Fast Augmentation Algorithms for Network Kernel Density Visualization*

Tsz Nam Chan
Hong Kong Baptist University
edisonchan@comp.hkbu.edu.hk

Zhe Li
Hong Kong Polytechnic University
richie.li@connect.polyu.hk

Leong Hou U
University of Macau
SKL of Internet of Things for Smart City
ryanlhu@um.edu.mo

Jianliang Xu
Hong Kong Baptist University
xujl@comp.hkbu.edu.hk

Reynold Cheng
The University of Hong Kong
Guangdong-Hong Kong-Macau Joint Laboratory for Smart Cities
ckcheng@cs.hku.hk

ABSTRACT

Network kernel density visualization, or NKDV, has been extensively used to visualize spatial data points in various domains, including traffic accident hotspot detection, crime hotspot detection, disease outbreak detection, and business and urban planning. Due to a wide range of applications for NKDV, some geographical software, e.g., ArcGIS, can also support this operation. However, computing NKDV is very time-consuming. Although NKDV has been used for more than a decade in different domains, existing algorithms are not scalable to million-sized datasets. To address this issue, we propose three efficient methods in this paper, namely aggregate distance augmentation (ADA), interval augmentation (IA), and hybrid augmentation (HA), which can significantly reduce the time complexity for computing NKDV. In our experiments, ADA, IA and HA can achieve *at least* 5x to 10x speedup, compared with the state-of-the-art solutions.

PVLDB Reference Format:
Tsz Nam Chan, Zhe Li, Leong Hou U, Jianliang Xu, and Reynold Cheng. Fast Augmentation Algorithms for Network Kernel Density Visualization. PVLDB, 14(9): 1503-1516, 2021.
doi:10.14778/3461535.3461540

1 INTRODUCTION

Data visualization is an important tool for understanding a dataset [17, 63, 67]. An important class of methods, collectively known as *kernel-density-estimation-based visualization* (or kernel density visualization (KDV)) [13], has been extensively used in a wide range of applications, including traffic accident hotspot detection [30, 38, 71, 77], crime hotspot detection [12, 31, 60], disease outbreak detection [2, 20, 80], and business and urban planning [45, 61, 78]. Geoscientists [30, 71] utilize KDV to visualize the spatial distribution of traffic accidents. Criminologists [12, 31] utilize KDV to identify crime hotspots (e.g., motor vehicle thefts in Figure 1). Public health experts [2, 80] utilize KDV to identify disease outbreak. Due to its wide range of applications, KDV is supported in many data analytics systems, including ArcGIS [1], QGIS [7], Scikit-learn [52], and KDV-Explorer [14].

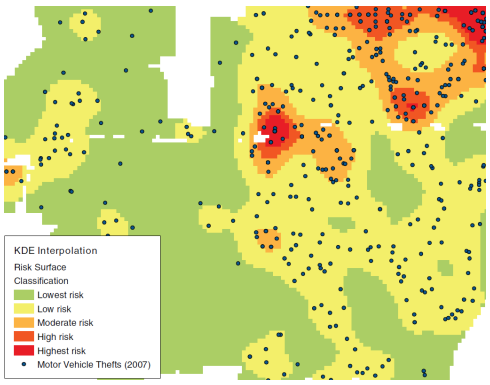


Figure 1: Planar KDV for visualizing the crime distribution (motor vehicle thefts) in Arlington, Texas (from [13, 31]).

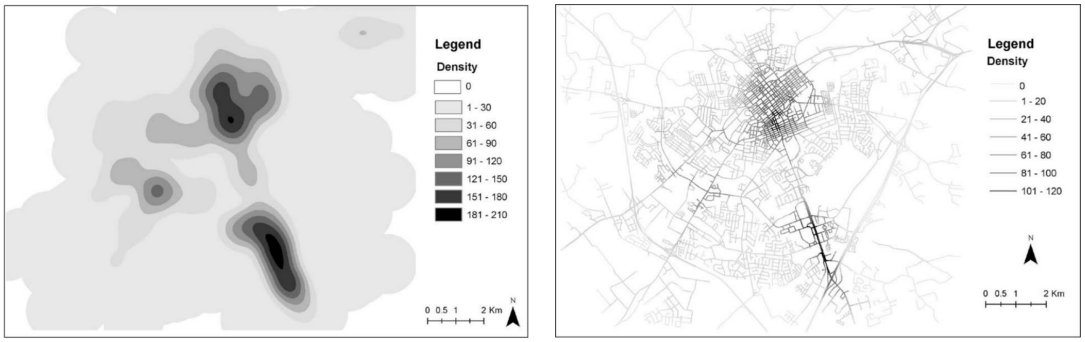
In the literature, most of the existing work mainly focuses on planar KDV [13, 16, 31, 32, 53, 82–84] (cf. Figure 1), in which they use $M \times N$ pixels (e.g., 2560×1920) to represent the visualized region. They color each pixel $\mathbf{q}$ based on the following kernel aggregation function $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1), where P denotes the set of two-dimensional data points $\mathbf{p_i}$ in the plane (e.g., black dots in Figure 1), w is the normalization constant and $K(\mathbf{q}, \mathbf{p_i})$ is the kernel function between two points.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w \cdot K(\mathbf{q}, \mathbf{p_i}) \quad (1)$$

Table 1 summarizes the commonly-used kernel functions $K(\mathbf{q}, \mathbf{p_i})$. Here, we denote $dist(\mathbf{q}, \mathbf{p_i})$ as the Euclidean distance and

*This work was supported by the National Key Research and Development Plan of China (No.2019YFB2102100), the Science and Technology Development Fund Macau (SKL-IOTSC-2021-2023, 0015/2019/AKP), University of Macau (MYRG2019-00119-FST), Guangdong Basic and Applied Basic Research Foundation (Project No. 2019B1515130001), the Research Grants Council of Hong Kong (RGC Projects HKBU 12201018, HKU 17229116 and 17205015), University of Hong Kong (Projects 104005858, 104005994), HKU-TCL Joint Research Center for Artificial Intelligence (Project no. 200009430), and Guangdong-Hong Kong-Macau Joint Laboratory Program 2020 (Project No: 2020B1212030009).

Proceedings of the VLDB Endowment, Vol. 14, No. 9 ISSN 2150-8097.
doi:10.14778/3461535.3461540



(a) Planar KDV (b) Network KDV

Figure 2: Differences between planar KDV and network KDV for visualizing the density of traffic accidents in Bowling Green, Kentucky (from [72]).

the parameter γ is used to control the bandwidth (i.e., $\frac{1}{\gamma}$) of the kernel functions [72]. If the distance value $dist(\mathbf{q}, \mathbf{p_i})$ is bigger than the bandwidth, the kernel function value is zero.

Table 1: Commonly-used kernel functions ($K(\mathbf{q}, \mathbf{p_i})$).

Kernel	$K(\mathbf{q}, \mathbf{p_i})$	Used in
Triangular	$\begin{cases} 1-\gamma dist(\mathbf{q}, \mathbf{p_i}) & \text{if } dist(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases}$	[10, 38]
Epanechnikov	$\begin{cases} 1-\gamma^2 dist(\mathbf{q}, \mathbf{p_i})^2 & \text{if } dist(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases}$	[10, 80]
Quartic	$\begin{cases} (1-\gamma^2 dist(\mathbf{q}, \mathbf{p_i})^2)^2 & \text{if } dist(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases}$	[38, 73]

However, there are two drawbacks for using the planar KDV to analyze the location distribution of traffic accidents or crime activities. Here, we use the traffic accidents in Bowling Green, Kentucky as an example (cf. Figure 2). In Figure 2a, we can only know the rough spatial regions that have a high density for traffic accidents. However, it is hard to identify which road segments are the hotspot regions. Moreover, two positions $\mathbf{q_1}$ and $\mathbf{q_2}$ that are close in terms of Euclidean distance can be far away in the road network (cf. Figure 3). Therefore, the distribution of points around these two positions might be very different (e.g., there are many traffic accidents near $\mathbf{q_1}$, but there are very few traffic accidents near $\mathbf{q_2}$). Since planar KDV does not consider the topology of road network for estimating the density, it is possible for planar KDV to estimate that these two positions have a similar density (as $\mathbf{q_1}$ and $\mathbf{q_2}$ are close) and assign the same color to them.

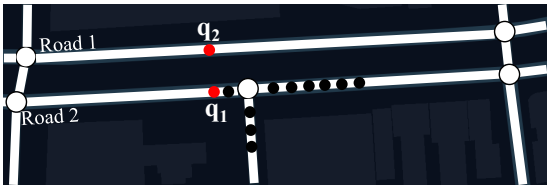


Figure 3: The positions q_1 and q_2 are close and far from each other in terms of Euclidean distance and the shortest path distance (network distance), respectively.

To achieve a more detailed and accurate visualization, existing studies [20, 30, 38, 47, 48, 60, 61, 72, 73] replace the Euclidean distance with the network distance in the kernel functions (cf. Table 1) and only visualize the densities of the positions in the road network (cf. Figure 2b). Here, we term this approach as *network kernel density visualization* (NKDV). Instead of outputting a large region with high density (i.e., black color) in Figure 2a, Figure 2b clearly shows which road segments contain more traffic accidents (i.e., hotspots). In addition, unlike Figure 2a, some road segments which are close to the hotspot region in terms of Euclidean distance may not necessarily have a high density. Therefore, NKDV can avoid overestimating the density for those road segments, which provides more accurate visualization.

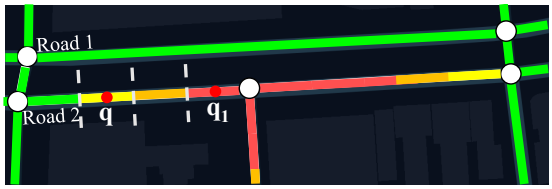


Figure 4: The 300m road (Road 2 in Figure 3) is segmented into four basic units (with 75m), called lixels. Each lixel is colored based on the kernel aggregation function, where the red color and green color denote the highest and lowest density values, respectively.

In order to obtain the NKDV in a region, we need to first segment each road of the road network into a set of basic units, called lixels (cf. Figure 4), the analogy of pixels [72]. Then, we need to color each lixel $\mathbf{q}$, based on evaluating the kernel aggregation function (cf. Equation 1 with network distance in the kernel function) [72]. In practice, Okabe et al. [47, 48] have developed the plug-in, called SANET [8], for ArcGIS to handle this NKDV with the simple algorithm, which will be discussed in Section 2. However, generating the NKDV is computationally expensive. Using the dataset of traffic accidents in New York city (with nearly 1.3M data points) [5] as an example, the state-of-the-art approaches [47, 57, 72, 73] take more than 3 hours to generate the visualization. As such, many recent studies also complain about the inefficiency issue for using NKDV.

- *"...the shortest-path distance can be used as an alternative to the Euclidean one, and since this distance strongly depends on the structure of the linear network, it adds a point of difficulty to the computation. Hence, having efficient approaches to analyze spatial point patterns on linear networks is a welcome issue."* [43]

- *"Previous implements on network kernel density estimation have so far been done, but due to the computational complexity of the shortest-path distance calculating, these methods all tend to be very time consuming, especially for large datasets."* [78]
- *"Kernel smoothing of point events, which is simple to define and very fast to compute in two dimensions (Diggle 1985), is mathematically complicated and can be extremely time-consuming to perform on a network..."* [57]

To reduce the time complexity of the state-of-the-art NKDV methods [47, 57, 72, 73], we propose a novel idea of augmenting each edge with distance statistics. Following this idea, we develop three efficient augmentation methods, namely (1) aggregate distance augmentation (ADA), (2) interval augmentation (IA), and (3) hybrid augmentation (HA), which wisely combines ADA and IA. Our theoretical analysis shows that all these methods can lower the time complexity for computing NKDV, compared with the state-of-the-art methods [47, 57, 72, 73]. Additionally, our experiments, based on five large-scale datasets, show that our methods ADA, IA and the best method HA (with the lowest time complexity) outperform the state-of-the-art methods [47, 57, 72, 73] by at least 5x to 10x speedup.

The rest of the paper is organized as follows. In Section 2, we formally define the problem of NKDV and present the baseline solutions for computing NKDV. Then, we proceed to discuss the three augmentation methods, which are ADA, IA and HA, and analyze their time complexity in Section 3. Next, we present the experimental results for all methods for computing NKDV with five large-scale datasets in Section 4. Then, we discuss the related work in Section 5. Lastly, we conclude and discuss the future work in Section 6.

2 PRELIMINARIES

In this section, we first formally define the concepts which are related to NKDV in Section 2.1. Then, we review the two-step framework [47, 72, 73] for computing the NKDV in Section 2.2. Lastly, we present two baseline methods, which are based on the two-step framework, namely range-query-based solution (RQS) [47, 72, 73] and shortest path sharing (SPS) [57], in Sections 2.3 and 2.4, respectively.

2.1 Basic Concepts for NKDV

Recall from Section 1, we need to obtain the kernel aggregation function value for each lixel (cf. Figure 4) in the road network in order to compute NKDV. Here, we first define the road network in Definition 1.

Definition 1. *A road network is a graph $G = (V, E)$ such that:*
(1) each node $\mathbf{v} \in V$ is augmented with the (x,y)-coordinate to represent its position.
(2) each edge $e \in E$ may contain one or more data points (events).

Then, we formally define the concept of lixel (cf. Definition 2), as shown in Figure 4. These lixels can be obtained by dividing each edge of the road network with the same length, e.g., 75m.

Definition 2. *(Lixel) Given a road network $G = (V, E)$, each edge $e \in E$ is divided into a set of small fixed-length segments. We represent each small segment as lixel, where its center point $\mathbf{q}$ ((x,y)-coordinate) represents the location of this lixel.*

Once we obtain the set of lixels, we can formally define the kernel aggregation function in network setting (cf. Definition 3). Here, we use Epanechnikov kernel (cf. Table 1) as an example.

Definition 3. *Given a road network $G = (V, E)$, a lixel with center point $\mathbf{q}$ in G and a set P of data points, we define the kernel aggregation function $\mathcal{F}_P(\mathbf{q})$ as:*

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w \cdot \begin{cases} 1 - \gamma^2 dist_G(\mathbf{q}, \mathbf{p_i})^2 & \text{if } dist_G(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where $dist_G(\mathbf{q}, \mathbf{p_i})$ is the shortest path distance from $\mathbf{q}$ to $\mathbf{p_i}$ and $\frac{1}{\gamma}$ denotes the bandwidth of the Epanechnikov kernel.

In the following sections, we refer to a lixel $\mathbf{q}$ with a lixel with center point $\mathbf{q}$ for convenience.

2.2 Two-Step Framework for Computing NKDV

In order to compute NKDV, existing studies [47, 72, 73] adopt a two-step framework (cf. Figure 5) to obtain $\mathcal{F}_P(\mathbf{q})$ for each lixel $\mathbf{q}$, which are (1) computing the shortest path distance from $\mathbf{q}$ to each node in V and (2) computing $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 2), based on the shortest path distance value from $\mathbf{q}$ to each $\mathbf{p_i}$.

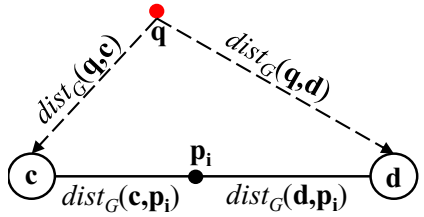


Figure 5: Illustration of two-step framework for computing $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 1), where $dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{q}, \mathbf{d})$ (dashed lines) are computed by the shortest path algorithm and $dist_G(\mathbf{c}, \mathbf{p_i})$ and $dist_G(\mathbf{d}, \mathbf{p_i})$ are augmented in the point $\mathbf{p_i}$ (solid line).

In the first step, we can utilize the shortest-path (SP) algorithm for obtaining the shortest path distance from $\mathbf{q}$ to each node in V, which takes $O(T_{\text{SP}})$ time. As a remark, even though existing studies [47, 72, 73] utilize the Dijkstra's algorithm [18], where $T_{\text{SP}} = |V| \log |V| + |E|$, for computing the shortest path distances, we can also utilize other efficient algorithms (e.g., [23, 40]) for finding the shortest path distances, which will be discussed in our related work (cf. Section 5). Once we obtain these distance values (e.g., $dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{q}, \mathbf{d})$ in Figure 5), we can compute $\mathcal{F}_P(\mathbf{q})$, based on obtaining the shortest path distance $dist_G(\mathbf{q}, \mathbf{p_i})$, using the following equation:

$$dist_G(\mathbf{q}, \mathbf{p_i}) = \min \begin{cases} dist_G(\mathbf{q}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_i}) \\ dist_G(\mathbf{q}, \mathbf{d}) + dist_G(\mathbf{d}, \mathbf{p_i}) \end{cases} \quad (3)$$

As such, the computational time in the second step takes $O(n)$ time for each lixel $\mathbf{q}$.

Suppose that we have L lixels in total in the graph G, this straightforward implementation (for step 1 and step 2) takes $O(L(T_{\text{SP}} + n))$ time for generating NKDV.

2.3 Baseline 1: Range-Query-based Solution (RQS)

Recall from Equation 2, each data point $\mathbf{p_i}$ can contribute to the kernel aggregation function $\mathcal{F}_P(\mathbf{q})$, only if $\mathbf{p_i}$ is within the bandwidth of the lixel $\mathbf{q}$, i.e., $dist_G(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma}$. Therefore, instead of finding the shortest path distances from the lixel $\mathbf{q}$ to all data points in the two-step framework, Xie et al. [72, 73] and Okabe et al. [47] modify the shortest path algorithm, namely SP_γ, which only finds those nodes from $\mathbf{q}$ that are within the bandwidth ($\frac{1}{\gamma}$) in step 1 (cf. yellow nodes in Figure 6), and then filters those data points that are larger than the bandwidth (i.e., black points in Figure 6) in step 2. In this paper, we term this method as RQS in which the pseudocode is described in Algorithm 1[1].

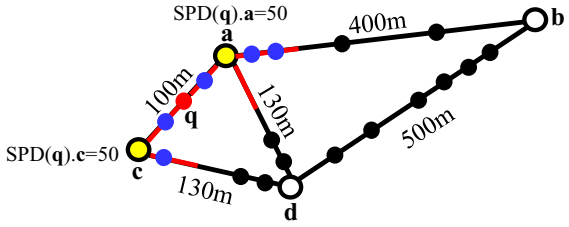


Figure 6: Illustration of using RQS for generating NKDV, where yellow nodes and blue points are within the bandwidth $\frac{1}{\gamma} = 100$ meters from $\mathbf{q}$ (red lines).

Algorithm 1 Range-Query-based Solution for NKDV

1: **procedure** RQS($G = (V, E)$, $P = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}\}$, weight w, parameter γ)
2: **for** each edge $e \in E$ **do**
3: **for** each lixel $\mathbf{q} \in e$ **do**
4: $SPD(\mathbf{q}) \leftarrow SP_\gamma(G, \mathbf{q})$
5: R.$\mathbf{q} \leftarrow 0$ ▷ Result for position $\mathbf{q}$
6: **for** each edge $\widetilde{e} := (\mathbf{c}, \mathbf{d}) \in E$ **do**
7: **if** $SPD(\mathbf{q}).\mathbf{c} \leq \frac{1}{\gamma}$ or $SPD(\mathbf{q}).\mathbf{d} \leq \frac{1}{\gamma}$ **then**
8: **for** each $\mathbf{p_i} \in \widetilde{e}$ **do**
9: $\Delta \leftarrow dist_G(\mathbf{q}, \mathbf{p_i})$ ▷ Equation 3
10: $R.\mathbf{q} \leftarrow R.\mathbf{q} + \max(1 - \gamma^2\Delta^2, 0)$

Even though RQS can improve the practical efficiency for generating NKDV, the response time can still be large, once we have a large bandwidth $\frac{1}{\gamma}$. Theoretically, the worst case time complexity of RQS is still in $O(L(T_{\text{SP}} + n))$, which is the same as the straightforward implementation of Equation 2, as the bandwidth can be arbitrarily large.

2.4 Baseline 2: Shortest Path Sharing (SPS)

In the two-step framework (cf. Section 2.2), we need to evaluate L-times shortest path algorithm for all lixels $\mathbf{q}$ in order to obtain every shortest path distance $dist_G(\mathbf{q}, \mathbf{p_i})$, which can be very time-consuming. Recently, Rakshit et al. [57] propose the shortest path sharing (SPS) method, which can significantly reduce the number of shortest path computations. Similar ideas, e.g., shortest path caching [35, 68], can be also found in the database community.

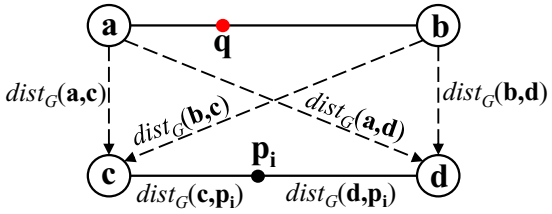


Figure 7: There are at most four possible routes for the lixel $\mathbf{q}$ to reach the data point $\mathbf{p_i}$.

Figure 7 summarizes the general idea of this method. Here, $\mathbf{q}$ and $\mathbf{p_i}$ can be in two edges[2], which are $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{c}, \mathbf{d})$, respectively. In this case, there are at most four routes for $\mathbf{q}$ to reach the data point $\mathbf{p_i}$, which are (1) $\mathbf{q} \to \mathbf{a} \to \mathbf{c} \to \mathbf{p_i}$, (2) $\mathbf{q} \to \mathbf{a} \to \mathbf{d} \to \mathbf{p_i}$, (3) $\mathbf{q} \to \mathbf{b} \to \mathbf{c} \to \mathbf{p_i}$ and (4) $\mathbf{q} \to \mathbf{b} \to \mathbf{d} \to \mathbf{p_i}$. As such, the shortest path distance between $\mathbf{q}$ and $\mathbf{p_i}$ is:

$$dist_G(\mathbf{q}, \mathbf{p_i}) = \min \begin{cases} dist_G(\mathbf{q}, \mathbf{a}) + dist_G(\mathbf{a}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_i}) \\ dist_G(\mathbf{q}, \mathbf{a}) + dist_G(\mathbf{a}, \mathbf{d}) + dist_G(\mathbf{d}, \mathbf{p_i}) \\ dist_G(\mathbf{q}, \mathbf{b}) + dist_G(\mathbf{b}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_i}) \\ dist_G(\mathbf{q}, \mathbf{b}) + dist_G(\mathbf{b}, \mathbf{d}) + dist_G(\mathbf{d}, \mathbf{p_i}) \end{cases} \quad (4)$$

Since $dist_G(\mathbf{c}, \mathbf{p_i})$ and $dist_G(\mathbf{d}, \mathbf{p_i})$ only depend on the position of the point $\mathbf{p_i}$ in the edge $(\mathbf{c}, \mathbf{d})$, they can compute these distance values and augment them in the point $\mathbf{p_i}$ in advance. Then, they adopt the modified shortest path algorithm, i.e., SP_γ (cf. Section 2.3), with the initial nodes $\mathbf{a}$ and $\mathbf{b}$ to obtain $dist_G(\mathbf{a}, \mathbf{c})$/$dist_G(\mathbf{a}, \mathbf{d})$ and $dist_G(\mathbf{b}, \mathbf{c})$/$dist_G(\mathbf{b}, \mathbf{d})$, respectively, i.e., the dash lines in Figure 7. Once they store all these shortest path distance values from nodes $\mathbf{a}$ and $\mathbf{b}$ in the memory, they can evaluate $dist_G(\mathbf{q}, \mathbf{p_i})$, for all lixels $\mathbf{q}$ in the edge $(\mathbf{a}, \mathbf{b})$, by only computing $dist_G(\mathbf{q}, \mathbf{a})$, $dist_G(\mathbf{q}, \mathbf{b})$ and performing the lookup operations for other distance values in Equation 4. Hence, instead of calling the SP_γ algorithm per lixel, they only need to call this algorithm per edge and reuse these distance values. Therefore, this method can reduce the worst case time complexity to $O(|E|T_{\text{SP}} + nL)$ (cf. Theorem 1), which is better than the method RQS (cf. Section 2.4). The detailed pseudocode is shown in Algorithm 2[3].

Theorem 1. *The time complexity of Algorithm 2 is $O(|E|T_{SP} + nL)$.*

3 FAST NKDV VIA AUGMENTATION

Even though existing methods RQS and SPS (cf. Sections 2.3 and 2.4, respectively) can improve the efficiency for using the two-step framework (cf. Section 2.2) to generate the NKDV, the worst case time complexity of the state-of-the-art method SPS [57] is still very high, which is $O(|E|T_{\text{SP}} + nL)$ (cf. Theorem 1). Moreover, the running time in step 2 of the methods RQS and SPS occupies

[1] In practical implementation, we can also sort the data points in advance for each edge, e.g., sort the data points $\mathbf{p_i}$ in the edge $(\mathbf{a}, \mathbf{b})$ of Figure 6, based on the ascending order of $dist_G(\mathbf{a}, \mathbf{p_i})$. Then, once we scan the data point $\mathbf{p_i}$ with $dist_G(\mathbf{q}, \mathbf{p_i}) > \frac{1}{\gamma}$ (e.g., the third point in the edge $(\mathbf{a}, \mathbf{b})$ in Figure 6), we can terminate the scanning process in this edge. To simplify the presentation, we omit this early-stop technique in line 10.

[2] We omit the case when $\mathbf{q}$ and $\mathbf{p_i}$ are in the same edge, since we can easily compute $dist_G(\mathbf{q}, \mathbf{p_i})$ in $O(1)$ time.

[3] As a remark, this method can further reduce the time complexity to $O(|V|T_{\text{SP}} + nL)$, once it computes and stores shortest path distance values for all pairs of nodes in advance. However, this approach needs $O(|V|^2)$ space for storing all these values, which is infeasible in practice.

Algorithm 2 Shortest-Path-Sharing-Based NKDV Algorithm

procedure SPS($G = (V, E)$, $P = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}\}$, weight w, parameter γ)
  for each edge $\mathbf{e} := (\mathbf{a}, \mathbf{b}) \in E$ **do**
    $SPD(\mathbf{a}) \leftarrow SP_\gamma(G, \mathbf{a})$, $SPD(\mathbf{b}) \leftarrow SP_\gamma(G, \mathbf{b})$
    for each lixel $\mathbf{q} \in e$ **do**
      $R.\mathbf{q} \leftarrow 0$
      for each edge $\widetilde{e} := (\mathbf{c}, \mathbf{d}) \in E$ **do**
        $\tau_\mathbf{a} \leftarrow \frac{1}{\gamma} - dist_G(\mathbf{q}, \mathbf{a})$, $\tau_\mathbf{b} \leftarrow \frac{1}{\gamma} - dist_G(\mathbf{q}, \mathbf{b})$
        if $SPD(\mathbf{a}).\mathbf{c} \le \tau_\mathbf{a}$ or $SPD(\mathbf{a}).\mathbf{d} \le \tau_\mathbf{a}$ or $SPD(\mathbf{b}).\mathbf{c} \le \tau_\mathbf{b}$ or $SPD(\mathbf{b}).\mathbf{d} \le \tau_\mathbf{b}$ **then**
          for each $\mathbf{p_i} \in \widetilde{e}$ **do**
            $\Delta \leftarrow dist_G(\mathbf{q}, \mathbf{p_i})$ ▷ Equation 4
            $R.\mathbf{q} \leftarrow R.\mathbf{q} + \max(1 - \gamma^2\Delta^2, 0)$
    $SPD(\mathbf{a}) \leftarrow \phi$, $SPD(\mathbf{b}) \leftarrow \phi$ ▷ Clear memory

more than 80% of the total time for computing NKDV in large-scale datasets (i.e., large n), which will be discussed in detail (cf. Figure 17) in Section 4. Therefore, we ask a question, can we further reduce the time complexity in step 2, i.e., $O(nL)$, in the two-step framework for generating the NKDV? In this section, we propose aggregate distance augmentation (ADA), interval augmentation (IA) and its combination, hybrid augmentation (HA), in Sections 3.1 - 3.3, which can significantly boost the efficiency for computing $\mathcal{F}_P(\mathbf{q})$ (cf. Equation 2), i.e., step 2. We theoretically show that all these methods can improve the time complexity for evaluating NKDV, against the state-of-the-art methods RQS (cf. Section 2.3) and SPS (cf. Section 2.4). We summarize the space and time complexity of all methods in Section 3.4.

3.1 Aggregate Distance Augmentation (ADA)

To illustrate the idea of aggregate distance augmentation (ADA), we let the point set of the edge e in graph G be $P(e)$.

Definition 4. *Given an edge e in the graph $G = (V, E)$, we let $P(e)$ be the set of points in the edge e.*

With the above definition, we can decompose the kernel aggregation function with respect to $P(e)$ for each edge e (cf. Lemma 1). Due to its simplicity, we omit the proof of this lemma.

Lemma 1. *Let $f_e(\mathbf{q})$ be the kernel aggregation function for the edge e, where:*

$$f_e(\mathbf{q}) = \sum_{\mathbf{p_i} \in P(e)} w \cdot \begin{cases} 1 - \gamma^2 dist_G(\mathbf{q}, \mathbf{p_i})^2 & \text{if } dist_G(\mathbf{q}, \mathbf{p_i}) \le \frac{1}{\gamma} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

We have:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{e \in E} f_e(\mathbf{q}) \quad (6)$$

Observe from Equation 6, since $\mathcal{F}_P(\mathbf{q})$ only depends on $f_e(\mathbf{q})$ for each edge e, we focus on evaluating $f_e(\mathbf{q})$ efficiently. Figure 8 illustrates the general idea of ADA. Here, we denote $P(\mathbf{c}, \mathbf{p_i})$ and $P(\mathbf{d}, \mathbf{p_i})$ to be two sets of points from $\mathbf{c}$ to $\mathbf{p_i}$ and $\mathbf{d}$ to $\mathbf{p_i}$, respectively. Observe that each point $\mathbf{p_i}$ is augmented by the aggregation of distance values with degree deg from the vertices $\mathbf{c}$ and $\mathbf{d}$ to $\mathbf{p_i}$, i.e., $a^{(deg)}_{P(\mathbf{c},\mathbf{p_i})}$ and $a^{(deg)}_{P(\mathbf{d},\mathbf{p_i})}$, respectively, where:

$$a^{(deg)}_{P(\mathbf{c},\mathbf{p_i})} = \sum_{\mathbf{p_j} \in P(\mathbf{c},\mathbf{p_i})} dist_G(\mathbf{c}, \mathbf{p_j})^{deg} \quad (7)$$

$$a^{(deg)}_{P(\mathbf{d},\mathbf{p_i})} = \sum_{\mathbf{p_j} \in P(\mathbf{d},\mathbf{p_i})} dist_G(\mathbf{d}, \mathbf{p_j})^{deg} \quad (8)$$

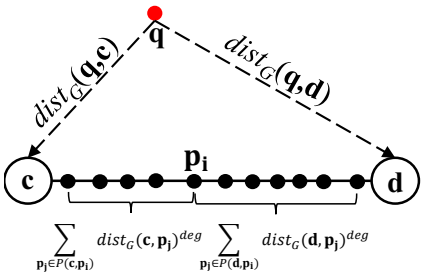


Figure 8: Aggregate distance augmentation (ADA) for each data point $\mathbf{p_i}$ in the edge $e = (\mathbf{c}, \mathbf{d})$, where deg is the degree (depends on the kernel function).

Table 2 summarizes which degrees of $a^{(deg)}_{P(\mathbf{c},\mathbf{p_i})}$ and $a^{(deg)}_{P(\mathbf{d},\mathbf{p_i})}$ we should augment for $\mathbf{p_i}$ in different kernel functions.

Table 2: Summarization of the augmented values for different kernel functions.

Kernel	Augmented values for each point $\mathbf{p_i}$	Chosen deg
Triangular	$a^{(1)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(1)}_{P(\mathbf{d},\mathbf{p_i})}$	1
Epanechnikov	$a^{(1)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(1)}_{P(\mathbf{d},\mathbf{p_i})}$, $a^{(2)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(2)}_{P(\mathbf{d},\mathbf{p_i})}$	1,2
Quartic	$a^{(1)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(1)}_{P(\mathbf{d},\mathbf{p_i})}$, $a^{(2)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(2)}_{P(\mathbf{d},\mathbf{p_i})}$ $a^{(3)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(3)}_{P(\mathbf{d},\mathbf{p_i})}$, $a^{(4)}_{P(\mathbf{c},\mathbf{p_i})}$, $a^{(4)}_{P(\mathbf{d},\mathbf{p_i})}$	1,2,3,4

Once we have these augmented values for each $\mathbf{p_i}$ (cf. Table 2), we can evaluate $f_e(\mathbf{q})$ in $O(\log |P(e)|)$ time (cf. Lemma 2), based on the binary search method [18], if we have obtained the shortest path distances from $\mathbf{q}$ to $\mathbf{c}$ and $\mathbf{q}$ to $\mathbf{d}$, i.e., $dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{q}, \mathbf{d})$, respectively.

Lemma 2. *Given the edge $e = (\mathbf{c}, \mathbf{d})$, a lixel $\mathbf{q}$ and the shortest path distance values $dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{q}, \mathbf{d})$, if we have the aggregate distance augmentation (cf. Table 2) and $D(\mathbf{p_i}) = dist_G(\mathbf{c}, \mathbf{p_i}) - dist_G(\mathbf{d}, \mathbf{p_i})$ for each $\mathbf{p_i}$, we can compute $f_e(\mathbf{q})$ in $O(\log |P(e)|)$ time, using the kernel functions in Table 1.*

Proof. In this proof, we focus on Epanechnikov kernel function. However, this proof can be easily extended to other kernel functions in Table 1. In order to obtain the correct proof for this lemma, we need to consider the following four possible cases for the shortest path distances, which are: (1) $dist_G(\mathbf{q}, \mathbf{c}) > \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) > \frac{1}{\gamma}$, (2) $dist_G(\mathbf{q}, \mathbf{c}) \le \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) > \frac{1}{\gamma}$, (3) $dist_G(\mathbf{q}, \mathbf{c}) > \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) \le \frac{1}{\gamma}$ and (4) $dist_G(\mathbf{q}, \mathbf{c}) \le \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) \le \frac{1}{\gamma}$.

Case 1 $\left(dist_G(\mathbf{q}, \mathbf{c}) > \frac{1}{\gamma} \text{ and } dist_G(\mathbf{q}, \mathbf{d}) > \frac{1}{\gamma}\right)$:

Recall from Equation 3, we have:

$$dist_G(\mathbf{q}, \mathbf{p_i}) = \min \begin{cases} dist_G(\mathbf{q}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_i}) \\ dist_G(\mathbf{q}, \mathbf{d}) + dist_G(\mathbf{d}, \mathbf{p_i}) \end{cases}$$
$$\ge \min(dist_G(\mathbf{q}, \mathbf{c}), dist_G(\mathbf{q}, \mathbf{d})) > \frac{1}{\gamma}$$

Hence, based on Equation 5, we can conclude that $f_e(\mathbf{q}) = 0$ and we can directly ignore all data points in this edge.

Case 2 $\left(dist_G(\mathbf{q}, \mathbf{c}) \leq \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) > \frac{1}{\gamma}\right)$:

Based on the condition of Equation 5, each point $\mathbf{p_i}$ can contribute to the value of $f_e(\mathbf{q})$ only if $dist_G(\mathbf{q}, \mathbf{p_i}) \leq \frac{1}{\gamma}$. Therefore, we only consider each point $\mathbf{p_i}$ such that:

1: the shortest path from $\mathbf{q}$ to $\mathbf{p_i}$ should pass through $\mathbf{c}$ (but not $\mathbf{d}$).

2: $dist_G(\mathbf{c}, \mathbf{p_i}) \leq \frac{1}{\gamma} - dist_G(\mathbf{q}, \mathbf{c})$

From the second condition, we need to find the position of the data point $\mathbf{p_l}$ such that it just fulfills this condition (cf. Figure 9). Since each point $\mathbf{p_i}$ is augmented by two distance values $dist_G(\mathbf{c}, \mathbf{p_i})$ and $dist_G(\mathbf{d}, \mathbf{p_i})$ (cf. Figure 5), we can use the binary search method [18] to obtain this data point $\mathbf{p_l}$, with $O(\log |P_e|)$ time.

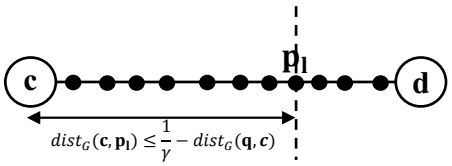


Figure 9: All data points from node c to the data point $\mathbf{p_l}$ can contribute to $f_e(\mathbf{q})$.

Once we have known the position of $\mathbf{p_l}$, we can compute $f_e(\mathbf{q})$ for this edge $e = (\mathbf{c}, \mathbf{d})$:

$$
\begin{aligned}
& f_e(\mathbf{q}) \\
= & \sum_{\mathbf{p_i} \in P(\mathbf{c}, \mathbf{p_l})} w \cdot (1 - \gamma^2 dist_G(\mathbf{q}, \mathbf{p_i})^2) \\
= & \sum_{\mathbf{p_i} \in P(\mathbf{c}, \mathbf{p_l})} w \cdot (1 - \gamma^2 (dist_G(\mathbf{q}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_i}))^2) \\
= & \; w(1 - \gamma^2 dist_G(\mathbf{q}, \mathbf{c})^2) |P(\mathbf{c}, \mathbf{p_l})| - 2w\gamma^2 dist_G(\mathbf{q}, \mathbf{c}) a^{(1)}_{P(\mathbf{c}, \mathbf{p_l})} - w\gamma^2 a^{(2)}_{P(\mathbf{c}, \mathbf{p_l})}
\end{aligned}
$$

Since the values $a^{(1)}_{P(\mathbf{c}, \mathbf{p_l})}$ and $a^{(2)}_{P(\mathbf{c}, \mathbf{p_l})}$ can be computed in advance, we can reuse these values for evaluating $f_e(\mathbf{q})$. As such, computing $f_e(\mathbf{q})$ only takes $O(1)$ time.

Case 3 $\left(dist_G(\mathbf{q}, \mathbf{c}) > \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) \leq \frac{1}{\gamma}\right)$:

The proof for this case is same as the one in Case 2, except that we consider only the shortest path from $\mathbf{q}$ to the node $\mathbf{d}$, rather than $\mathbf{q}$ to the node $\mathbf{c}$.

Case 4 $\left(dist_G(\mathbf{q}, \mathbf{c}) \leq \frac{1}{\gamma}$ and $dist_G(\mathbf{q}, \mathbf{d}) \leq \frac{1}{\gamma}\right)$:

Observe from Figure 10, suppose that we can find $\mathbf{p_l}$ and $\mathbf{p_r}$ such that they just fulfill $dist_G(\mathbf{c}, \mathbf{p_l}) \leq \frac{1}{\gamma} - dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{d}, \mathbf{p_r}) \leq \frac{1}{\gamma} - dist_G(\mathbf{q}, \mathbf{d})$, respectively, and these two sets $P(\mathbf{c}, \mathbf{p_l})$ and $P(\mathbf{d}, \mathbf{p_r})$ do not have any intersection. Then, we can conclude that $f_e(\mathbf{q})$ can be computed in $O(\log |P(e)|)$ time, based on the combination of the solutions of Cases 2 and 3.

However, unlike Figure 10, it is possible that these two sets $P(\mathbf{c}, \mathbf{p_l})$ and $P(\mathbf{d}, \mathbf{p_r})$ may have the intersection with each other (cf. Figure 11). As such, for each point in between $\mathbf{p_r}$ and $\mathbf{p_l}$, it is inconclusive which node ($\mathbf{c}$ or $\mathbf{d}$) the shortest path (from $\mathbf{q}$) to this point should pass through. In this case, there exist the point $\mathbf{p_v}$ in between $\mathbf{p_r}$ and $\mathbf{p_l}$ such that the following inequality just holds:

$$dist_G(\mathbf{q}, \mathbf{c}) + dist_G(\mathbf{c}, \mathbf{p_v}) \leq dist_G(\mathbf{q}, \mathbf{d}) + dist_G(\mathbf{d}, \mathbf{p_v})$$

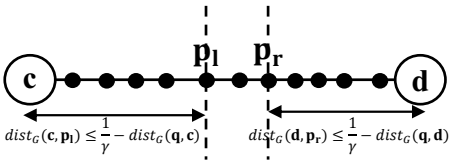


Figure 10: No intersection between two sets $P(\mathbf{c}, \mathbf{p_l})$ and $P(\mathbf{d}, \mathbf{p_r})$.

Here, we regard $D(\mathbf{p_v}) = dist_G(\mathbf{c}, \mathbf{p_v}) - dist_G(\mathbf{d}, \mathbf{p_v})$, we only need to find $\mathbf{p_v}$ such that this inequality $D(\mathbf{p_v}) \leq dist_G(\mathbf{q}, \mathbf{d}) - dist_G(\mathbf{q}, \mathbf{c})$ just holds. Once we have stored the value $D(\mathbf{p_i})$ for each point $\mathbf{p_i}$ in advance, we can utilize the binary search method to find this $\mathbf{p_v}$ in $O(\log(|P_e|))$ time.

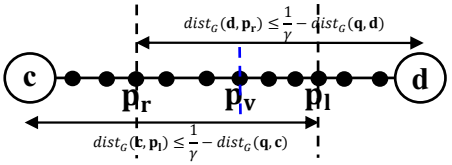


Figure 11: The sets $P(\mathbf{c}, \mathbf{p_l})$ and $P(\mathbf{d}, \mathbf{p_r})$ have the intersection.

After we have found this $\mathbf{p_v}$, we can compute $f_e(\mathbf{q})$ in $O(1)$ time, by adopting the similar approach in Case 2. □

Based on Lemma 2, we do not need to evaluate each point $\mathbf{p_i}$ one by one in each edge e, which can significantly improve the efficiency for computing NKDV. Algorithm 3 shows the detailed pseudocode of this method.

Algorithm 3 Aggregate Distance Augmentation Algorithm

```
1:  procedure ADA(G = (V, E), P = {p_1, p_2, ..., p_n})
2:      //Preprocessing (Used for f_e(q))
3:      for each edge e := (c, d) ∈ E do
4:          for each point p_i ∈ P(e) do
5:              Compute a^(deg)_P(c,p_i)                    ▷ Table 2
6:              Compute a^(deg)_P(d,p_i)                    ▷ Table 2
7:              Compute D(p_i) := dist_G(c, p_i) − dist_G(d, p_i)
8:      //Computing NKDV
9:      for each edge e := (a, b) ∈ E do
10:         SPD(a) ← SP_γ(G, a), SPD(b) ← SP_γ(G, b)
11:         for each lixel q ∈ e do
12:             R.q ← 0                                     ▷ Result for position q
13:             for each edge e ∈ E do
14:                 R.q ← R.q + f_e(q)                      ▷ Lemma 2
15:         SPD(a) ← φ, SPD(b) ← φ                          ▷ Clear memory
```

In Theorem 2, we formally show that ADA (cf. Algorithm 3) only takes $O\big(|E|\big(T_{\text{SP}} + L\log\big(\frac{n}{|E|}\big)\big)\big)$ time, which can be significantly faster than the existing methods RQS (cf. Section 2.3) and SPS (cf. Section 2.4).

Theorem 2. *The time complexity of Algorithm 3 is* $O\big(|E|\big(T_{SP} + L\log\big(\frac{n}{|E|}\big)\big)\big)$.

Proof. In this proof, we mainly focus on showing that the time complexity for the evaluation of one lixel $\mathbf{q}$ (cf. lines 12-14 in Algorithm 3) is $O\big(|E|\log\big(\frac{n}{|E|}\big)\big)$, given the known shortest path distances (cf line 10 in Algorithm 3).

Recall from Lemma 1, for each edge $e \in E$, it takes $O(\log |P(e)|)$ time for evaluating $f_e(\mathbf{q})$. Therefore, the time complexity for evaluating $f_e(\mathbf{q})$ with all edges in E takes $O(\sum_{e\in E} \log |P(e)|) = O(\log(\prod_{e\in E} |P(e)|))$ time. Since the number of points in the road network is n, we also have the constraint $\sum_{e\in E} |P(e)| = n$. To analyze the worst case time complexity of our method, we need to find the maximum possible value of $\log(\prod_{e\in E} |P(e)|)$, which can be formulated as the following optimization problem (T).

$$\begin{aligned} \max_{P(e)} \quad & \log\Big(\prod_{e\in E} |P(e)|\Big) \\ \text{such that} \quad & \sum_{e\in E} |P(e)| = n \\ & |P(e)| \geq 0 \text{ for each } e \in E \end{aligned} \tag{T}$$

Based on the AM–GM inequality [64], we have:

$$\frac{\sum_{e\in E} |P(e)|}{|E|} \geq \sqrt[|E|]{\prod_{e\in E} |P(e)|} \iff \prod_{e\in E} |P(e)| \leq \Big(\frac{n}{|E|}\Big)^{|E|}$$

Therefore, we have:

$$\log\Big(\prod_{e\in E} |P(e)|\Big) \leq \log\Big(\frac{n}{|E|}\Big)^{|E|} = |E| \log\Big(\frac{n}{|E|}\Big)$$

Therefore, we can show that the worst case time complexity is $O(|E| \log(\frac{n}{|E|}))$ for each lixel $\mathbf{q}$, given the known shortest path distances (cf. line 10 in Algorithm 3). As a remark, we can theoretically achieve this worst case time complexity, if the number of points in each edge $e \in E$ is the same, i.e., $|P(e)| = \frac{n}{|E|}$, and the bandwidth $\frac{1}{\gamma} \to \infty$.

Since there are in total L lixels in the road network G, the time complexity for lines 11 to 14 in Algorithm 3 is $O(L|E| \log(\frac{n}{|E|}))$. By combining the time complexity ($O(|E|T_{\text{SP}})$) for computing the shortest path distances, we can prove that Algorithm 3 only takes $O(|E|(T_{\text{SP}} + L \log(\frac{n}{|E|})))$ time. □

As a remark, in terms of big-O notation, we know that:

$$\begin{aligned} O\Big(\log\Big(\frac{n}{|E|}\Big)\Big) &< O\Big(\frac{n}{|E|}\Big) \\ O\Big(|E|L\log\Big(\frac{n}{|E|}\Big)\Big) &< O(nL) \end{aligned}$$

Therefore, we know that the time complexity of ADA (cf. Theorem 2) is theoretically faster than SPS (cf. Theorem 1) and RQS (with $O(L(T_{SP} + n))$ time).

3.2 Interval-based Augmentation (IA)

Even though ADA can significantly reduce the time complexity for evaluating NKDV, this method still needs to perform the binary search for each edge, which incurs $O(\log(\frac{n}{|E|}))$ term in the time complexity (cf. Theorem 2). In this section, we develop the method, called interval-based augmentation (IA), which can further remove this logarithmic factor, at the expense of a higher preprocessing time and space.

Observe from Figures 12 and 13, we first obtain the smallest distance between any two consecutive data points in each edge ($dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ in Figure 12) and then obtain the intervals, $I_1, I_2, ..., I_N$, with the same length (cf. Figure 13), except for the last interval I_N, where:

$$N = \left\lceil \frac{dist_G(\mathbf{c}, \mathbf{d})}{dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})} \right\rceil \tag{9}$$

With this set of intervals, we can observe that each interval can only contain at most one data point (cf. Figure 13). In Lemma 3, we claim that it is always true. As a remark, we omit the case that the interval I_N can have smaller length, compared with other intervals in edge $e = (\mathbf{c}, \mathbf{d})$, in Lemma 3, due to simplicity.

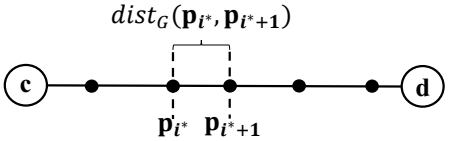


Figure 12: The smallest distance $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ between two consecutive data points in the edge $e = (\mathbf{c}, \mathbf{d})$.

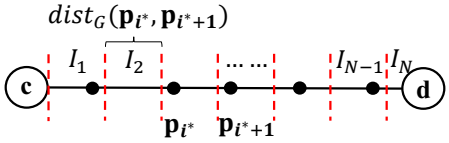


Figure 13: Augment the intervals $I_1, I_2, ..., I_N$ (with length $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ (except for I_N)) from the node c in the edge $e = (\mathbf{c}, \mathbf{d})$.

Lemma 3. *Suppose that we have a set of equal-length and contiguous intervals in the edge $e = (\mathbf{c}, \mathbf{d})$, where the length of each interval is $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ (the smallest distance between any two consecutive points in $P(e)$), each interval can only contain at most one data point in $P(e)$.*

Proof. Suppose that there exists an interval which contains more than one point. Therefore, there exist two data points, e.g., $\mathbf{p_i}$ and $\mathbf{p_j}$, such that they are in the same interval, i.e.,

$$dist_G(\mathbf{p_i}, \mathbf{p_j}) < dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$$

However, we know that $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ must be the smallest distance value between any two consecutive points in $P(e)$. As such, it leads to contradiction. □

Based on Lemma 3, given any distance value from the node $\mathbf{c}$ (or node $\mathbf{d}$), instead of using the binary search to find the data point $\mathbf{p_i}$ in ADA (cf. Section 3.1), we can use $O(1)$ time to identify the interval, since the interval size is fixed ($dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$, except for I_N). Once we further augment the following distance aggregation values[4], i.e., $a^{(deg)}_{P(\cup_{v=1}^{k} I_v)}$ (cf. Equation 10) and $a^{(deg)}_{P(\cup_{v=k}^{N} I_v)}$ (cf. Equation 11), which are the analogy of Equations 7 and 8, respectively, for each interval I_k in the edge e, we can further reduce the time complexity for computing $f_e(\mathbf{q})$ from $O(\log |P(e)|)$ time (cf. Lemma 2) to $O(1)$ time (cf. Lemma 4).

$$a^{(deg)}_{P(\cup_{v=1}^{k} I_v)} = \sum_{\mathbf{p_j} \in \cup_{v=1}^{k} I_v} dist_G(\mathbf{c}, \mathbf{p_j})^{deg} \tag{10}$$

$$a^{(deg)}_{P(\cup_{v=k}^{N} I_v)} = \sum_{\mathbf{p_j} \in \cup_{v=k}^{N} I_v} dist_G(\mathbf{d}, \mathbf{p_j})^{deg} \tag{11}$$

[4]We can choose the same degrees based on Table 2 for different kernel functions. For example, we choose $deg = 1, 2$ for Epanechnikov kernel.

Lemma 4. *Given the edge $e = (\mathbf{c}, \mathbf{d})$, a lixel $\mathbf{q}$ and the shortest path distance values $dist_G(\mathbf{q}, \mathbf{c})$ and $dist_G(\mathbf{q}, \mathbf{d})$, if we have the interval-based augmentation (cf. Equations 10 and 11), we can compute $f_e(\mathbf{q})$ in O(1) time, using the kernel functions in Table 1.*

We omit the proof of Lemma 4, as the proof is similar to the one in Lemma 2. Instead, we focus on finding the interval (with $O(1)$ time, based on Lemma 3) for those four cases. Here, we summarize our method IA in Algorithm 4. Theorem 3 illustrates the time complexity of this algorithm.

Algorithm 4 Interval-based Augmentation Algorithm (IA)

1: **procedure** IA($G = (V, E)$,$P = \{\mathbf{p_1}, \mathbf{p_2}, ..., \mathbf{p_n}\}$)
2: //Preprocessing (Used for $f_e(\mathbf{q})$)
3: **for** each edge $e := (\mathbf{a}, \mathbf{b}) \in E$ **do**
4: Compute $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}})$ ▷ $O(|P(e)|)$ time
5: Compute $N(e)$ ▷ Equation 9
6: **for** $k \leftarrow 1$ to $N(e)$ **do**
7: Compute $a^{(deg)}_{P(\cup_{v=1}^{k} I_v)}$ ▷ Table 2
8: Compute $a^{(deg)}_{P(\cup_{v=k}^{N} I_v)}$ ▷ Table 2
9: //Computing NKDV (Same as lines 9-15 in Algorithm 3)

Theorem 3. *Given the number $N(e)$ of intervals for each edge $e \in E$ (cf. Equation 9), the time complexity of Algorithm 4 is $O(|E|(T_{SP} + L) + \sum_{e \in E} N(e))$.*

Compared with the time complexity of ADA, which takes $O\big(|E|\big(T_{\text{SP}} + L \log\big(\frac{n}{|E|}\big)\big)\big)$ (cf. Theorem 2), IA can further remove this $\log\big(\frac{n}{|E|}\big)$ term, with the additional construction (or preprocessing) cost $\sum_{e \in E} N(e)$ for intervals. Once the construction cost $\sum_{e \in E} N(e)$ is low, IA can further improve the efficiency for generating NKDV, compared with the method ADA.

3.3 Hybrid Augmentation (HA)

As discussed in Section 3.2, IA needs to augment the intervals for each edge $e \in E$. Observe from Figure 14, once the length of the road is very long and the smallest distance between two consecutive points is very small, we need to augment many intervals $N(e)$ into this edge e (cf. Equation 9) for using the method IA, which can incur both high preprocessing time and space.

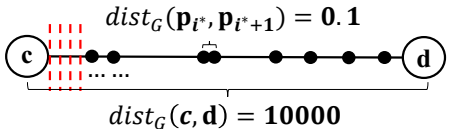


Figure 14: Many intervals (e.g., 100000) can be augmented in the edge with (1) the long length of the road (e.g., $dist_G(\mathbf{c}, \mathbf{d}) = 10000$) and (2) the small distance value between two consecutive points with the smallest distance (e.g., $dist_G(\mathbf{p_{i^*}}, \mathbf{p_{i^*+1}}) = 0.1$).

In this section, we develop a method, called hybrid augmentation (HA), that can wisely select which method, ADA (cf. Section 3.1) or IA (cf. Section 3.2), we should choose for each edge $e \in E$, based on the estimation of the minimum cost. Recall from Section 3.1, ADA takes $O(\log(|P(e)|))$ time in the worst case to evaluate $f_e(\mathbf{q})$ (cf. Lemma 2) for each lixel $\mathbf{q}$ and there are totally L lixels. As such, we can model the cost for this edge e as:

$$Cost(ADA(e)) = L \times \log(|P(e)|) \tag{12}$$

On the other hand, IA takes $N(e)$ time to construct the intervals in edge e and takes O(1) time to obtain the interval for each lixel $\mathbf{q}$. Therefore, we can model the cost for this edge e in the worst case as:

$$Cost(IA(e)) = N(e) + L \tag{13}$$

Based on these cost functions, HA chooses the method for each edge e which can provide minimum cost, i.e., $\min(Cost(ADA(e)), Cost(IA(e)))$, in advance. Even though this combination is simple, we can theoretically show that HA can further reduce the worst case time complexity of evaluating NKDV (cf. Theorem 4).

Theorem 4. *Given the number $N(e)$ of intervals for each edge $e \in E$ (cf. Equation 9), the time complexity of HA is $O\big(|E|T_{SP} + \min\big(L|E| \log\big(\frac{n}{|E|}\big), L|E| + \sum_{e \in E} N(e)\big)\big)$.*

Proof. Since HA can choose the edge with minimum cost in advance, we have the following cost (in the worst case):

$$\begin{aligned}
& \sum_{e \in E} \min(Cost(ADA(e)), Cost(IA(e))) \\
= & \sum_{e \in E} \min(L \log(|P(e)|), N(e) + L) \\
\leq & \min\Big(\sum_{e \in E} L \log(|P(e)|), \sum_{e \in E} (N(e) + L)\Big) \\
\leq & \min\Big(L \times |E| \log\Big(\frac{n}{|E|}\Big), L|E| + \sum_{e \in E} N(e)\Big)
\end{aligned}$$

Here, the last inequality is based on the proof in Theorem 2. Since we know $O(|E|T_{\text{SP}})$ is the worst case time complexity in step 1 (cf. Section 2.4) and $O(\sum_{e \in E} \min(Cost(ADA(e)), Cost(IA(e))))$ is the worst case time complexity in step 2 in the two-step framework, we can conclude that HA takes $O\big(|E|T_{\text{SP}} + \min\big(L|E| \log\big(\frac{n}{|E|}\big), L|E| + \sum_{e \in E} N(e)\big)\big)$ time for computing NKDV. □

3.4 Summarization of the Theoretical Results of All Methods

In this section, we summarize the worst case time and space complexity of all methods. Here, we denote $\mathcal{I}_{\text{SP}}$ as the space for the SP algorithm, e.g., $\mathcal{I}_{\text{SP}} = O(|V|)$ for Dijkstra's algorithm [18]. Observe from Table 3, once we restrict the space consumption to be $O(|V| + |E| + n + \mathcal{I}_{\text{SP}})$, which can be much smaller than $O(|V| + |E| + n + \mathcal{I}_{\text{SP}} + \sum_{e \in E} N(e))$, the method ADA is theoretically the most efficient algorithm, compared with other methods. Moreover, once the term $\sum_{e \in E} N(e)$ is very large, ADA can be theoretically faster than IA, as IA needs to use $O(\sum_{e \in E} N(e))$ time to construct the intervals for each edge e.

However, as discussed in Section 3.2, if the construction cost $\sum_{e \in E} N(e)$ is small, the time complexity of IA, i.e., $O(|E|(T_{\text{SP}} + L) + \sum_{e \in E} N(e))$, can be smaller than the time complexity of the method ADA, i.e., $O\big(|E|\big(T_{\text{SP}} + L \log\big(\frac{n}{|E|}\big)\big)\big)$, at the expense of higher space complexity $O(|V| + |E| + n + \mathcal{I}_{\text{SP}} + \sum_{e \in E} N(e))$.

By combining both the advantages of ADA and IA, HA can theoretically provide the smallest time complexity for evaluating NKDV (cf. Table 3).

Table 3: Time and space complexity of all methods for computing NKDV.

Method	Time complexity	Space complexity	Ref.
RQS	$O(L(T_{\text{SP}} + n))$	$O(\|V\| + \|E\| + n + \mathcal{I}_{\text{SP}})$	Section 2.3 [47, 72, 73]
SPS	$O(\|E\|T_{\text{SP}} + nL)$	$O(\|V\| + \|E\| + n + \mathcal{I}_{\text{SP}})$	Section 2.4 [57]
ADA	$O\big(\|E\|\big(T_{\text{SP}} + L\log\big(\frac{n}{\|E\|}\big)\big)\big)$	$O(\|V\| + \|E\| + n + \mathcal{I}_{\text{SP}})$	Section 3.1
IA	$O(\|E\|(T_{\text{SP}} + L) + \sum_{e\in E} N(e))$	$O(\|V\| + \|E\| + n + \mathcal{I}_{\text{SP}} + \sum_{e\in E} N(e))$	Section 3.2
HA	$O\big(\|E\|T_{\text{SP}} + \min\big(L\|E\|\log\big(\frac{n}{\|E\|}\big), L\|E\| + \sum_{e\in E} N(e)\big)\big)$	$O(\|V\| + \|E\| + n + \mathcal{I}_{\text{SP}} + \sum_{e\in E} N(e))$	Section 3.3

4 EXPERIMENTAL EVALUATION

We first introduce the experimental setting in Section 4.1. Then, we compare the efficiency performance of all methods in Section 4.2, using Epanechnikov kernel. After that, we further investigate the efficiency performance of all methods in other kernel functions, e.g., triangular and quartic kernels, in Section 4.3.

4.1 Experimental Setting

We utilize three categories, police call, crime and traffic accident, of large-scale real point datasets (with latitude and longitude for each point) in our experiments. These datasets are the open data (last accessed: 15$^{\text{th}}$ October, 2020) from the local governments of different cities, including Johns Creek [3], Seattle [9], Los Angeles [4] and New York [5]. For each point dataset, we extract the corresponding road network from the OpenStreetMap [6]. After that, we integrate the point dataset into the road network, using the software OSMnx[5] [11]. To further test the scalability of different methods, we also randomly generate a large-scale synthetic dataset with four million data points and integrate it into the New York road network. Table 4 summarizes the details of the integrated datasets.

Table 4: Integrated datasets.

Dataset	n	|V|	|E|	Category
Johns Creek [3]	609423	3529	8124	Police call
Seattle [9]	862873	11371	31433	Crime
Los Angeles [4]	1255668	36715	111670	Crime
New York [5]	1294779	41467	116081	Traffic accident
New York$_{\text{Synthetic}}$	4000000	41467	116081	Synthetic

In our experiments, we compare the baseline solutions RQS [47, 72, 73] and SPS [57] with our methods, ADA, IA and HA, as shown in Table 3. We implemented all methods[6] with C++ and conducted experiments on an Intel i7 3.19GHz PC with 32GB memory. In this paper, we use the response time (sec) to measure the efficiency of all methods and only report the results in which the response time is smaller than 14400sec (i.e., 4 hours).

4.2 Efficiency Performance of NKDV

Even though our methods can theoretically achieve better efficiency for computing NKDV (cf. Section 3.4), we do not know the practical improvement of our methods, compared with the state-of-the-art methods (i.e., RQS and SPS in Table 3). In this section, we investigate the practical efficiency of all methods for computing NKDV.

Response time of all methods under the default setting of parameters: In this experiment, we follow [72] and choose 1000m as the default bandwidth, i.e., $\frac{1}{\gamma} = 1000$ and $\gamma = 0.001$, and 10m as the default lixel size for testing. Table 5 summarizes the response time of all methods. Since ADA, IA and HA achieve lower time complexity for solving NKDV (cf. Table 3), these three methods achieve at least 5x speedup, compared with RQS and SPS. We omit the results of RQS and SPS in the New York$_{\text{Synthetic}}$ dataset, since these methods could not generate NKDV within 14400sec.

Table 5: Response time (sec) of all methods with $\gamma = 0.001$ and 10m lixel size.

Dataset	RQS	SPS	ADA	IA	HA
Johns Creek	55.51	52.31	7.49	**7.14**	7.35
Seattle	894.35	848.43	193.04	**174.98**	188.51
Los Angeles	12185.39	10818.48	2017.88	**1842.09**	1868.93
New York	13233.35	11526.83	2116.32	**1969.71**	1976.91
New York$_{\text{Synthetic}}$	n.a.	n.a.	5128.63	**2177.61**	2719.18

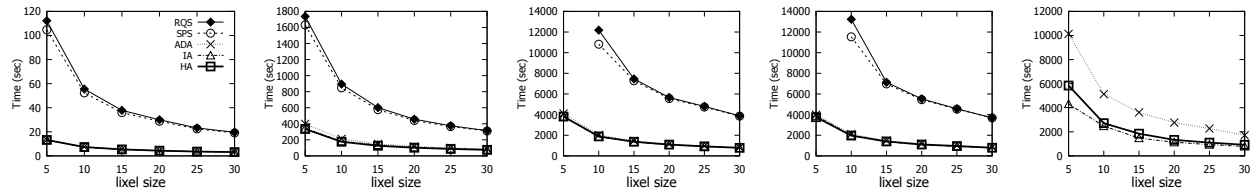
Varying the lixel size: In this experiment, we vary the lixel size (5m, 10m, 15m, 20m, 25m and 30m) and measure the response time of all methods (cf. Table 3). Here, we adopt the default value $\gamma = 0.001$. In Figure 15, once we vary the lixel size from 30m to 5m, the number of lixels in each edge of the graph G increases. Therefore, the response time of all methods also increases. However, since ADA, IA and HA achieve much smaller time complexity, compared with the methods RQS and SPS, these three methods can provide 5x to 10x speedup in all datasets.

Varying the parameter γ (or bandwidth of kernel function): We proceed to investigate how the parameter γ affects the efficiency performance of different methods. We choose five γ values, 0.00005, 0.0001, 0.0002, 0.0005 and 0.001, which correspond to the bandwidth values, 20000m, 10000m, 5000m, 2000m and 1000m, of the kernel function (bandwidth = $\frac{1}{\gamma}$), respectively, in this experiment. With the smaller γ value, i.e., higher bandwidth value, each algorithm needs to process more data points and edges. As such, once we vary γ from 0.001 to 0.00005, the response time of all methods can increase (cf. Figure 16). Observe that no matter which γ we choose in this range, our methods, ADA, IA and HA can significantly outperform the state-of-the-art methods, RQS and SPS, by at least 5x (large γ) to 71x (small γ) in all datasets.

Distribution of the response time in different methods: In this section, we further investigate the response time of two components, i.e., shortest path computation in step 1 and $\mathcal{F}_P(\mathbf{q})$ computation in step 2, in the two-step framework (cf. Section 2.2) for all methods. Here, we use two datasets, Johns Creek and Seattle, with $\gamma = 0.001$ and $\gamma = 0.0001$ (i.e., 1000m and 10000m bandwidth values, respectively), for testing. Observe from Figure 17, the computation time in step 1 (shortest path computation) of RQS method is normally much smaller, compared with step 2 ($\mathcal{F}_P(\mathbf{q})$ computation), where the response time of step 2 occupies more than 80% of the overall response time. The main reason is that the method RQS needs to scan more data points, compared with the vertices and edges, as the number n of data points is much larger than the numbers of vertices and edges, i.e., $|V|$ and $|E|$, respectively, in these

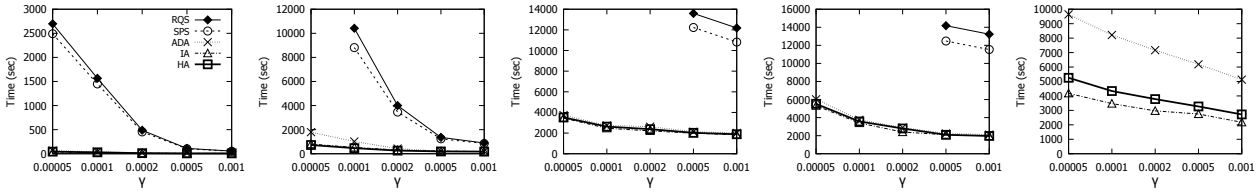
[5]https://github.com/gboeing/osmnx

[6]The source codes of all methods can be found in this Github repository https://github.com/edisonchan2013928/Network-Kernel-Density-Visualization-NKDV-Code.



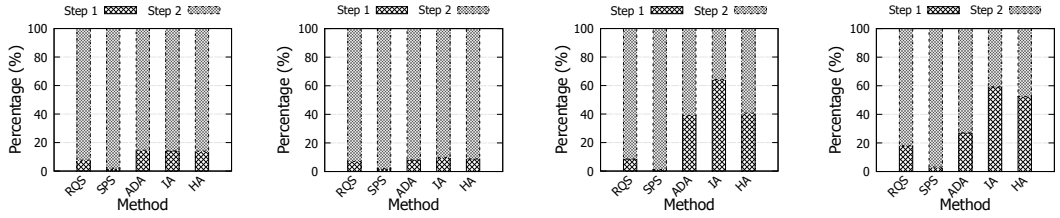
(a) Johns Creek (b) Seattle (c) Los Angeles (d) New York (e) New York$_{\text{Synthetic}}$

Figure 15: Response time for NKDV with $\gamma = 0.001$, varying the lixel size.



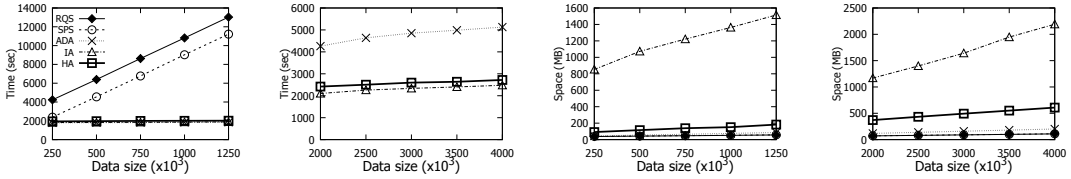
(a) Johns Creek (b) Seattle (c) Los Angeles (d) New York (e) New York$_{\text{Synthetic}}$

Figure 16: Response time for NKDV with 10m lixel size, varying the parameter γ (or the bandwidth of kernel function).



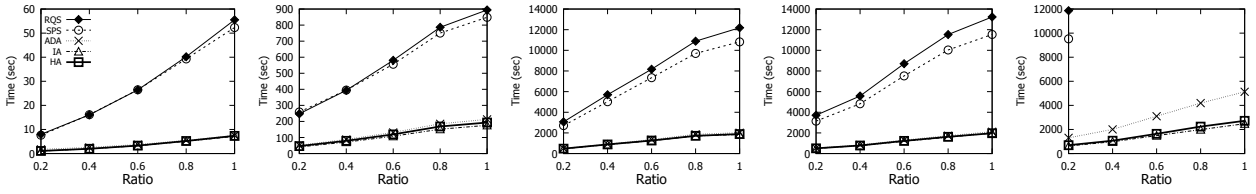
(a) Johns Creek ($\gamma = 0.001$) (b) Seattle ($\gamma = 0.001$) (c) Johns Creek ($\gamma = 0.0001$) (d) Seattle ($\gamma = 0.0001$)

Figure 17: Percentage of the response time for step 1 and step 2 in the two-step framework for all methods, with $\gamma = 0.001$ (1000m bandwidth) and $\gamma = 0.0001$ (10000m bandwidth).



(a) New York (time) (b) New York$_{\text{Synthetic}}$ (time) (c) New York (space) (d) New York$_{\text{Synthetic}}$ (space)

Figure 18: Response time (a and b) and space consumption (c and d) for all methods with $\gamma = 0.001$ and 10m lixel size, varying the data size.

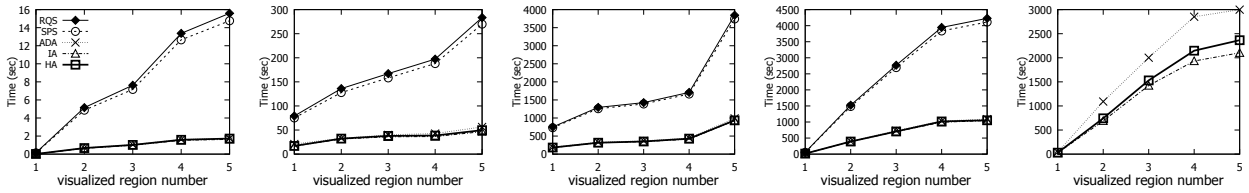


(a) Johns Creek (b) Seattle (c) Los Angeles (d) New York (e) New York$_{\text{Synthetic}}$

Figure 19: Response time for NKDV with $\gamma = 0.001$ and 10m lixel size, varying the ratio (i.e., size) of the rectangle (i.e., visualized region).

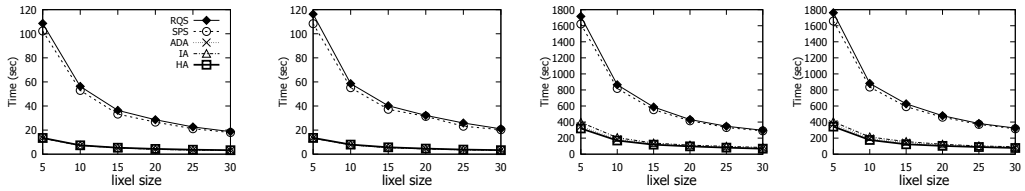
two road networks (cf. Table 4). Therefore, even though SPS can reduce the time complexity in step 1, SPS does not show significant improvement in the efficiency performance, compared with the RQS method (cf. Figures 15 and 16). On the other hand, since ADA, IA and HA can significantly improve the efficiency performance in step 2, the percentage of response time in step 1 for these methods significantly increases (cf. Figure 17).

Varying the size of the dataset: In this experiment, we test how the size of the dataset affects the response time and the space consumption of all methods. Here, we use two datasets for testing, which are New York and New York$_{\text{Synthetic}}$. In the New York dataset, we randomly sample five subsets of data points, where the sample sizes are 250K, 500K, 750K, 1000K and 1250K. In the New York$_{\text{Synthetic}}$ dataset, we randomly generate 2000K, 2500K, 3000K,



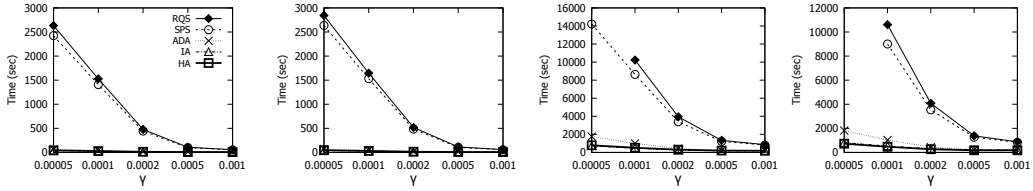
(a) Johns Creek (b) Seattle (c) Los Angeles (d) New York (e) New York$_{\text{Synthetic}}$

Figure 20: Response time for NKDV with $\gamma = 0.001$ and 10m lixel size, using different rectangles (i.e., visualized regions), which are generated by uniform distribution.



(a) Johns Creek (Triangular) (b) Johns Creek (Quartic) (c) Seattle (Triangular) (d) Seattle (Quartic)

Figure 21: Response time for NKDV with $\gamma = 0.001$, varying the lixel size and using triangular and quartic kernel functions.



(a) Johns Creek (Triangular) (b) Johns Creek (Quartic) (c) Seattle (Triangular) (d) Seattle (Quartic)

Figure 22: Response time for NKDV with 10m lixel size, varying the parameter γ and using triangular and quartic kernel functions.

3500K and 4000K data points for testing. By default, we choose the parameter $\gamma = 0.001$ and the lixel size as 10m. Observe from Figure 18a and Figure 18b, once we increase the size of the dataset, all methods need to process more data points. As such, the response time of all methods increase. Since all our methods ADA, IA and HA are not linearly scalable to the number of data points n (cf. Table 3), the response time of ADA, IA and HA does not significantly increase, compared with the methods RQS and SPS. Observe that once the data size is larger (e.g., New York$_{\text{Synthetic}}$ dataset), IA and HA can further outperform ADA by a visible margin. Figure 18c and Figure 18d report the space consumption of all methods. Compared with other methods, we observe that IA consumes much more memory. Since HA further improves the efficiency, compared with ADA, without consuming huge amounts of memory space, we consider HA, which combines both the advantages of ADA and IA, to be the best method for large-scale datasets.

Varying the size of the visualized region: In practice, the users can also zoom in and visualize the density of different sub-regions in a city. Therefore, we proceed to test the efficiency performance of all methods, by adopting different visualized regions.

In the first experiment, we first specify the minimum bounding rectangle, which covers the whole region of each city (e.g., Johns Creek in Table 4). This rectangle can represent the visualized region. Then, we vary the size of this rectangle by multiplying its height and width with these five values of ratio, 0.2, 0.4, 0.6, 0.8 and 1 (the original one). As a remark, we utilize the same center for all these five rectangles. With the smaller value of ratio, the rectangle (i.e., visualized region) can cover smaller number of edges (i.e., smaller number of lixels), which can reduce the response time of different methods. Observe from Figure 19, our methods ADA, IA and HA can achieve at least 5x to 10x speedup in all datasets, compared with the baseline solutions, RQS and SPS, in different sizes, by varying the ratio, of the visualized region.

In the second experiment, we randomly generate five rectangles (visualized regions), based on the uniform distribution, inside the minimum bounding rectangle of each city. Then, we report the response time of each method, using the increasing order of the visualized region size (i.e., smaller visualized region number means that this region has smaller size.), in Figure 20. Observe that our methods ADA, IA and HA can consistently achieve better efficiency in all datasets, compared with the state-of-the-art methods RQS and SPS, no matter which visualized regions we use.

4.3 NKDV with Other Kernel Functions

In this section, we further test the efficiency performance of all methods with other kernel functions, including triangular and quartic kernels. Here, we use two datasets, which are Johns Creek and Seattle, for testing. Observe from Figure 21, no matter which kernel function, either triangular or quartic kernel, we use, our methods ADA, IA and HA can also achieve at least 5x to 10x speedup with different lixel sizes, compared with other methods. In addition, since the time complexity of all methods (cf. Table 3) does not depend on the chosen kernel function in Table 1, we observe that the same method can provide similar response time (cf. Figure 21), using the triangular and quartic kernels.

We proceed to investigate how the response time of all methods changes, once we change the parameter γ of triangular and quartic kernel functions. Here, we fix the lixel size as 10m. Observe from Figure 22, once we vary the parameter γ from 0.001 to 0.00005, i.e., larger bandwidth value of the kernel functions, the response time of all methods increase. However, our methods ADA, IA and HA can significantly outperform the state-of-the-art methods, including RQS and SPS, by at least 5x to 73x for both triangular and quartic kernel functions. Since our methods ADA, IA and HA can reduce the time complexity for generating NKDV, we also expect that this time gap can also increase, compared with the baseline methods, once we further reduce the γ value.

5 RELATED WORK

Kernel density estimation (KDE) [24, 69] or kernel density visualization (KDV) [13] has been the de facto nonparametric statistical method for a wide range of applications in different domains, especially for hotspot detection (cf. Figure 1). However, KDE/KDV is a very time-consuming operation, which is not scalable to large-scale datasets [13, 53]. As such, many efficient algorithms have been developed for evaluating exact or approximate KDE/KDV. Zheng et al. [82–84] and Phillips et al. [54–56] utilize the sampling methods to reduce the size of the dataset and then apply the exact KDV method for the reduced dataset. On the other hand, Chan et al. [13–16], Gan et al. [22] and Gray et al. [25] develop the efficient and tight bound functions of $\mathcal{F}_P(\mathbf{q})$ for different kernel functions (e.g., Gaussian kernel). Raykar et al. [58] and Yang et al. [74] adopt the fast Gauss transform to efficiently and approximately compute $\mathcal{F}_P(\mathbf{q})$. Some researchers also utilize the modern hardware [24, 79] and parallel/distributed algorithms [82] to further boost the efficiency for computing $\mathcal{F}_P(\mathbf{q})$. Although most of these research studies can improve the efficiency of KDE/KDV, they only regard the events in the plane. However, most of these events (e.g., traffic accidents or crime events) are mainly in or alongside the road network [48]. As such, using this planar KDV can provide the inaccurate density estimation in many geographical applications [20, 30, 38, 48, 60, 73].

In order to provide more accurate density visualization, geographical researchers [47, 72] propose to only visualize the density in the road network and replace the Euclidean distance $dist(\mathbf{q}, \mathbf{p})$ by the shortest path distance $dist_G(\mathbf{q}, \mathbf{p})$ in the kernel functions (cf. Table 1), in which we term this approach as network kernel density visualization (NKDV). After they formally define NKDV, many criminologists [34, 60], geoscientists [30, 71] and business planners [45, 61, 78] have utilized NKDV to identify the hotspot region. Due to its wide range of applications, Okabe et al. [47] further develop the plug-in, called SANET [8], for the ArcGIS software [1], which can support the evaluation of NKDV. However, like KDV, computing NKDV is computationally expensive, which has been complained by many recent studies [43, 57, 78]. Even though many efficient algorithms have been developed for KDV, there is no efficient algorithm for computing NKDV which is scalable to million-sized datasets, to the best of our knowledge.

As discussed in Section 2.2, existing studies [47, 72] adopt the two-step framework to evaluate NKDV, (1) shortest path distance computation and (2) $\mathcal{F}_P(\mathbf{q})$ computation (cf. Equation 2). In step 1, even though many advanced and efficient algorithms, e.g., hierarchical indexing [23], hub labeling [40] and shortest path caching [35, 68] can significantly improve the practical efficiency for computing the shortest path distance, step 1 is not the bottleneck for computing NKDV (smaller than 20% in the overall performance for the baseline solutions RQS and SPS (cf. Figure 17)), especially for the large-scale dataset P. As such, we still adopt the traditional Dijkstra's method [18] for obtaining the shortest path distance. In step 2, even though many research studies have focused on improving the efficiency for computing $\mathcal{F}_P(\mathbf{q})$ in planar KDV (cf. Figure 2a), e.g., [13, 16, 22, 82, 84], all these research studies do not consider the network distance in the kernel functions (cf. Table 1). Therefore, it is not trivial to directly extend these methods in this setting. To the best of our knowledge, this is the first work which can provide the theoretically efficient algorithms for computing NKDV, compared with the state-of-the-art methods RQS [47, 72] (cf. Section 2.3) and SPS [57] (cf. Section 2.4).

In spatial database community, many research studies also propose different analytic tasks in the road network, including clustering [75], nearest neighbor queries [50, 76], hotspot detection [65, 66], route planning [39, 44, 49], traffic analysis [19, 37, 46] and trajectory analysis [28, 29, 81]. However, most of these studies either do not need to consider the density of different positions of the road network or do not utilize the kernel density function for generating the density. As such, all of these studies cannot be easily extended to support the NKDV operation, which is used in geographical applications (e.g., [72]) extensively and supported by ArcGIS software [1] (via SANET [8]). Among most of the research studies in spatial database community, Romano et al. [59] develop the system for supporting spatial-temporal network kernel density visualization, which can be regarded as the generalization of NKDV. However, this work does not propose any efficient algorithm for evaluating the kernel aggregation function.

There are also many other types of visualization techniques [21, 26, 27, 33, 36, 41, 42, 51, 62, 70], which are not based on NKDV. However, geographical users [30, 38, 47, 48, 60, 72, 73] mainly utilize the NKDV to detect the hotspots in road networks. As such, these research studies cannot be directly applied for this scenario.

6 CONCLUSION

In this paper, we study network kernel density visualization (NKDV), which has been extensively used in many geographical applications. However, existing algorithms are not scalable to million-sized datasets. To achieve significant speedup over the state-of-the-art methods (RQS [47, 72] and SPS [57]) and existing software (e.g., SANET [8] plugin for ArcGIS [1]), we develop three efficient algorithms, namely aggregate distance augmentation (ADA), interval-based augmentation (IA) and hybrid augmentation (HA). Theoretically, we show that all our methods can provide lower time complexity, compared with the state-of-the-art solutions (cf. Table 3). In practice, ADA, IA and HA can achieve at least 5x to 10x speedup for computing NKDV.

In the future, we will extend our methods for handling NKDV with other commonly-used kernel functions. Moreover, like [14], we also plan to develop an interactive network kernel density visualization system for supporting some meaningful applications, e.g., visualizing the COVID-19 cases. Furthermore, we will extend this work to support spatial-temporal network kernel density visualization [59], which is the generalization of NKDV.

REFERENCES

[1] ArcGIS. http://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kernel-density-works.htm (last accessed: 2020-10-15).

[2] IKCEST: Disaster risk reduction. http://drr.ikcest.org/knowledge_service/ncp.html (last accessed: 2020-10-15).

[3] Johns Creek open data. https://opendata.atlantaregional.com/datasets/JohnsCreekGA::police-calls-for-service-archive-2009-to-2018 (last accessed: 2020-10-15).

[4] Los Angeles open data. https://data.lacity.org/A-Safe-City/Crime-Data-from-2010-to-2019/63jg-8b9z (last accessed: 2020-10-15).

[5] NYC open data. https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95 (last accessed: 2020-10-15).

[6] Openstreetmap. https://www.openstreetmap.org/ (last accessed: 2020-10-15).

[7] QGIS. https://docs.qgis.org/2.18/en/docs/user_manual/plugins/plugins_heatmap.html (last accessed: 2020-10-15).

[8] SANET. http://sanet.csis.u-tokyo.ac.jp/ (last accessed: 2020-10-15).

[9] Seattle open data. https://data.seattle.gov/Public-Safety/SPD-Crime-Data-2008-Present/tazs-3rd5 (last accessed: 2020-10-15).

[10] M. Bíl, R. Andrášik, and Z. Janoška. Identification of hazardous road locations of traffic accidents by means of kernel density estimation and cluster significance evaluation. *Accident Analysis & Prevention*, 55:265 – 273, 2013.

[11] G. Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126 – 139, 2017.

[12] S. Chainey, L. Tompson, and S. Uhlig. The utility of hotspot mapping for predicting spatial patterns of crime. *Security Journal*, 21(1):4–28, Feb 2008.

[13] T. N. Chan, R. Cheng, and M. L. Yiu. QUAD: Quadratic-bound-based kernel density visualization. In *SIGMOD*, pages 35–50, 2020.

[14] T. N. Chan, P. L. Ip, L. H. U, W. H. Tong, S. Mittal, Y. Li, and R. Cheng. KDV-Explorer: A near real-time kernel density visualization system for spatial analysis. *Proc. VLDB Endow.*, 2021, (To appear).

[15] T. N. Chan, L. H. U, R. Cheng, M. L. Yiu, and S. Mittal. Efficient algorithms for kernel aggregation queries. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[16] T. N. Chan, M. L. Yiu, and L. H. U. KARL: Fast kernel aggregation queries. In *ICDE*, pages 542–553, 2019.

[17] W. Chen, F. Guo, and F. Wang. A survey of traffic data visualization. *IEEE Trans. Intelligent Transportation Systems*, 16(6):2970–2984, 2015.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[19] D. Deng, C. Shahabi, U. Demiryurek, L. Zhu, R. Yu, and Y. Liu. Latent space model for road networks to predict time-varying traffic. In *SIGKDD*, pages 1525–1534, 2016.

[20] 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.

[21] A. Eldawy, M. F. Mokbel, and C. Jonathan. HadoopViz: A mapreduce framework for extensible visualization of big spatial data. In *ICDE*, pages 601–612, 2016.

[22] E. Gan and P. Bailis. Scalable kernel density classification via threshold-based pruning. In *ACM SIGMOD*, pages 945–959, 2017.

[23] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[24] A. Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Studies in Big Data. Springer International Publishing, 2017.

[25] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[26] T. Guo, K. Feng, G. Cong, and Z. Bao. Efficient selection of geospatial data on maps for interactive and visualized exploration. In *SIGMOD*, pages 567–582, 2018.

[27] T. Guo, M. Li, P. Li, Z. Bao, and G. Cong. POIsam: a system for efficient selection of large-scale geospatial data on maps. In *SIGMOD*, pages 1677–1680, 2018.

[28] B. Han, L. Liu, and E. Omiecinski. Road-network aware trajectory clustering: Integrating locality, flow, and density. *IEEE Trans. Mob. Comput.*, 14(2):416–429, 2015.

[29] B. Han, L. Liu, and E. Omiecinski. A systematic approach to clustering whole trajectories of mobile objects in road networks. *IEEE Trans. Knowl. Data Eng.*, 29(5):936–949, 2017.

[30] H. Harirforoush and L. Bellalite. A new integrated gis-based analysis to detect hotspots: A case study of the city of sherbrooke. *Accident Analysis & Prevention*, 130:62 – 74, 2019. Road Safety Data Considerations.

[31] T. Hart and P. Zandbergen. Kernel density estimation and hotspot mapping: examining the influence of interpolation method, grid cell size, and bandwidth on crime forecasting. *Policing: An International Journal of Police Strategies and Management*, 37:305–323, 2014.

[32] S. C. Joshi, R. V. Kommaraju, J. M. Phillips, and S. Venkatasubramanian. Comparing distributions and shapes using the kernel distance. In *SOCG*, pages 47–56, 2011.

[33] P. K. Kefaloukos, M. A. V. Salles, and M. Zachariasen. Declarative cartography: In-database map generalization of geospatial datasets. In *ICDE*, pages 1024–1035, 2014.

[34] S. Khalid, F. Shoaib, T. Qian, Y. Rui, A. 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, 9 2018.

[35] L. Li, M. Zhang, W. Hua, and X. Zhou. Fast query decomposition for batch shortest path processing in road networks. In *ICDE*, pages 1189–1200, 2020.

[36] M. Li, Z. Bao, F. M. Choudhury, and T. Sellis. Supporting large-scale geographical visualization in a multi-granularity way. In *WSDM*, pages 767–770, 2018.

[37] P. H. Li, M. L. Yiu, and K. Mouratidis. Discovering historic traffic-tolerant paths in road networks. *GeoInformatica*, 21(1):1–32, 2017.

[38] Q. Li, T. Zhang, H. Wang, and Z. Zeng. Dynamic accessibility mapping using floating car data: a network-constrained density estimation approach. *Journal of Transport Geography*, 19(3):379 – 393, 2011. Special Issue : Geographic Information Systems for Transportation.

[39] Y. Li, H. Su, U. Demiryurek, B. Zheng, T. He, and C. Shahabi. PaRE: A system for personalized route guidance. In *WWW*, pages 637–646, 2017.

[40] Y. Li, L. H. U, M. L. Yiu, and N. M. Kou. An experimental study on hub labeling based shortest path algorithms. *Proc. VLDB Endow.*, 11(4):445–457, 2017.

[41] A. Mayorga and M. Gleicher. Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1526–1538, Sept 2013.

[42] L. Micallef, G. Palmas, A. Oulasvirta, and T. Weinkauf. Towards perceptual optimization of the visual design of scatterplots. *IEEE Trans. Vis. Comput. Graph.*, 23(6):1588–1599, 2017.

[43] M. M. Moradi, F. J. Rodríguez-Cortés, and J. Mateu. On kernel-based intensity estimation of spatial point patterns on linear networks. *Journal of Computational and Graphical Statistics*, 27(2):302–311, 2018.

[44] K. Mouratidis, Y. Lin, and M. L. Yiu. Preference queries in large multi-cost transportation networks. In *ICDE*, pages 533–544, 2010.

[45] J. Ni, T. Qian, C. Xi, Y. Rui, and J. Wang. Spatial distribution characteristics of healthcare facilities in nanjing: Network point pattern analysis and correlation analysis. *International journal of environmental research and public health*, 13(8):833, 2016.

[46] J. Ni and C. V. Ravishankar. Pointwise-dense region queries in spatio-temporal databases. In *ICDE*, pages 1066–1075, 2007.

[47] A. Okabe, T. Satoh, and K. Sugihara. A kernel density estimation method for networks, its computational method and a gis-based tool. *International Journal of Geographical Information Science*, 23(1):7–32, 2009.

[48] A. Okabe and K. Sugihara. *Spatial Analysis Along Networks: Statistical and Computational Methods*. Statistics in Practice. Wiley, 2012.

[49] D. Oliver, S. Shekhar, J. M. Kang, R. Laubscher, V. Carlan, and A. Bannur. A k-main routes approach to spatial network activity summarization. *IEEE Trans. Knowl. Data Eng.*, 26(6):1464–1478, 2014.

[50] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *VLDB*, pages 802–813, 2003.

[51] Y. Park, M. J. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, pages 755–766, 2016.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, and D. Auber. Large interactive visualization of density functions on big data infrastructure. In *LDAV*, pages 99–106, 2015.

[54] J. M. Phillips. ϵ-samples for kernels. In *SODA*, pages 1622–1632, 2013.

[55] J. M. Phillips and W. M. Tai. Improved coresets for kernel density estimates. In *SODA*, pages 2718–2727, 2018.

[56] J. M. Phillips and W. M. Tai. Near-optimal coresets of kernel density estimates. In *SOCG*, pages 66:1–66:13, 2018.

[57] S. Rakshit, A. Baddeley, and G. Nair. Efficient code for second order analysis of events on a linear network. *Journal of Statistical Software, Articles*, 90(1):1–37, 2019.

[58] V. C. Raykar, R. Duraiswami, and L. H. Zhao. Fast computation of kernel estimators. *Journal of Computational and Graphical Statistics*, 19(1):205–220, 2010.

[59] B. Romano and Z. Jiang. Visualizing traffic accident hotspots based on spatial-temporal network kernel density estimation. In *SIGSPATIAL*, pages 98:1–98:4, 2017.

[60] G. Rosser, T. O. Davies, K. Bowers, S. D. Johnson, and T. Cheng. Predictive crime mapping: Arbitrary grids or street networks? *Journal of Quantitative Criminology*, 33:569 – 594, 2017.

[61] Y. Rui, Z. Yang, T. Qian, S. Khalid, N. Xia, and J. Wang. Network-constrained and category-based point pattern analysis for suguo retail stores in nanjing, china. *International Journal of Geographical Information Science*, 30(2):186–199, 2016.

[62] A. D. Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Y. Halevy. Efficient spatial sampling of large geographical tables. In *SIGMOD*, pages 193–204, 2012.

[63] D. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* A Wiley-interscience publication. Wiley, 1992.

[64] J. Steele. *The Cauchy-Schwarz Master Class: An Introduction to the Art of Mathematical Inequalities.* MAA problem books series. Cambridge University Press, 2004.

[65] X. Tang, E. Eftelioglu, and S. Shekhar. Detecting isodistance hotspots on spatial networks: A summary of results. In *SSTD*, pages 281–299, 2017.

[66] X. Tang, J. Gupta, and S. Shekhar. Linear hotspot discovery on all simple paths: A summary of results. In *SIGSPATIAL*, pages 476–479, 2019.

[67] A. C. Telea. *Data Visualization: Principles and Practice, Second Edition.* A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014.

[68] J. R. Thomsen, M. L. Yiu, and C. S. Jensen. Effective caching of shortest paths for location-based services. In *SIGMOD*, pages 313–324, 2012.

[69] M. Wand and M. Jones. *Kernel Smoothing.* Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[70] D. Wilkie, J. Sewall, and M. C. Lin. Transforming GIS data into functional road models for large-scale traffic simulation. *IEEE Trans. Vis. Comput. Graph.*, 18(6):890–901, 2012.

[71] K. Xie, K. Ozbay, A. Kurkcu, and H. Yang. Analysis of traffic crashes involving pedestrians using big data: Investigation of contributing factors and identification of hotspots. *Risk Analysis*, 37(8):1459–1476, 2017.

[72] Z. Xie and J. Yan. Kernel density estimation of traffic accidents in a network space. *Computers, Environment and Urban Systems*, 32(5):396 – 406, 2008.

[73] Z. Xie and J. Yan. Detecting traffic accident clusters with network kernel density estimation and local spatial statistics: an integrated approach. *Journal of Transport Geography*, 31:64 – 71, 2013.

[74] C. Yang, R. Duraiswami, and L. S. Davis. Efficient kernel machines using the improved fast gauss transform. In *NIPS*, pages 1561–1568, 2004.

[75] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, pages 443–454, 2004.

[76] M. L. Yiu, N. Mamoulis, and D. Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Trans. Knowl. Data Eng.*, 17(6):820–833, 2005.

[77] H. Yu, P. Liu, J. Chen, and H. Wang. Comparative analysis of the spatial analysis methods for hotspot identification. *Accident Analysis and Prevention*, 66:80 – 88, 2014.

[78] W. Yu, T. Ai, and S. Shao. The analysis and delimitation of central business district using network kernel density estimation. *Journal of Transport Geography*, 45:32–47, 2015.

[79] G. Zhang, A. Zhu, and Q. Huang. A GPU-accelerated adaptive kernel density estimation approach for efficient point pattern analysis on spatial big data. *International Journal of Geographical Information Science*, 31(10):2068–2097, 2017.

[80] Z. Zhang, D. Chen, W. Liu, J. Racine, S.-H. Ong, Y. Chen, G. Zhao, and Q. Jiang. Nonparametric evaluation of dynamic disease risk: A spatio-temporal kernel approach. *PloS one*, 6:e17381, 03 2011.

[81] Y. Zheng. Trajectory data mining: An overview. *ACM Trans. Intell. Syst. Technol.*, 6(3), May 2015.

[82] Y. Zheng, J. Jestes, J. M. Phillips, and F. Li. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*, pages 433–444, 2013.

[83] Y. Zheng, Y. Ou, A. Lex, and J. M. Phillips. Visualization of big spatial data using coresets for kernel density estimates. In *IEEE Symposium on Visualization in Data Science (VDS '17), to appear*. IEEE, 2017.

[84] Y. Zheng and J. M. Phillips. L∞ error and bandwidth selection for kernel density estimates of large data. In *SIGKDD*, pages 1533–1542, 2015.