incre
    update optimal_r
return optimal_r
```

and a become 81.78%, 64.37%, and 89.79%, respectively. Therefore, the optimal radius for events d, b, and a is determined to be 7.

6 Experimental Evaluation

In this section, we present an extensive experimental evaluation of our proposed algorithms against the state-of-the-art algorithms for hotspot detection over spatial networks.

6.1 Experimental Setup

We evaluate our algorithms for hotspot detection with a predefined radius (HDPR), i.e., BBT and IBT against NS*. NS* is a modified version of state-of-the-art neighborhood sharing (NS) [5] algorithm that computes the network global K-Function. While NS computes N values for each event vertex and sums it up, NS* also tracks L values per event vertex with shared processing among neighbor vertices. This allows it to compute the network local K-function. For detecting hotspots without predefined radius (HDWPR), we evaluate our proposed algorithm, i.e., AH-IBT against two state-of-the-art algorithms, namely, AH-NS* and NPP*. AH-NS* combines NS* with our proposed approximate hotspot identification algorithm, i.e., AH from AH-IBT. NPP* is a modified version of NPP [33]. NPP is the state-of-the-art algorithm for object-centered hotspot detection in spatial networks. NPP returns hotspots with scores assigned to all possible radii. NPP* modifies NPP to return the radius with the maximum score per detected hotspot.

We measure the runtime for algorithms that detect hotspots with a predefined radius (HDPR). For algorithms that detect hotspots without a predefined radius (HDWPR), we measure both the runtime and average error, i.e., t_{err}. The average error represents the mean difference between the radius computed by the hotspot detection algorithm and the optimal hotspot radius. $t_{err} = \frac{1}{n}\sum_{i=1}^{n}\frac{|t_c^i - t_g^i|}{max(t_c^i, t_g^i)}$, where t_c^i and t_g^i refer to the radius value computed by the algorithm and the optimal hotspot radius for the i^{th} hotspot, respectively. The optimal hotspot radius is defined in Section 5.1. We use both real and synthetic datasets in the evaluation of the proposed algorithms. The real datasets are the car collisions in the Seattle network [11], car collisions in the New York City network [19], 911 calls of the Detroit network [20], and crime locations of the Chicago network [24]. *Synth-Seattle-Random* is a synthetic dataset that is based on the road network of Seattle with randomly generated events. *Synth-Detroit-Hotspots* is a set of synthetic datasets with hotspots based on the Detroit road network topology with different numbers of location vertices and events. The parameters of the datasets are given in Table 1.

Table 1: Datasets

Dataset	#Locations	#Events
Seattle	11K	0.2M
Detroit	16K	5.1M
Chicago	23K	7.6M
Synth-Seattle-Random	11K	1M
Synth-Detroit-Hotspots	1 - 5K	0.5K - 500K

Table 2: Parameters

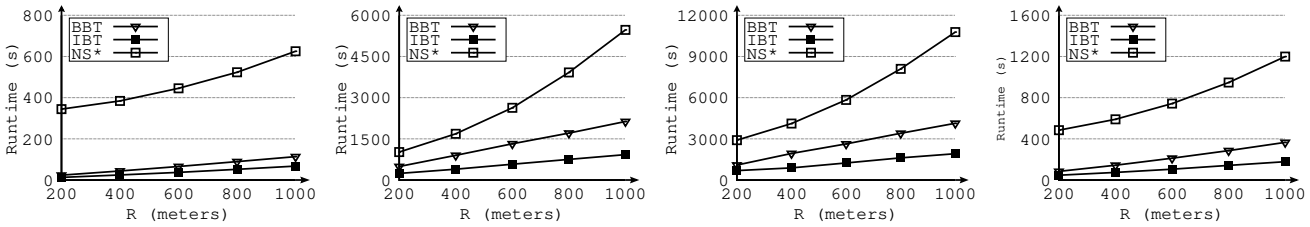
Parameter	Values
R - Radius (in meters)	200, 400, **600**, 800, 1000
SS - Statistical Significance	90%, 92%, **94%**, 96%,98%
MSS - Min Statistical Significance	90%, 92%, **94%**, 96%,98%

All algorithms are implemented using Python 3.10 and all experiments are conducted over an Intel Xeon(R) server with CPU E5-2637 v4 (3.50 GHz) and 128GB RAM running Ubuntu 16.04. Table 2 summarizes the parameters used in all experiments. Bold values represent the default settings for each parameter.

6.2 HDPR Evaluation

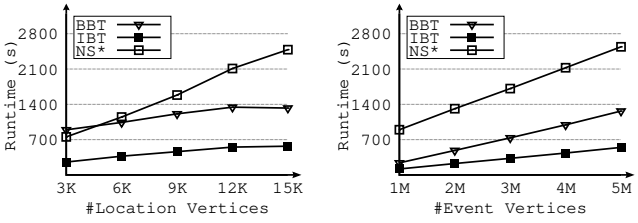
This section evaluates the performance of BBT and IBT against the state-of-the-art algorithm NS* in computing hotspots with predefined radius (HDPR).

6.2.1 ***The effect of radius (t)*** Figure 7 shows the runtime of hotspot detection algorithms, i.e., BBT, IBT, and NS* while changing the predefined hotspot radius under both real and synthetic datasets. In this figure, we see that IBT consistently outperforms other algorithms. IBT is up to 28.67 times faster than NS*. The efficiency of IBT is due to the following reasons: (1) efficient pruning of vertices and edges that do not contribute to hotspots and (2) batch



(a) Seattle (b) Detroit (c) Chicago (d) Synth-Seattle-Random

Figure 7: Effect of Varying Radius in HDPR



(a) Varying location vertices (b) Varying event vertices

Figure 8: Effect of Varying Number of Vertices in HDPR

processing of events. This batch and shared processing allow updating N and L values of multiple event vertices along the same edge simultaneously. Although NS* uses neighborhood sharing, NS* has a quadratic time step used to evaluate all pairs of edges in the spatial network and it processes each event vertex individually, which is considerably expensive and leads to a longer runtime. Also, Figure 7 shows that as the radius increases, the runtime for all algorithms increases. The reason is that with larger radii, hotspot detection algorithms need to explore more edges and vertices. This results in an increase in runtime. Notice that, IBT remains stable while increasing the radius. This is due to the efficiency of batch processing and neighborhood sharing used in IBT.

6.2.2 ***The effect of the number of locations*** In this experiment, we investigate the effect of increasing the number of location vertices on the performance of predefined-radius hotspot detection algorithms. We generate multiple granularities of the Detroit road network using the Python OSMnx library [23] by tuning the library's parameters. We change the number of location vertices from 3K to 15K. Figure 8a shows the effect of increasing the number of location vertices. IBT consistently achieves performance that is up to 4.36 times faster than NS*. Notice that, the runtime of all algorithms increases as the number of location vertices increases. The main reason is that, as the number of location vertices increases, the number of edges connecting location vertices increases. This increases the edge traversal overhead required by all algorithms.

6.2.3 ***The effect of the number of events*** To study the effect of the number of events on performance, we sample events from the Detroit road network to obtain event sets with sizes ranging from 1M to 5M. Figure 8b shows that increasing the number of event vertices results in increasing the runtime for all algorithms. The reason is that increasing the number of event vertices results in more processing per hotspot for all algorithms. Notice that, IBT is the most efficient algorithm and consistently achieves up to 7.58 times faster compared to NS*. An additional experiment investigating how the statistical significance affects the runtime and accuracy is given in Appendix C.1. Statistical significance has little impact on the runtime because it only affects the final validation of hotspots.

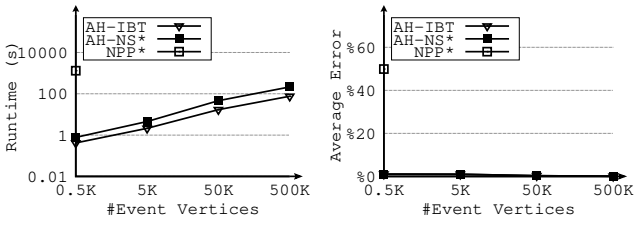
6.3 HDWPR Evaluation

In this section, we evaluate the performance of AH-IBT against AH-NS* and NPP* for spatial hotspot detection without a predefined radius (HDWPR).

6.3.1 ***The effect of the number of events*** Figure 9a shows the runtime of AH-IBT, AH-NS*, and NPP* evaluated with the number of events ranging from 0.5K to 500K. It is worth noting that the results for NPP* with larger datasets (events exceeding 5K) are not shown due to its extremely long experimentation time. The figure reveals that AH-IBT is more than three orders of magnitude faster than NPP* and up to 2.81 times faster than AH-NS*. The efficiency of AH-IBT stems from two main factors: (1) the inherent efficiency of IBT due to its batch processing, and (2) AH-IBT's focus on investigating hotspot radii that are likely to contribute to the final answer, unlike NPP* which performs a nearly exhaustive search on hotspot radii. Figure 9b presents the relative error, as defined in Section 6.1, for hotspot detection algorithms while varying the number of events. This figure clearly demonstrates that AH-IBT achieves up to a 50% higher accuracy compared to NPP*. The reason behind this improvement is that AH-IBT identifies the radius that yields the first local maximum statistical significance, whereas NPP* focuses more on the maximum log-likelihood ratio. Although the log-likelihood score, combined with Monte-Carlo trials, provides evidence of hotspot existence, it overlooks the spatial locality principle, which states that hotspots should be centralized and unaffected by nearby clusters.

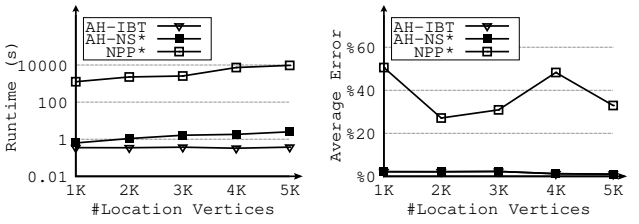
6.3.2 ***The effect of the number of locations*** Figure 10a illustrates the runtime of AH-IBT, AH-NS*, and NPP* using the synthetic dataset that is based on the Detroit road network with varying numbers of location nodes. Similar to the reasons discussed earlier, AH-IBT consistently outperforms the other algorithms, being up to more than four orders of magnitude faster than NPP* and 8.06 times faster than AH-NS*. As the number of location vertices increases, the runtime of all algorithms also increases due to the need to visit more edges between the expanded nodes. However, it is important to note that the performance of AH-IBT remains stable. Figure 10b shows the error of AH-IBT, AH-NS*, and NPP*. For similar reasons, both AH-IBT and AH-NS* achieve higher accuracy compared to NPP*, with improvements of over 50%.

Additional experiments investigating how the minimum statistical significance, the number of hotspots, and the radii of hotspots in the dataset affect the runtime and accuracy are presented in



(a) Runtime **(b) Accuracy**

Figure 9: Effect of Varying Number of Events in HDWPR



(a) Runtime **(b) Accuracy**

Figure 10: Effect of Varying Number of Locations in HDWPR

Appendix C.2. In these experiments, AH-IBT consistently achieves the best runtime and accuracy.

7 Conclusion

This paper studies scalable hotspot detection in spatial networks, ensuring statistical significance. We advocate the usage of the network local K-function for discerning hotspots and present IBT, a highly efficient batch processing algorithm, specifically designed to compute hotspots over spatial networks with predefined radii. An exhaustive experimental evaluation employing both real and synthetic datasets indicates that IBT exhibits a performance that is up to 28 times faster than its contemporary counterparts. Additionally, we unveil AH-IBT, an adaptive algorithm derived from IBT, purposed for the detection of hotspots along with their radii. AH-IBT boasts a speed that surpasses the state-of-the-art algorithm by up to four orders of magnitude times and delivers an accuracy that is over 50% superior. Our future work will extend our proposed algorithms to accommodate moving objects within spatial networks.

References

[1] M. A. Andresen. *Environmental Criminology: Evolution, Theory, and Practice*. Routledge, 2019.

[2] S. Berglund and A. Karlström. Identifying Local Spatial Association in Flow Data. *Journal of Geographical Systems*, 1:219–236, 1999.

[3] T. N. Chan, P. L. Ip, L. H. U, B. Choi, and J. Xu. Sws: A complexity-optimized solution for spatial-temporal kernel density visualization. *Proceedings of the VLDB Endowment*, 15(4):814–827, 2021.

[4] T. N. Chan, Z. Li, L. H. U, J. Xu, and R. Cheng. Fast Augmentation Algorithms for Network Kernel Density Visualization. *Proceedings of the VLDB Endowment*, 14(9):1503–1516, 2021.

[5] T. N. Chan, L. H. U, Y. Peng, B. Choi, and J. Xu. Fast Network k-function-based Spatial Analysis. *Proceedings of the VLDB Endowment*, 15(11):2853–2866, 2022.

[6] M. Deng, X. Yang, Y. Shi, J. Gong, Y. Liu, and H. Liu. A Density-based Approach for Detecting Network-constrained Clusters in Spatial Point Events. *International Journal of Geographical Information Science*, 33(3):466–488, 2019.

[7] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[9] A. Getis and J. K. Ord. The Analysis of Spatial Association by Use of Distance Statistics. *Geographical Analysis*, 24(3):189–206, 1992.

[10] M. Joshi, A. Singh, S. Ranu, A. Bagchi, P. Karia, and P. Kala. FoodMatch: Batching and Matching for Food Delivery in Dynamic Road Networks. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 8(1):1–25, 2022.

[11] Kaggle. Seattle SDOT Collisions Data. https://www.kaggle.com/datasets/jonleon/seattle-sdot-collisions-data, 2020.

[12] M. Kampa and E. Castanas. Human Health Effects of Air Pollution. *Environmental pollution*, 151(2):362–367, 2008.

[13] S. Khalid, F. Shoaib, T. Qian, Y. Rui, A. I. Bari, M. Sajjad, M. Shakeel, and J. Wang. Network Constrained Spatio-temporal Hotspot Mapping of Crimes in Faisalabad. *Applied Spatial Analysis and Policy*, 11:599–622, 2018.

[14] A. B. Lawson. Hotspot Detection and Clustering: Ways and Means. *Environmental and Ecological Statistics*, 17(2):231, 2010.

[15] Y. Li, Y. Xie, P. Wang, S. Shekhar, and W. Northrop. Significant Lagrangian Linear Hotspot Discovery. In *Proceedings of the 13th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 1–10, 2020.

[16] X. Lin and J. Xu. Road Network Partitioning Method Based on Canopy-kmeans Clustering Algorithm. *Archives of Transport*, 54(2):95–106, 2020.

[17] S. Mclafferty. Disease Cluster Detection Methods: Recent Developments and Public Health Implications. *Annals of GIS*, 21(2):127–133, 2015.

[18] K. Nie, Z. Wang, Q. Du, F. Ren, and Q. Tian. A Network-constrained Integrated Method for Detecting Spatial Cluster and Risk Location of Traffic Crash: A Case Study from Wuhan, China. *Sustainability*, 7(3):2662–2677, 2015.

[19] NYCOpenData. New York City Motor Vehicle Collisions - Crashes. https://opendata.cityofnewyork.us, 2023.

[20] C. of Detroit Open Data Portal. Detroit 911 Calls For Service. https://data.detroitmi.gov, 2023.

[21] A. Okabe, K.-i. Okunuki, and S. Shiode. SANET: a Toolbox for Spatial Analysis on a Network. *Geographical Analysis*, 38(1):57–66, 2006.

[22] A. Okabe and K. Sugihara. *Spatial Analysis Along Networks: Statistical and Computational Methods*. John Wiley & Sons, 2012.

[23] Python OSMnx Library User reference. https://osmnx.readthedocs.io/en/stable/osmnx.html, 2020.

[24] C. D. Portal. Chicago Crime Data. https://data.cityofchicago.org/, 2023.

[25] Y. Ren, M. Ercsey-Ravasz, P. Wang, M. C. González, and Z. Toroczkai. Predicting Commuter Flows in Spatial Networks Using a Radiation Model Based on Temporal Ranges. *Nature Communications*, 5(1):5347, 2014.

[26] B. D. Ripley. The Second-order Analysis of Stationary Point Processes. *Journal of Applied Probability*, 13(2):255–266, 1976.

[27] B. Romano and Z. Jiang. Visualizing Traffic Accident Hotspots based on Spatial-temporal Network Kernel Density Estimation. In *Proceedings of the International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 1–4, 2017.

[28] Y. Shi, Y. Chen, M. Deng, L. Xu, and J. Xia. Discovering Source Areas of Disease Outbreaks based on Ring-shaped Hotspot Detection in Road Network Space. *International Journal of Geographical Information Science*, 36(7):1343–1363, 2022.

[29] S. Shiode and N. Shiode. Detection of Multi-scale Clusters in Network Space. *International Journal of Geographical Information Science*, 23(1):75–92, 2009.

[30] M. Souris. *Epidemiology and Geography: Principles, Methods and Tools of Spatial Analysis*. John Wiley & Sons, 2019.

[31] T. Steenberghen, K. Aerts, and I. Thomas. Spatial Clustering of Events on a Network. *Journal of Transport Geography*, 18(3):411–418, 2010.

[32] X. Tang, E. Eftelioglu, D. Oliver, and S. Shekhar. Significant Linear Hotspot Discovery. *IEEE Transactions on Big Data*, 3(2):140–153, 2017.

[33] X. Tang, E. Eftelioglu, and S. Shekhar. Detecting Isodistance Hotspots on Spatial Networks: A Summary of Results. In *International Symposium on Spatial and Temporal Databases*, pages 281–299, 2017.

[34] X. Tang, J. Gupta, and S. Shekhar. Linear Hotspot Discovery on All Simple Paths: A Summary of Results. In *Proceedings of the International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 476–479, 2019.

[35] W. R. Tobler. A Computer Movie Simulating Urban Growth in the Detroit Region. *Economic Geography*, 46(sup1):234–240, 1970.

[36] R. J. Trudeau. *Introduction to Graph Theory*. Courier Corporation, 2013.

[37] T. Wang, C. Ren, Y. Luo, and J. Tian. NS-DBSCAN: A Density-based Clustering Algorithm in Network Space. *ISPRS International Journal of Geo-Information*, 8(5):218, 2019.

[38] Y. Xie, S. Shekhar, and Y. Li. Statistically-robust Clustering Techniques for Mapping Spatial Hotspots: A Survey. *ACM Computing Surveys (CSUR)*, 55(2):1–38, 2022.

[39] Y. Xie, X. Zhou, and S. Shekhar. Discovering Interesting Subpaths with Statistical Significance from Spatiotemporal Datasets. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(1):1–24, 2020.

[40] I. Yamada and J.-C. Thill. Local Indicators of Network-constrained Clusters in Spatial Point Patterns. *Geographical Analysis*, 39(3):268–292, 2007.

[41] M. L. Yiu and N. Mamoulis. Clustering Objects on a Spatial Network. In *Proceedings of the International Conference on Management of Data (ACM SIGMOD)*, pages 443–454, 2004.

Appendix

A Complexity Analysis

This section presents the time and space complexity for all the proposed algorithms.

A.1 Batch Based Traversal (BBT)

The time complexity of the BBT algorithm is $O(|V_e||E_e|\log\frac{|V_e|}{|E_e|} + |E_e|^2 log|V_l|+|V_e|log|V_e|)$ and the space complexity is $O(|V_e|+|V_l|+|E_e|)$.

BBT has two steps. (1) *sorting the events along each edge*, which takes $O(|V_e|\log|V_e|)$. This is because it takes $O(|e|\log|e|)$ to sort the events along e. Consequently, the complexity of sorting events along all edges is

$$\sum_{e\in E_e} O(|e|\log|e|) \le O(\log|V_e|)\sum_{e\in E_e} O(|e|) = O(|V_e|log|V_e|)$$

(2) *batch processing of events for every edge.* For an Edge $e=(P,Q)$, we first compute the location vertices reachable from P or Q within a predefined distance, which takes $O(|E_e|\log|V_l|)$. Then, for events along e, denote the cost of evaluating the contribution of an edge $\tilde{e}$ to events along e as $C_{e,\tilde{e}}$. If $\tilde{e}$ falls into Category 1, then $C_{e,\tilde{e}} = O(1)$. Otherwise, $\tilde{e}$ falls into Category 3, and $C_{e,\tilde{e}} = O(|e|\log|\tilde{e}|)$. The reason is that BBT invokes binary search along $\tilde{e}$ twice for all events on edge e, which takes $O(|e|\log|\tilde{e}|)$. BBT visits $O(|E_e|)$ edges when evaluating events along e. The complexity of evaluating events along e becomes

$$\begin{aligned}\sum_{\tilde{e}\in E_e} O(|e|\log|\tilde{e}|) &= O(|e|)\sum_{\tilde{e}\in E_e} O(\log|\tilde{e}|)\\ &= O(|e|)O(log\prod_{\tilde{e}\in E_e}|\tilde{e}|)\\ &\le O(|e|)O(log(\frac{\sum_{\tilde{e}\in E_e}|\tilde{e}|}{|E_e|})^{|E_e|})\\ &= O(|e||E_e|log\frac{|V_e|}{|E_e|})\end{aligned}$$

Consequently, batch processing events along e takes a complexity of $O(|e||E_e|log\frac{|V_e|}{|E_e|}) + O(|E_e|log|V_l|)$. Since there are $|E_e|$ edges to batch evaluate, the total time complexity for Step 2 for batch processing all edges becomes

$$\begin{aligned}\sum_{e\in E_e}(O(|e||E_e|\log\frac{|V_e|}{|E_e|}) + O(|E_e|log|V_l|)) &= (\sum_{e\in E_e}|e|)O(|E_e|\log\frac{|V_e|}{|E_e|}) + |E_e|^2\log|V_l|\\ &= O(|V_e||E_e|\log\frac{|V_e|}{|E_e|} + |E_e|^2log|V_l|)\end{aligned}$$

The total complexity for Step 2 is $O(|V_e||E_e|\log\frac{|V_e|}{|E_e|} + |E_e|^2log|V_l|)$. Combining the time complexities of Step 1 and Step 2, we conclude that the time complexity of BBT is $O(|V_e||E_e|\log\frac{|V_e|}{|E_e|}+|E_e|^2log|V_l|+|V_e|log|V_e|)$.

The space complexity of BBT is $O(|V_e| + |V_l| + |E_e|)$. This is due to the storage required for each event vertex's distance to the endpoint of the edge it is located on, taking up $O(|V_e|)$ space, the storage required for shortest path distances between location vertices, taking $O(V_l)$ space since we process one edge at a time. In addition, the set $E_{e,t}$ includes up to $|E_e|$ edges, which takes $O(|E_e|)$ space. Consequently, the total space complexity of BBT is $O(|V_e| + |V_l| + |E_e|)$.

A.2 Incremental Batched Traversal (IBT)

The time complexity of the IBT algorithm is $O(|V_e||E_e| + |E_e|^2log|V_l| + |V_e|log|V_e|)$ and the space complexity is $O(|V_e| + |V_l| + |E_e|)$. Similar to BBT, IBT also has two steps. (1) *sorting the events along each edge*, which takes $O(|V_e|\log|V_e|)$ as proved above, and (2) *batch processing of events in every edge.* However, IBT differs from BBT in Step 2, i.e., addressing edges in Category 3. In IBT, $C_{e,\tilde{e}}$ is bounded by $O(|e|+|\tilde{e}|)$ instead of $O(|e|\log|\tilde{e}|)$. The reason is that IBT utilizes two pointers, $ptr_{\tilde{P}}$ and $ptr_{\tilde{Q}}$, and updates them in each iteration. Given the relationship between $sdist(P,\tilde{P})$, $sdist(Q,\tilde{P})$, and $w(e)$, three scenarios are considered for updating $ptr_{\tilde{P}}$ (updating $ptr_{\tilde{Q}}$ is similar):

Scenario 1: $sdist(P,\tilde{P}) + w(e) < sdist(Q,\tilde{P})$. Here, $r^i(\tilde{P})$ decreases monotonically, causing $ptr_{\tilde{P}}$ to move from $\tilde{Q}$ to $\tilde{P}$ as i increases, resulting $O(|e|+|\tilde{e}|)$ complexity.

Scenario 2: $sdist(Q,\tilde{P})+w(e) < sdist(P,\tilde{P})$. The complexity is similar to Scenario 1, i.e., $O(|e|+|\tilde{e}|)$.

Scenario 3: $|sdist(P,\tilde{P}) - sdist(Q,\tilde{P})| \le w(e)$. The update of pointer $ptr_{\tilde{P}}$ is monotonic for $i\in[1,j]$ and $i\in(j,|S_e|]$ but in opposite directions, where $j\in[1,|S_e|]$ such that for any events $S_e[i]$ with $i\le j$, $sdist(S_e[i],P)\le sdist(S_e[i],Q)$ and for any events $S_e[i]$ with $i>j$, $sdist(S_e[i],P) > sdist(S_e[i],Q)$. This also gives a complexity of $O(|e|+|\tilde{e}|)$.

Updating pointer $ptr_{\tilde{Q}}$ also takes $O(|e|+|\tilde{e}|)$, proven similarly. IBT visits $O(|E_e|)$ edges when evaluating events along e. The complexity of evaluating events along e becomes

$$\sum_{\tilde{e}\in E_e} O(|e|+|\tilde{e}|) = \sum_{\tilde{e}\in E_e} O(|e|) + \sum_{\tilde{e}\in E_e} O(|\tilde{e}|) = O(|e||E_e|+|V_e|)$$

Consequently, batch processing events along e takes a complexity of $O(|e||E_e|+|V_e|) + O(|E_e|log|V_l|)$. Since there are $|E_e|$ edges to batch evaluate, the total complexity for Step 2 for batch processing all edges becomes

$$\begin{aligned}\sum_{e\in E_e}(O(|e||E_e|+V_e) + O(|E_e|log|V_l|)) &= (\sum_{e\in E_e}O(|e|))O(|E_e|) + O(|V_e||E_e|) + O(|E_e|^2log|V_l|)\\ &= O(|V_e|)O(|E_e|) + O(|V_e||E_e|) + O(|E_e|^2log|V_l|)\\ &= O(|V_e||E_e|+|E_e|^2log|V_l|)\end{aligned}$$

The total complexity for Step 2 is $O(|V_e||E_e|+|E_e|^2log|V_l|)$. Combining the time complexities of Step 1 and Step 2, we conclude that the time complexity of IBT is $O(|V_e||E_e| + |E_e|^2log|V_l| + |V_e|log|V_e|)$. The space complexity of IBT is $O(|V_e| + |V_l| + |E_e|)$, proved similarly to BBT.

A.3 Approximate Hotspot Identification via Batched Edge Traversal (AH-IBT)

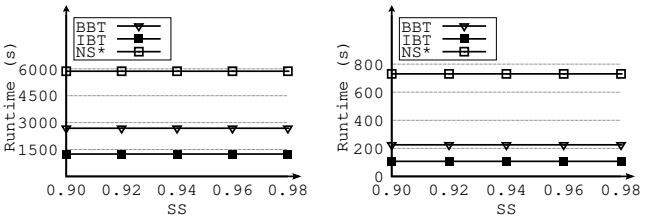
AH-IBT discovers hotspots without predefined radius by incrementally increasing the radius and computing the N and L using IBT for each radius. Similar to IBT, AH-IBT has two steps (1) sorting the events along each edge, and (2) batch processing of events and determine their optimal radii. Step 1 takes $O(|V_e|\log|V_e|)$, which is proved above. Step 2 takes $O(|e||E_e|+|V_e|)$ for the events along a single edge e. Since there can be $O(|E_e|)$ different radii, plus the cost of incrementally retrieving location vertices closest to the endpoint of $\tilde{e}$, which takes $O(|E_e|\log|V_l|)$, the complexity of processing a single edge e is $O(|e||E_e| + |V_e|) + O(|E_e|log|V_l|)$. Taking into account all $|E_e|$ edges, the total complexity for Step 2 is $O(|V_e||E_e|+|E_e|^2log|V_l|)$, proved similar to above.

Hence, the overall time complexity for AH-IBT is $O(|V_e||E_e| + |E_e|^2log|V_l| + |V_e|log|V_e|)$. The space complexity of AH-IBT is $O(|V_e| + |V_l| + |E_e|)$. The proof is similar to the above.

B Correctness Proof

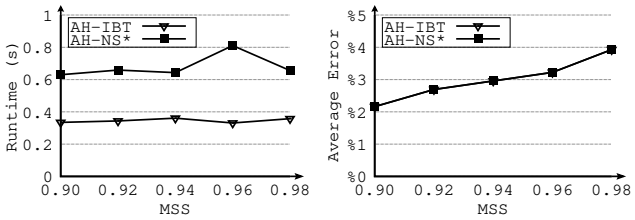
In this section, we give the correctness proof for all the proposed algorithms. We prove that the proposed algorithms, i.e., GT, BBT, and IBT compute HDPR correctly by proving that they compute the N and L values correctly. Then, we prove that AH-IBT correctly answers HDWPR.

GT: For an event vertex $\tilde{\vartheta}$, if $sdist(\vartheta,\tilde{\vartheta}) \le t$, $\tilde{\vartheta}$ is visited before GT's termination due to Dijkstra's algorithm properties. Any Edge, or Segment, $e=(u,v)$ contributing to ϑ's L value is considered when visiting u or v. This guarantees that all event and location vertices within t distance of ϑ are factored into N and L updates. GT terminates when all remaining vertices have distances to ϑ exceeding t.



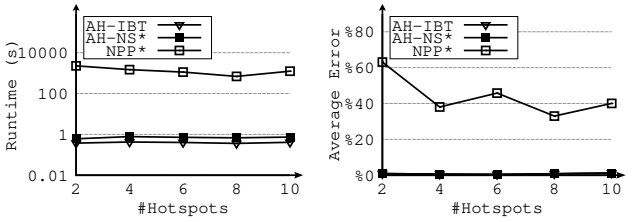
(a) Chicago **(b) Synth-Seattle-Random**

Figure 11: Effect of Varying Statistical Significance (SS) in HDPR



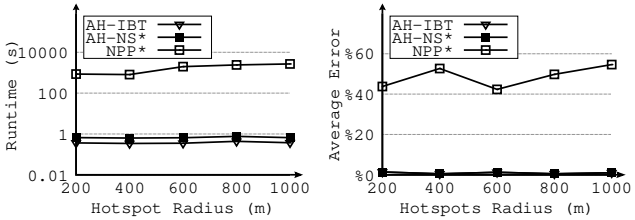
(a) Runtime **(b) Accuracy**

Figure 12: Effect of Varying Minimum Statistical Significance (MSS) in HDWPR



(a) Runtime **(b) Accuracy**

Figure 13: Effect of Varying Hotspot Numbers in HDWPR



(a) Runtime **(b) Accuracy**

Figure 14: Effect of Varying Hotspots Radii in HDWPR

BBT and IBT: For an event ϑ, BBT and IBT batch processes the computation of N and L with other events along the same edge e. Given another event, $\tilde{\vartheta}$ with $dist(\vartheta, \tilde{\vartheta}) \leq t$ attached to $\tilde{e} = (\tilde{P}, \tilde{Q})$, $\tilde{\vartheta}$ is considered for updating N and L since $\tilde{e}$ fulfills either Category 1 or 3 for e. BBT and IBT exclude any event $\overline{\vartheta}$ with $dist(\vartheta, \overline{\vartheta}) > t$ by pruning all edges that meet Category 2 for e.

AH-IBT: The final radius t output by AH-IBT is indeed the radius that brings the first local maximum statistical significance. Suppose this local maximum is identified at iteration i in Algorithm 2 when expanding the radius for the event ϑ, then for all iterations of expanding the radius before i, the statistical significance of ϑ monotonically increases according to our definition in Section 5.2 . Then, from the i^{th} to $(i+1)^{th}$ iteration of of expanding the radius , the statistical significance of ϑ drops. Thus, the radius identified at at iteration i is the first local maximum statistical significance for ϑ.

C Additional Experiments

C.1 Additional Experiments for HDPR

This section presents additional experiments for HDPR, i.e., hotspot detection with predefined radius.

C.1.1 ***The effect of statistical significance*** Figure 11 shows the runtime of BBT, IBT, and NS* while changing the statistical significance. IBT consistently outperforms the other algorithms with up to 6.82 times speed up for the same reasons explained in Section 6.2.1. Also, Figure 11 shows that the runtime remains stable under different statistical significance levels. The reason is that changing the statistical significance required for hotspot detection does not significantly impact the computational overhead of N and L in all algorithms. However, changing the required statistical significance only affects the final validation of hotspots, as discussed in Section 3. Although higher statistical significance results in fewer hotspots being qualified and detected, this has little impact on the overall runtime of algorithms.

C.2 Additional Experiments for HDWPR

This section presents additional experiments for HDWPR , i.e., hotspot detection without a predefined radius.

C.2.1 ***The effect of the minimum statistical significance*** In this experiment, we investigate how the minimum statistical significance affects the runtime and average error for AH-IBT and AH-NS*. Notice that we do not consider NPP* in this experiment as NPP* does not have statistical significance as an algorithm parameter. Figure 12a shows that AH-IBT is up to 2.44 times faster than AH-NS*. The reason is that AH-IBT is based on IBT and AH-NS* is based on NS* which is slower than IBT. Figure 12b shows that both AH-IBT and AH-NS* have the same average error because they adopt the same algorithm for identifying the radii of hotspots.

Although Figure 12b shows that the average error increases as *MSS* increases, there is no evident relationship between the average error and the minimum statistical significance. This is because under different *MSS*, the hotspots involved in computing the average error are different, which results in varying average error. However, in all cases, the average t error is less than 5%.

C.2.2 ***The effect of the number of hotspots and hotspots radii*** Figure 13a and Figure 14a show the runtime of AH-IBT, AH-NS*, and NPP* using Synth-Detroit-Hotspots dataset defined in Table 1. Both figures show that the number of hotspots and the radii of hotspots have a slight impact on the runtime for all algorithms. For similar reasons discussed in Section 6.3.1, AH-IBT consistently achieves better performance and it is up to four orders of magnitude faster than NPP* and up to 1.88 times faster than AH-NS*. Figure 13b and Figure 14b show the error of AH-IBT, AH-NS*, and NPP*. AH-IBT and AH-NS* achieve better accuracy compared to NPP*, with up to 50%, for the same reason discussed in Section 6.3.1.