Fast Network K-function-based Spatial Analysis

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

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

Yun Peng
Institute of Artificial Intelligence and Blockchain, Guangzhou University
yunpeng@gzhu.edu.cn

Byron Choi
Hong Kong Baptist University
bchoi@comp.hkbu.edu.hk

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

ABSTRACT

Network K-function has been the de facto operation for analyzing point patterns in spatial networks, which is widely used in many communities, including geography, ecology, transportation science, social science, and criminology. To analyze a location dataset, domain experts need to generate a network K-function plot that involves computing multiple network K-functions. However, network K-function is a computationally expensive operation that is not feasible to support large-scale datasets, let alone to generate a network K-function plot. To handle this issue, we develop two efficient algorithms, namely count augmentation (CA) and neighbor sharing (NS), which can reduce the worst-case time complexity for computing network K-functions. In addition, we incorporate the advanced shortest path sharing (ASPS) approach into these two methods to further lower the worst-case time complexity for generating network K-function plots. Experiment results on four large-scale location datasets (up to 7.33 million data points) show that our methods can achieve up to 165.85x speedup compared with the state-of-the-art methods.

1 INTRODUCTION

Point pattern analysis in spatial networks [39] is a fundamental topic in different communities, including geography, ecology, transportation science, social science, and criminology. One of the most important operations in point pattern analysis is network K-function [30, 39, 40], which counts all data points that are within a given distance τ from every data point. Figure 1 shows an example to illustrate the concept of network K-function. In Figure 1a, since there is no data point within the distance τ from the data points $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, and $\mathbf{p}_4$, the network K-function value for the dataset of Figure 1a is 0. In Figure 1b, since the sets $\{\mathbf{p}_2, \mathbf{p}_3\}$, $\{\mathbf{p}_1, \mathbf{p}_3\}$, and $\{\mathbf{p}_1, \mathbf{p}_2\}$ are within the distance τ from $\mathbf{p}_1$, $\mathbf{p}_2$, and $\mathbf{p}_3$, respectively, and there is no data point within the distance τ from $\mathbf{p}_4$, the network K-function value for this dataset is 6. In general, a high network K-function value indicates that the data points are clustered.

Network K-function has been extensively adopted by domain experts to analyze location datasets and perform meta-analysis. Consider the most representative example. Transportation experts [10, 14, 22, 30, 35, 57, 58] and criminologists [11, 12, 20, 34, 47] first compute the network K-function values for a given location dataset and L randomly generated datasets (e.g., $L = 100$ random datasets are generated in [30]) with D distance values, i.e., τ_1, τ_2,..., τ_D (D is 200 in [30]). Then, they can generate the network K-function plot (cf. Figure 2) based on these values. In Figure 2, the black curve shows the network K-function values for the location dataset with τ_1, τ_2,..., τ_D. In addition, the red dashed curve and the blue dashed curve denote the smallest and the largest network K-function values, respectively, among those L random datasets for each distance τ_d. Observe that once the black curve is above the blue dashed curve (i.e., orange region), they classify that these data points are clustered with the corresponding distance values τ. In contrast, they reckon that the data points are dispersed with those distance values τ if the black curve is below the red dashed curve (i.e., yellow region).

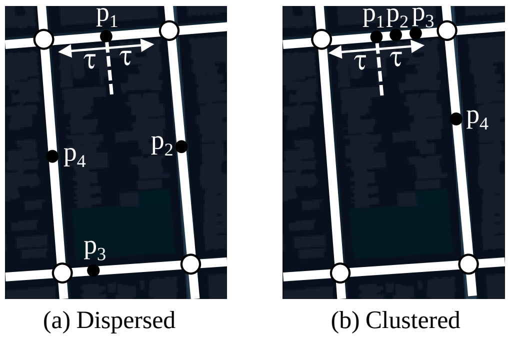


Figure 1: Network K-function with distance τ for two location datasets.

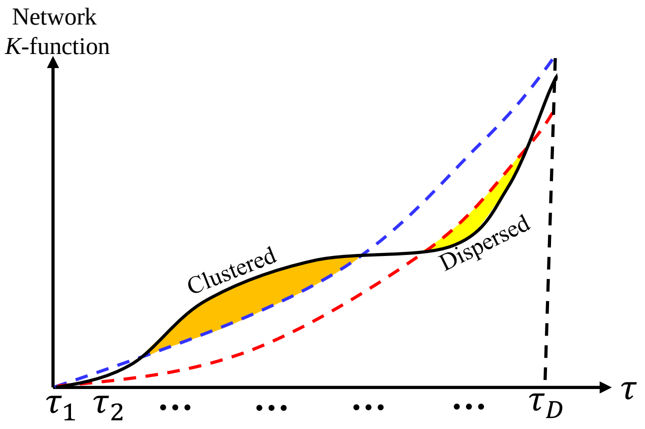


Figure 2: Network K-function plot.

Table 1: Theoretical results of different exact methods for generating the network K-function plot, where T_{SP} and $\mathcal{S}_{\text{SP}}$ denote the time and space complexity of the shortest path algorithm, respectively.

Method	Time complexity	Space complexity
RQS [7, 37–39] (cf. Section 2.2)	$O(LDn(T_{\text{SP}} + n))$	$O(\|V\| + \|E\| + nL + \mathcal{S}_{\text{SP}})$
SPS [46] (cf. Section 2.3)	$O(LD(\|E\|T_{\text{SP}} + n^2))$	
CA (cf. Section 3.2)	$O\big(LD\big(\|E\|T_{\text{SP}} + n\|E\| \log\big(\frac{n}{\|E\|}\big)\big) + Ln \log n\big)$ (cf. Theorem 1)	$O(\|V\| + \|E\| + nL + \mathcal{S}_{\text{SP}})$ (cf. Theorem 4)
NS (cf. Section 3.3)	$O(LD(\|E\|T_{\text{SP}} + n\|E\|) + Ln \log n)$ (cf. Theorem 2)	
CA$^{(\text{ASPS})}$ (cf. Sections 3.2 and 3.4)	$O\big(\|E\|T_{\text{SP}} + nLD\|E\| \log\big(\frac{n}{\|E\|}\big) + Ln \log n\big)$ (cf. Theorem 3)	
NS$^{(\text{ASPS})}$ (cf. Sections 3.3 and 3.4)	$O(\|E\|T_{\text{SP}} + nLD\|E\| + Ln \log n)$ (cf. Theorem 3)	

After domain experts obtain this network K-function plot, they can detect which distance τ reveals the meaningful clusters/hotspots for the location dataset (e.g., τ in the orange region of Figure 2), which can further facilitate their spatial analysis tasks (e.g., understanding which clusters/hotspots in a location dataset are the meaningful ones but not the random distributions [49, 55]). In addition, social scientists and urban planners [23, 36, 41, 49, 55, 56] utilize the network K-function to analyze the spatial patterns of different social phenomena, e.g., distribution properties of health care facilities in a city [36, 56]. Ecologists [50] utilize the network K-function to analyze the spatial patterns of different species. All of these studies also need to generate the network K-function plot (cf. Figure 2) for their tasks. Due to the wide applicability of network K-function, the famous software ArcGIS [1] (based on the SANET plugin [7, 37, 38]) and spNetwork [8] (an R package) can also support this operation.

Although network K-function is widely used in many applications, these studies normally restrict the usage of this tool in small-scale datasets (e.g., thousand-scale data points). The main reason is that network K-function is a time-consuming operation, which involves the evaluation of distances between all pairs of data points in the worst case. Consider a road network that has 10,000 data points, the network K-function could take up to 100 million operations ($10,000 \times 10,000$). However, with the ever growing data size in different applications, e.g., 7.3 million location data points in the Chicago crime dataset [2], the existing solutions [7, 37–39, 46] for network K-function are infeasible to handle these large-scale datasets. Worse still, all these studies need to compute $L+1$ network K-function values for each distance value τ in order to generate the network K-function plot, which further deteriorates this infeasibility issue to use this tool for handling large-scale datasets. As such, some research studies [34, 46] have explicitly pointed out this drawback for using the network K-function. For example: Lu et al. [34] explicitly state that they adopt small datasets (with 1,452 samples) in order to avoid the excessive computation when they use the network K-function tool.

Therefore, we ask a research question. *Can we reduce the time complexity for generating the exact network K-function plot, without increasing the space complexity?* To provide an affirmative answer to this question, we first develop the simple count augmentation (CA) method (by adapting the solution in [17]) and the advanced neighbor sharing (NS) method, which can significantly reduce the time complexity for generating the network K-function plot. Then, we incorporate the advanced shortest path sharing (ASPS) approach for these two methods, namely CA$^{(\text{ASPS})}$ and NS$^{(\text{ASPS})}$, which can further reduce the time complexity for generating the network K-function plot. As a remark, all these methods do not theoretically incur space overhead (i.e., with the same space complexity). Table 1 summarizes the time and space complexity of all methods. Experiment results show that our methods can achieve up to 165.85x speedup compared with the existing methods. Furthermore, we demonstrate how to use the network K-function plot to understand the cluster properties for the crime location dataset in Chicago, which cannot be feasibly supported by the existing solutions.

The rest of the paper is organized as follows. First, we define the problem of generating the network K-function plot and discuss the existing methods, RQS and SPS, for solving this problem in Section 2. Then, we present our methods in Section 3. Next, we show our experiment results in Section 4. After that, we discuss the related work in Section 5. Lastly, we conclude our paper in Section 6. The appendix of this paper can be found in Section 7.

2 PRELIMINARIES

In this section, we first formally define the network K-function and the problem for generating the network K-function plot in Section 2.1. Then, we discuss two baseline methods for generating this network K-function plot, namely range-query-based solution (RQS) and shortest path sharing solution (SPS), in Section 2.2 and Section 2.3, respectively.

2.1 Problem Definition

Recall from Section 1, the network K-function is to count all data points that are within a given distance τ from each data point in the location dataset (cf. Definition 1).

Definition 1. *Given the spatial network $G = (V, E)$, the location dataset P with n data points, i.e., $P = \{p_1, p_2, ..., p_n\}$, where each data point is on one and only one edge $e \in E$, and the distance τ, the network K-function for this dataset P is $K_P(\tau)$ (cf. Equation 1).*

$$K_P(\tau) = \sum_{p_i \in P} \sum_{\substack{p_j \in P \\ p_j \neq p_i}} \mathbb{I}(dist_G(p_i, p_j) \leq \tau) \tag{1}$$

where $\mathbb{I}(dist_G(p_i, p_j) \leq \tau)$ is the indicator function (cf. Equation 2) and $dist_G(p_i, p_j)$ is the shortest path distance between p_i and p_j.

$$\mathbb{I}(dist_G(p_i, p_j) \leq \tau) = \begin{cases} 1 & \text{if } dist_G(p_i, p_j) \leq \tau \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

With the definition of the network K-function, we can formally define the problem for generating the network K-function plot (cf. Figure 2) in Problem 1.

Problem 1. *Given the spatial network $G = (V, E)$, the distance values, τ_1, τ_2,..., τ_D, the location dataset P and L randomly generated datasets R_1, R_2,..., R_L with the same size n, we need to plot these* *three values $\mathcal{L}(\tau_d)$, $\mathcal{U}(\tau_d)$ and $K_P(\tau_d)$ for each distance τ_d, where* $1 \le d \le D$.

$$\mathcal{L}(\tau_d) = \min(K_{R_1}(\tau_d), K_{R_2}(\tau_d), \cdots, K_{R_L}(\tau_d)) \quad (3)$$
$$\mathcal{U}(\tau_d) = \max(K_{R_1}(\tau_d), K_{R_2}(\tau_d), \cdots, K_{R_L}(\tau_d)) \quad (4)$$

Observe that $\mathcal{L}(\tau_d)$ and $\mathcal{U}(\tau_d)$ are the smallest and largest network K-function values (with distance τ_d) of those L randomly generated datasets R_1, R_2,..., R_L.

2.2 Range-Query-based Solution (RQS)

To efficiently compute the network K-function $K_P(\tau)$ (cf. Equation 1), it is sufficient to count, for each p_i, those data points p_j that are within the distance τ, i.e., $dist_G(p_i, p_j) \le \tau$. Therefore, instead of scanning all data points in G for each p_i, Okabe et al. [39] first find the range query set for p_i, i.e., $RQ_\tau(p_i)$ (cf. Equation 5), where

$$RQ_\tau(p_i) = \{p_j \in P : dist_G(p_i, p_j) \le \tau, p_j \ne p_i\} \quad (5)$$

Then, they can compute the network K-function (cf. Equation 6), based on the range query set for each p_i.

$$K_P(\tau) = \sum_{p_i \in P} |RQ_\tau(p_i)| \quad (6)$$

In this paper, we term this method as range-query-based solution (RQS). Figure 3 shows an example for this method to compute $|RQ_\tau(p_i)|$. First, this RQS method utilizes the shortest path algorithm SP_τ (e.g., Dijkstra's algorithm [19]) to find the shortest path distances from p_i to those yellow nodes (cf. Figure 3) that are within the distance τ. Here, we denote $SPD(p_i).u$ to be the shortest path distance from p_i to each node $u \in V$ that is obtained by SP_τ (cf. Equation 7).

$$SPD(p_i).u = \begin{cases} dist_G(p_i, u) & \text{if } dist_G(p_i, u) \le \tau \\ \infty & \text{otherwise} \end{cases} \quad (7)$$

Then, this method scans the data points of each edge that is connected to the yellow node (i.e., the node u with $SPD(p_i).u \le \tau$) and determines whether these data points are within the distance τ from p_i. Algorithm 1 shows the pseudocode of this RQS method.

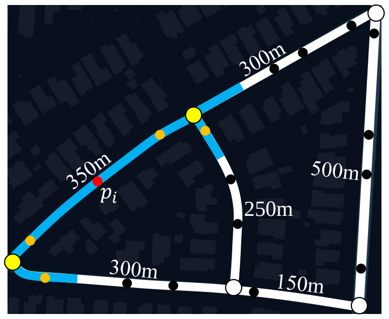


Figure 3: Find the range query set $RQ_\tau(p_i)$ for the data point p_i with $\tau = 250$m (We use the blue color to denote this range from p_i). Here, we have $|RQ_\tau(p_i)| = 4$ (orange data points).

Since this method needs to adopt the shortest path algorithm and scan all data points in the road network for each p_i in the worst case (with large distance τ), the time complexity for computing $K_P(\tau)$ is $O(n(T_{SP} + n))$, where T_{SP} denotes the time complexity of the

Algorithm 1 Range-Query-based Solution for Network K-function

```
1:  procedure RQS($G = (V, E)$, $P = \{p_1, p_2, ..., p_n\}$, distance $\tau$)
2:      $K \leftarrow 0$
3:      for each $p_i \in P$ do
4:          $SPD(p_i) \leftarrow SP_\tau(G, p_i)$
5:          for each $e = (u, v) \in E$ do
6:              if $SPD(p_i).u \le \tau$ or $SPD(p_i).v \le \tau$ then
7:                  for each $p_j$ in the edge $e$ do
8:                      $\Delta \leftarrow \min \begin{cases} SPD(p_i).u + dist_G(p_j, u) \\ SPD(p_i).v + dist_G(p_j, v) \end{cases}$
9:                      $K \leftarrow K + \mathbb{I}(\Delta \le \tau)$
10:         Clear $SPD(p_i)$                                  ▷ Clear memory
11:     Return $K$
```

shortest path algorithm, e.g., $T_{SP} = |V| \log |V| + |E|$ for Dijkstra's algorithm [19]. Recall that we need to compute $(L + 1) \times D$ network K-function values in order to generate the network K-function plot (cf. Problem 1). Therefore, the worst-case time complexity for using RQS to solve Problem 1 is $O(LDn(T_{SP} + n))$, which can be very time-consuming. As a remark, since this method needs to access the road network (with $O(|V| + |E|)$ space), access $L + 1$ datasets (with $O(nL)$ space), and adopt the shortest path algorithm (with $O(\mathcal{S}_{SP})$ space), the space complexity of RQS is $O(|V| + |E| + nL + \mathcal{S}_{SP})$.

2.3 Shortest Path Sharing Solution (SPS)

Recently, Rakshit et al. [46] utilize the shortest path sharing solution (SPS) to improve the efficiency for computing the network K-function. Figure 4 shows the core idea of this method. Consider the data points p_i and p_j in the edges (a, b) and (u, v), respectively. We can observe that there are at most four paths from p_i to p_j, which are (1) $p_i \to a \to u \to p_j$, (2) $p_i \to a \to v \to p_j$, (3) $p_i \to b \to u \to p_j$, and (4) $p_i \to b \to v \to p_j$. As such, we can use Equation 8 to denote the shortest path distance between p_i and p_j:

$$dist_G(p_i, p_j) = \min \begin{cases} dist_G(p_i, a) + dist_G(a, u) + dist_G(u, p_j) \\ dist_G(p_i, a) + dist_G(a, v) + dist_G(v, p_j) \\ dist_G(p_i, b) + dist_G(b, u) + dist_G(u, p_j) \\ dist_G(p_i, b) + dist_G(b, v) + dist_G(v, p_j) \end{cases} \quad (8)$$

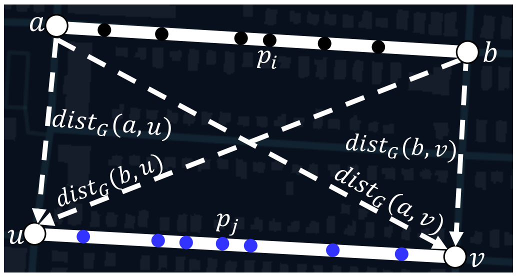


Figure 4: The core idea of the SPS method. Black and blue dots denote the data points in the edges (a, b) and (u, v), respectively.

Therefore, instead of calling the shortest path algorithm for each data point p_i (like RQS (cf. Section 2.2)), this SPS method first computes the shortest path distances from the nodes a and b to other nodes (e.g., u and v) in the spatial network G (e.g., $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$), i.e., $SPD(a)$ and $SPD(b)$ (replace p_i by a and b in Equation 7, respectively). Then, for each data point (black dot) p_i, this method computes the shortest path distance between p_i and p_j using Equation 8. Since $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$ are computed in advance, this method only calls the shortest path algorithm two times for each edge (e.g., (a,b)) and share these shortest path distance values to other data points (e.g., all black dots). As such, the time complexity of the SPS method is reduced to $O(LD(|E|T_{\text{SP}}+n^2))$ compared with the RQS method. As a remark, once the SPS method has processed all data points in the edge $\widetilde{e}=(a,b)$, it clears all the shortest path distances $SPD(a)$ and $SPD(b)$. Therefore, the space complexity of the SPS method remains the same as the RQS method. Algorithm 2 shows the pseudocode of this SPS method.

Algorithm 2 Shortest Path Sharing Solution for Network K-function

```
1:  procedure SPS($G=(V,E)$, $P=\{p_1,p_2,...,p_n\}$, distance $\tau$)
2:      $K \leftarrow 0$
3:      for each edge $\widetilde{e}=(a,b)\in E$ do
4:          $SPD(a) \leftarrow SP_\tau(G,a)$
5:          $SPD(b) \leftarrow SP_\tau(G,b)$
6:          for each $p_i$ in the edge $\widetilde{e}$ do
7:              for each $e=(u,v)\in E$ do
8:                  if $\left(\begin{array}{l} SPD(a).u\le\tau \text{ or } SPD(a).v\le\tau \\ \text{or } SPD(b).u\le\tau \text{ or } SPD(b).v\le\tau \end{array}\right)$ then
9:                      for each $p_j$ in the edge $e$ do
10:                         $\Delta \leftarrow dist_G(p_i,p_j)$                ▷ Equation 8
11:                         $K \leftarrow K+\mathbb{I}(\Delta\le\tau)$
12:         Clear $SPD(a)$, $SPD(b)$                              ▷ Clear memory
13:     Return $K$
```

3 OUR METHODS

Although the RQS and SPS methods can improve the efficiency for generating the network K-function plot, the time complexity of these two methods can still be very high, especially for large-scale datasets (i.e., large n), large number of distance values (i.e., large D) and large number of datasets (i.e., large L). To tackle this inefficiency issue, we first discuss how to decompose the network K-function (cf. Equation 1) in Section 3.1. Based on this idea, we then propose two complexity-optimized methods, which are (1) count augmentation (CA) and (2) neighbor sharing (NS) in Section 3.2 and Section 3.3, respectively. After that, we further develop the advanced shortest path sharing approach (ASPS) and integrate it with the CA and NS methods in Section 3.4. Lastly, we discuss the space complexity of our methods in Section 3.5.

3.1 Decomposition of Network K-function

We first define the set of data points for each edge e in the spatial network G as $P(e)$ (cf. Definition 2).[1]

[1] All our methods need to sort the data points p_j in $P(e)$ based on the distance values from one of the nodes in each edge e to p_j (e.g., $dist_G(u,p_j)$ or $dist_G(v,p_j)$ if $e=(u,v)$) in advance. The time complexity for sorting the data points in all edges is $O(n\log n+|E|)$ for each dataset. For simplicity, we have sorted $P(e)$ in advance for each edge $e\in E$ in this section.

Definition 2. *Given an edge e in the spatial network $G=(V,E)$, $P(e)$ is the point set in this edge e.*

Based on this definition, we can then express the network K-function $K_P(\tau)$ (cf. Equation 1) as follows.

$$\begin{aligned} K_P(\tau) &= \sum_{p_i\in P}\sum_{\substack{p_j\in P\\ p_j\neq p_i}}\mathbb{I}(dist_G(p_i,p_j)\le\tau)\\ &= \sum_{\widetilde{e}\in E}\sum_{p_i\in P(\widetilde{e})}\sum_{e\in E}\sum_{\substack{p_j\in P(e)\\ p_j\neq p_i}}\mathbb{I}(dist_G(p_i,p_j)\le\tau)\\ &= \sum_{\widetilde{e}\in E}\sum_{e\in E}C_P^{(\widetilde{e},e)}(\tau) \end{aligned} \tag{9}$$

where we denote $C_P^{(\widetilde{e},e)}(\tau)$ as the $(\widetilde{e},e)$-count function (cf. Equation 10), which counts the number of data points p_j in the edge e that are within the distance τ from each data point p_i in the edge $\widetilde{e}$.

$$C_P^{(\widetilde{e},e)}(\tau)=\sum_{p_i\in P(\widetilde{e})}\sum_{\substack{p_j\in P(e)\\ p_j\neq p_i}}\mathbb{I}(dist_G(p_i,p_j)\le\tau) \tag{10}$$

Therefore, if we can efficiently compute the $(\widetilde{e},e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$, we can improve the efficiency for computing the network K-function $K_P(\tau)$.

Here, we also define the (p_i,e)-count function (cf. Equation 11), i.e., the inner summation term of $C_P^{(\widetilde{e},e)}(\tau)$, which will be also used in our methods. In addition, we will use $\widetilde{e}=(a,b)$ and $e=(u,v)$ as an example (cf. Figure 4) to illustrate our methods in this paper.

$$C_P^{(p_i,e)}(\tau)=\sum_{\substack{p_j\in P(e)\\ p_j\neq p_i}}\mathbb{I}(dist_G(p_i,p_j)\le\tau) \tag{11}$$

3.2 Count Augmentation (CA) Method

Recently, Chan et al. [17] propose the augmentation-based approach to reduce the time complexity for solving another important point pattern analysis operation, called network kernel density visualization, which is closely related to this work. Therefore, a natural question is whether we can extend their idea to efficiently generate the network K-function plot (cf. Problem 1). In this section, we adapt their idea (aggregate distance augmentation) and develop the method, called count augmentation (CA), which can also reduce the time complexity for generating the network K-function plot.

The core idea of the CA method is to first augment the count values $|P(p_j,u)|$ and $|P(p_j,v)|$ (instead of the aggregate distance values in [17]) for each data point p_j in the edge $e=(u,v)$ (cf. Figure 5), where

$$P(p_j,u) = \{p\in P(e): dist_G(u,p)\le dist_G(u,p_j)\} \tag{12}$$
$$P(p_j,v) = \{p\in P(e): dist_G(v,p)\le dist_G(v,p_j)\} \tag{13}$$

Based on these count values, $|P(p_j,u)|$ and $|P(p_j,v)|$, and the sorted order of data points in $P(e)$ (cf. footnote 1), we can utilize the binary search approach to avoid scanning the set of data points $P(e)$ for evaluating the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ (cf. Equation 11) once we have $dist_G(p_i,u)$ and $dist_G(p_i,v)$ (i.e., the green dashed line and the orange dashed line, respectively, in Figure 5). In Lemma 1, we claim that we can compute the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ in $O(\log|P(e)|)$ time, given the shortest path distances from node a

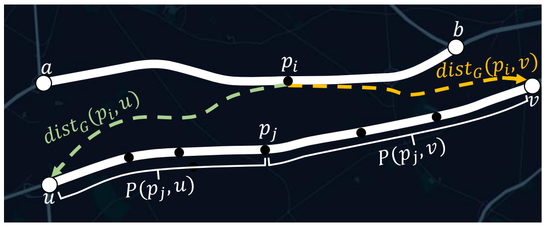


Figure 5: The core idea of the count augmentation (CA) method, $P(p_j,u)$ and $P(p_j,v)$ denote the sets of data points from u to p_j and v to p_j, respectively, in the edge $e=(u,v)$.

and node b (i.e., $SPD(a)$ and $SPD(b)$). The proof of this lemma can be found in Section 7.1.

Lemma 1. *Given the spatial network $G=(V,E)$, the count values $|P(p_j,u)|$ and $|P(p_j,v)|$ for each data point p_j in the edge $e=(u,v)$ and the shortest path distances from nodes a and b (in the edge $\widetilde{e}$) to nodes u and v, i.e., $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$, the time complexity for evaluating the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ (cf. Equation 11) is $O(\log|P(e)|)$ time.*

Based on Lemma 1, we further conclude in Lemma 2 that we can use $O(|P(\widetilde{e})|\log|P(e)|)$ time to compute the $(\widetilde{e},e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$ (cf. Equation 10). The proof of this lemma can be found in Section 7.2.

Lemma 2. *Given two edges $\widetilde{e}=(a,b)$ and $e=(u,v)$ in the spatial network $G=(V,E)$ and the shortest path distances $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$, the time complexity for computing the $(\widetilde{e},e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$ is $O(|P(\widetilde{e})|\log|P(e)|)$.*

Algorithm 3 Count Augmentation (CA) Method for Network K-function

```
 1: procedure CA($G=(V,E)$, $P=\{p_1,p_2,...,p_n\}$, distance $\tau$)
 2:     //Augment the count values for each point $p_j$
 3:     for each edge $e=(u,v)\in E$ do
 4:         for each $p_j\in P(e)$ do
 5:             Augment $|P(p_j,u)|$ in $p_j$            ▷ Equation 12
 6:             Augment $|P(p_j,v)|$ in $p_j$            ▷ Equation 13
 7:     $K\leftarrow 0$
 8:     for each edge $\widetilde{e}=(a,b)\in E$ do
 9:         $SPD(a)\leftarrow SP_\tau(G,a)$
10:         $SPD(b)\leftarrow SP_\tau(G,b)$
11:         for each edge $e=(u,v)\in E$ do
12:             Compute $C_P^{(\widetilde{e},e)}(\tau)$            ▷ Lemma 2
13:             $K\leftarrow K+C_P^{(\widetilde{e},e)}(\tau)$
14:         Clear $SPD(a)$, $SPD(b)$
15:     Return $K$
```

Algorithm 3 shows the pseudocode of our CA method for computing the network K-function, based on the fast evaluation of $C_P^{(\widetilde{e},e)}(\tau)$ (cf. Lemma 2). In Lemma 3, we conclude that the CA method takes $O\big(|E|T_{\text{SP}}+n|E|\log\big(\frac{n}{|E|}\big)+n\log n\big)$ to compute the network K-function. The proof of this lemma is in Section 7.3.

Lemma 3. *The time complexity of the CA method (cf. Algorithm 3) is $O\big(|E|T_{SP}+n|E|\log\big(\frac{n}{|E|}\big)+n\log n\big)$ for computing the network K-function.*

Recall that we need to compute the $D(L+1)$ network K-function values in order to generate the network K-function plot (cf. Problem 1). Based on Lemma 3 and the footnote 1, the time complexity for using the CA method is $O\big(LD\big(|E|T_{\text{SP}}+n|E|\log\big(\frac{n}{|E|}\big)\big)+Ln\log n\big)$ to solve Problem 1 (cf. Theorem 1).

Theorem 1. *The time complexity for using the CA method to generate the network K-function plot is $O\big(LD\big(|E|T_{SP}+n|E|\log\big(\frac{n}{|E|}\big)\big)+Ln\log n\big)$.*

Compared with the state-of-the-art method, SPS, which takes $O(LD(|E|T_{\text{SP}}+n^2))$ time, our CA method reduces the worst-case time complexity for generating the network K-function plot based on the following reason.

$$O\Big(\log\Big(\frac{n}{|E|}\Big)\Big)<O\Big(\frac{n}{|E|}\Big)\Longrightarrow O\Big(n|E|\log\Big(\frac{n}{|E|}\Big)\Big)<O(n^2)$$

3.3 Neighbor Sharing (NS) Method

Although the CA method reduces the time complexity for generating the network K-function plot, the time complexity of this method can still be high, especially for large number of data points n. Here, we ask a question. Can we further reduce the time complexity for generating the network K-function plot? In this section, we provide an affirmative answer to this question.

Recall that we need to count the data points p_j in the edge e with $dist_G(p_i,p_j)\le\tau$ from the data point p_i in the edge $\widetilde{e}$ in order to compute the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ (cf. Equation 11). In Figure 6, observe that there are four possible routes from p_i (on the edge $\widetilde{e}=(a,b)$) to the edge $e=(u,v)$ with length τ. Therefore, we only need to count those data points in the edge $e=(u,v)$ that are passed through by any of these four routes. Here, we let $\tau_{au}(p_i)$, $\tau_{bu}(p_i)$, $\tau_{av}(p_i)$, and $\tau_{bv}(p_i)$ be the lengths of these four routes in the edge $e=(u,v)$ (cf. Figure 6), where

$$\tau_{au}(p_i)=\tau-dist_G(p_i,a)-dist_G(a,u)\quad(14)$$
$$\tau_{bu}(p_i)=\tau-dist_G(p_i,b)-dist_G(b,u)\quad(15)$$
$$\tau_{av}(p_i)=\tau-dist_G(p_i,a)-dist_G(a,v)\quad(16)$$
$$\tau_{bv}(p_i)=\tau-dist_G(p_i,b)-dist_G(b,v)\quad(17)$$

As a remark, it is possible for any of these four values to be negative, which indicates that we do not need to consider that route (e.g., we can omit the route $p_i\to b\to u$ if $\tau_{bu}(p_i)<0$, i.e., $dist_G(p_i,b)+dist_G(b,u)>\tau$) for counting the number of data points.

Based on these four values $\tau_{au}(p_i)$, $\tau_{bu}(p_i)$, $\tau_{av}(p_i)$, and $\tau_{bv}(p_i)$, we maintain four sets of data points, i.e., $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$, respectively, in the edge $e=(u,v)$ for computing the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$, where

$$\ell_{au}(p_i)=\{p_j\in P(e):dist_G(u,p_j)\le\tau_{au}(p_i)\}\quad(18)$$
$$\ell_{bu}(p_i)=\{p_j\in P(e):dist_G(u,p_j)\le\tau_{bu}(p_i)\}\quad(19)$$
$$\ell_{av}(p_i)=\{p_j\in P(e):dist_G(v,p_j)\le\tau_{av}(p_i)\}\quad(20)$$
$$\ell_{bv}(p_i)=\{p_j\in P(e):dist_G(v,p_j)\le\tau_{bv}(p_i)\}\quad(21)$$

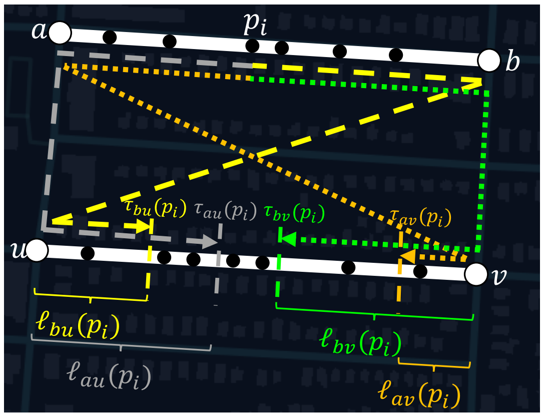


Figure 6: Four possible routes (grey dashed, yellow dashed, orange dotted, and green dotted lines) with length τ from the data point p_i in the edge $\widetilde{e} = (a, b)$ to the edge $e = (u, v)$, where the data points that are inside $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$ can contribute to $C_P^{(p_i,e)}(\tau)$.

In Lemma 4, we claim that we can compute $C_P^{(p_i,e)}(\tau)$ in $O(1)$ time, once we have these four sets of data points (cf. Equations 18 to 21).

Lemma 4. *Given the data point p_i in the edge $\widetilde{e} = (a, b)$ and the sets of data points $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$ in the edge $e = (u, v)$, the (p_i, e)-count function $C_P^{(p_i,e)}(\tau)$ can be computed in $O(1)$ time based on the following two conditions (**C1** and **C2**).*

C1 ($\max(|\ell_{au}(p_i)|, |\ell_{bu}(p_i)|) + \max(|\ell_{av}(p_i)|, |\ell_{bv}(p_i)|) > |P(e)|$):

$$C_P^{(p_i,e)}(\tau) = |P(e)| \tag{22}$$

C2 ($\max(|\ell_{au}(p_i)|, |\ell_{bu}(p_i)|) + \max(|\ell_{av}(p_i)|, |\ell_{bv}(p_i)|) \le |P(e)|$):

$$C_P^{(p_i,e)}(\tau) = \max(|\ell_{au}(p_i)|, |\ell_{bu}(p_i)|) + \max(|\ell_{av}(p_i)|, |\ell_{bv}(p_i)|) \tag{23}$$

Proof. Note that computing the (p_i, e)-count function $C_P^{(p_i,e)}(\tau)$ is equivalent to counting the number of data points that are covered by the sets $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$, i.e.,

$$C_P^{(p_i,e)}(\tau) = |\ell_{au}(p_i) \cup \ell_{bu}(p_i) \cup \ell_{av}(p_i) \cup \ell_{bv}(p_i)|$$

Observe from Figure 6 that $\ell_{au}(p_i) \cup \ell_{bu}(p_i)$ and $\ell_{av}(p_i) \cup \ell_{bv}(p_i)$ cover the data points in the edge $e = (u, v)$ that are from the node u and node v, respectively. Therefore, if $\ell_{au}(p_i) \cup \ell_{bu}(p_i)$ intersects with $\ell_{av}(p_i) \cup \ell_{bv}(p_i)$, i.e., $\max(|\ell_{au}(p_i)|, |\ell_{bu}(p_i)|) + \max(|\ell_{av}(p_i)|, |\ell_{bv}(p_i)|) > |P(e)|$, $\ell_{au}(p_i) \cup \ell_{bu}(p_i) \cup \ell_{av}(p_i) \cup \ell_{bv}(p_i) = P(e)$, i.e., covers all data points in the edge $e = (u, v)$. Hence, we have proved the correctness of the condition **C1**.

Suppose that $\ell_{au}(p_i) \cup \ell_{bu}(p_i)$ does not intersect with $\ell_{av}(p_i) \cup \ell_{bv}(p_i)$, i.e., $(\ell_{au}(p_i) \cup \ell_{bu}(p_i)) \cap (\ell_{av}(p_i) \cup \ell_{bv}(p_i)) = \phi$. We have:

$$\begin{aligned} C_P^{(p_i,e)}(\tau) &= |\ell_{au}(p_i) \cup \ell_{bu}(p_i) \cup \ell_{av}(p_i) \cup \ell_{bv}(p_i)| \\ &= |\ell_{au}(p_i) \cup \ell_{bu}(p_i)| + |\ell_{av}(p_i) \cup \ell_{bv}(p_i)| \\ &= \max(|\ell_{au}(p_i)|, |\ell_{bu}(p_i)|) + \max(|\ell_{av}(p_i)|, |\ell_{bv}(p_i)|) \end{aligned}$$

Hence, we have proved the correctness of the condition **C2**. □

Recall that we need to compute the (p_i, e)-count function $C_P^{(p_i,e)}(\tau)$ (cf. Equation 11) for each p_i in the edge $\widetilde{e} = (a, b)$ (cf. Figure 6) in order to compute the $(\widetilde{e}, e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$ (cf. Equation 10). In Figure 7, observe that we change from the data point p_i to the data point p_{i+1} in the edge $\widetilde{e} = (a, b)$ (i.e., away from the node a ($dist_G(p_{i+1}, a) > dist_G(p_i, a)$)). Based on Equations 14 and 16, we have:

$$\tau_{au}(p_{i+1}) \le \tau_{au}(p_i) \text{ and } \tau_{av}(p_{i+1}) \le \tau_{av}(p_i)$$

Consider Equations 18 and 20, we can further conclude that:

$$\ell_{au}(p_{i+1}) \subseteq \ell_{au}(p_i) \text{ and } \ell_{av}(p_{i+1}) \subseteq \ell_{av}(p_i)$$

Similarly, the values $\tau_{bu}(p_{i+1}) > \tau_{bu}(p_i)$ and $\tau_{bv}(p_{i+1}) > \tau_{bv}(p_i)$ (cf. Figure 8) if we change from p_i to p_{i+1}. Therefore, we have:

$$\ell_{bu}(p_{i+1}) \supseteq \ell_{bu}(p_i) \text{ and } \ell_{bv}(p_{i+1}) \supseteq \ell_{bv}(p_i)$$

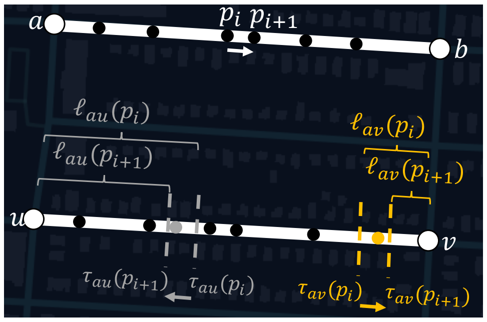


Figure 7: Remove the data points from $\ell_{au}(p_i)$ (grey data point) and $\ell_{av}(p_i)$ (orange data point) to obtain $\ell_{au}(p_{i+1})$ and $\ell_{av}(p_{i+1})$, respectively, once we change from p_i to p_{i+1} (i.e., move away from the node a) in the edge $\widetilde{e} = (a, b)$.

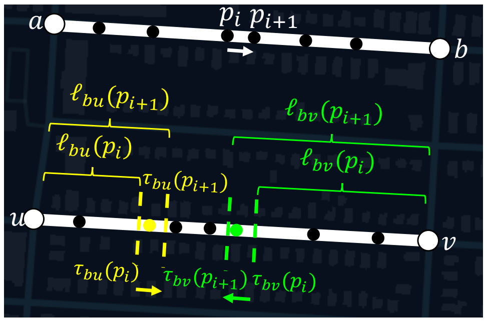


Figure 8: Insert the data points to $\ell_{bu}(p_i)$ (yellow data point) and $\ell_{bv}(p_i)$ (green data point) to obtain $\ell_{bu}(p_{i+1})$ and $\ell_{bv}(p_{i+1})$, respectively, once we change from p_i to p_{i+1} (i.e., move towards the node b) in the edge $\widetilde{e} = (a, b)$.

Hence, we conclude that these four sets of data points exhibit the monotonicity property (cf. Lemma 5). The proof of this lemma follows from the above discussion.

Lemma 5. *Given the data points p_i and p_{i+1} in the edge $\widetilde{e} = (a, b)$, where $dist_G(p_i, a) \le dist_G(p_{i+1}, a)$, and the sets of data points $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$ in the edge $e = (u, v)$, these four sets exhibit the monotonicity property, where*

$$\ell_{au}(p_{i+1}) \subseteq \ell_{au}(p_i) \text{ and } \ell_{av}(p_{i+1}) \subseteq \ell_{av}(p_i)$$

$$\ell_{bu}(p_{i+1}) \supseteq \ell_{bu}(p_i) \text{ and } \ell_{bv}(p_{i+1}) \supseteq \ell_{bv}(p_i)$$

Algorithm 4 Neighbor Sharing (NS) Method for $(\widetilde{e}, e)$-count function

```
1: procedure NS_EDGE($\widetilde{e} = (a,b)$, $e = (u,v)$, $SPD(a)$, $SPD(b)$, distance $\tau$)
2:     Obtain $dist_G(a,u)$, $dist_G(a,v)$                ▷ Obtain from $SPD(a)$
3:     Obtain $dist_G(b,u)$, $dist_G(b,v)$                ▷ Obtain from $SPD(b)$
4:     $C \leftarrow 0$
5:     for each $p_i \in P(\widetilde{e})$ do
6:         Update $\tau_{au}(p_i)$, $\tau_{av}(p_i)$              ▷ Equations 14, 16
7:         Update $\tau_{bu}(p_i)$, $\tau_{bv}(p_i)$              ▷ Equations 15, 17
8:         Update $\ell_{au}(p_i)$, $\ell_{av}(p_i)$              ▷ Equations 18, 20
9:         Update $\ell_{bu}(p_i)$, $\ell_{bv}(p_i)$              ▷ Equations 19, 21
10:        $C \leftarrow C + C_P^{(p_i,e)}(\tau)$                 ▷ Lemma 4
11:    Return $C$
```

Algorithm 4 shows how we compute the $(\widetilde{e}, e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$, once we have obtained the shortest path distances from the node a and node b (i.e., $SPD(a)$ and $SPD(b)$). Based on Lemma 4 and Lemma 5, we conclude that Algorithm 4 takes at most $O(|P(\widetilde{e})| + |P(e)|)$ time to compute $C_P^{(\widetilde{e},e)}(\tau)$ (cf. Lemma 6).

Lemma 6. *Given two edges $\widetilde{e} = (a,b)$ and $e = (u,v)$ in the graph $G = (V,E)$ and the shortest path distances $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$, Algorithm 4 takes $O(|P(\widetilde{e})| + |P(e)|)$ time to compute $C_P^{(\widetilde{e},e)}(\tau)$.*

Proof. In this proof, we show that the main bottleneck of this algorithm (i.e., lines 5 to 10) takes at most $O(|P(\widetilde{e})| + |P(e)|)$ time. Suppose that we process each data point p_i in $P(\widetilde{e})$ from node a to node b in the edge $\widetilde{e} = (a,b)$ in line 5 of Algorithm 4 (the data points in $P(\widetilde{e})$ are sorted in advance (cf. footnote 1).).

Since we can use Equations 14 to 17 to obtain $\tau_{au}(p_i)$, $\tau_{bu}(p_i)$, $\tau_{av}(p_i)$, and $\tau_{bv}(p_i)$ (lines 6 and 7) in $O(1)$ time for each p_i, the time complexity of lines 6 and 7 is $O(|P(\widetilde{e})|)$ time throughout the loop (in line 5).

Consider $\ell_{au}(p_i)$ in Algorithm 4. Based on Lemma 5, we can conclude that:

$$\ell_{au}(p_{|P(\widetilde{e})|}) \subseteq \cdots \subseteq \ell_{au}(p_2) \subseteq \ell_{au}(p_1)$$

where $p_1, p_2, \ldots, p_{|P(\widetilde{e})|}$ are the data points in the edge $\widetilde{e} = (a,b)$ with:

$$dist_G(a,p_1) \leq dist_G(a,p_2) \leq \cdots \leq dist_G(a,p_{|P(\widetilde{e})|})$$

Since the data points in $P(e)$ are sorted (cf. footnote 1), the total number of deleted points for maintaining $\ell_{au}(p_i)$, where $1 \leq i \leq |P(\widetilde{e})|$, in line 8 is $O(|P(e)|)$ throughout this loop (cf. Figure 9). Similarly, we can also prove that the total number of deleted points for $\ell_{av}(p_i)$ and the total number of inserted points for $\ell_{bu}(p_i)$ and $\ell_{bv}(p_i)$ are at most $O(|P(e)|)$ throughout this loop. As such, the time complexity of lines 8 and 9 in Algorithm 4 is $O(|P(\widetilde{e})| + |P(e)|)$ for all iterations.

Consider line 10 in Algorithm 4. Based on Lemma 4, we can compute $C_P^{(p_i,e)}(\tau)$ in $O(1)$ time for each p_i given the sets $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$ (obtained from lines 8 and 9). Therefore, the time complexity of line 10 is $O(|P(\widetilde{e})|)$ throughout the loop.

Hence, we conclude that Algorithm 4 takes $O(|P(\widetilde{e})| + |P(e)|)$ time to compute $C_P^{(\widetilde{e},e)}(\tau)$. □

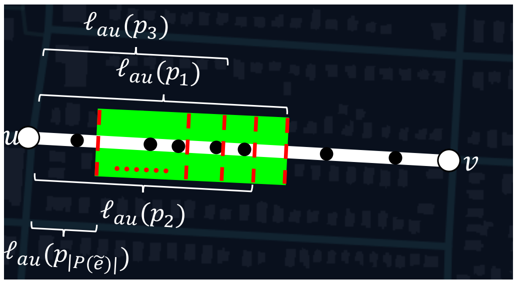


Figure 9: The green region denotes the set of deleted points in $e = (u,v)$, which contains at most $O(|P(e)|)$ points, throughout the loop (from $\ell_{au}(p_1)$ to $\ell_{au}(p_{|P(\widetilde{e})|})$).

Algorithm 5 Neighbor Sharing (NS) Method for Network K-function

```
1: procedure NS($G = (V,E)$, $P = \{p_1, p_2, \ldots, p_n\}$, distance $\tau$)
2:     $K \leftarrow 0$
3:     for each edge $\widetilde{e} = (a,b) \in E$ do
4:         $SPD(a) \leftarrow SP_\tau(G,a)$
5:         $SPD(b) \leftarrow SP_\tau(G,b)$
6:         for each edge $e = (u,v) \in E$ do
7:             //Use Algorithm 4
8:             $K \leftarrow K + NS_{EDGE}(\widetilde{e}, e, SPD(a), SPD(b), \tau)$
9:         Clear $SPD(a)$, $SPD(b)$
10:    Return $K$
```

Algorithm 5 shows how the NS method computes the network K-function (cf. Equation 9). Based on Lemma 6 and the footnote 1, we further claim that this algorithm only takes $O(|E|T_{SP} + n|E| + n\log n)$ time and $O(LD(|E|T_{SP} + n|E|) + Ln\log n)$ time to compute the network K-function and generate the network K-function plot, respectively (cf. Theorem 2).

Theorem 2. *The time complexity of using the NS method (cf. Algorithm 5) to compute the network K-function and generate the network K-function plot is $O(|E|T_{SP} + n|E| + n\log n)$ and $O(LD(|E|T_{SP} + n|E|) + Ln\log n)$, respectively.*

Proof. Since we need to find the shortest path distances from node a and node b, i.e., $SPD(a)$ and $SPD(b)$, respectively, for each edge $\widetilde{e} = (a,b)$, the time complexity of lines 4 and 5 for all iterations is $O(|E|T_{SP})$. With these shortest path distances (i.e., $SPD(a)$ and $SPD(b)$), the time complexity for computing the $(\widetilde{e}, e)$-count function $C_P^{(\widetilde{e},e)}(\tau)$ is $O(|P(\widetilde{e})| + |P(e)|)$ time (cf. Lemma 6). Therefore, the time complexity of line 8 for all iterations is:

$$\begin{aligned} & O\Big(\sum_{\widetilde{e}\in E}\sum_{e\in E}\big(|P(\widetilde{e})| + |P(e)|\big)\Big) \\ = \;& O\Big(|E|\sum_{\widetilde{e}\in E}|P(\widetilde{e})| + |E|\sum_{e\in E}|P(e)|\Big) = O(n|E|) \end{aligned}$$

Based on the above discussion and the footnote 1, the time complexity for using the NS method (cf. Algorithm 5) to compute the network K-function and generate the network K-function plot (cf. Problem 1) is $O(|E|T_{SP} + n|E| + n\log n)$ and $O(LD(|E|T_{SP} + n|E|) + Ln\log n)$, respectively. □

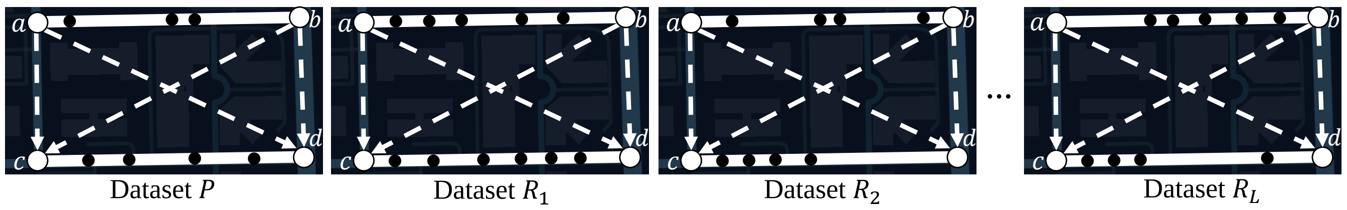


Figure 10: The shortest path distances from node a and node b (e.g., dashed lines) can be shared for computing the network K-function values in multiple datasets.

Compared with the CA method, which takes $O\big(LD\big(|E|T_{\text{SP}} + n|E|\log\big(\frac{n}{|E|}\big)\big) + Ln\log n\big)$ time to generate the network K-function plot (cf. Theorem 1), the NS method further removes the $\log\big(\frac{n}{|E|}\big)$ factor in order to theoretically improve the efficiency, especially for $n >> |E|$.

3.4 An Advanced Shortest Path Sharing (ASPS) Approach for Our Methods

In previous sections, both CA and NS methods mainly focus on efficiently counting the data points on a single edge $e = (u, v)$ in order to reduce the time complexity for computing the network K-function. However, none of these methods reduces the number of shortest path computations. Although the existing method, SPS (cf. Section 2.3), successfully achieves this goal, this method only focuses on improving the efficiency for computing a single network K-function. However, generating a network K-function plot (cf. Problem 1) involves computing many network K-functions. As such, we ask a question. Can we share the shortest path computations in order to further reduce the time complexity for generating a network K-function plot? To answer this question, we discuss the advanced shortest path sharing (ASPS) approach based on these two core ideas, namely (1) sharing the shortest path distances for multiple datasets and (2) sharing the shortest path distances for multiple distance thresholds, and investigate how to combine this approach with our CA and NS methods.

Core idea 1: Sharing the shortest path distances for multiple datasets. Suppose that we have computed the shortest path distances from node a and node b (cf. Figure 10) to all other nodes in the road network $G = (V, E)$. We can reuse these distance values (i.e., $SPD(a)$ and $SPD(b)$) for evaluating the network K-function values in multiple datasets, i.e., P, R_1,..., R_L (cf. Problem 1).

Core idea 2: Sharing the shortest path distances for multiple distance thresholds. Recall that all the methods need to call the shortest path algorithm, i.e., SP_τ, to find the shortest path distances for other nodes that are smaller than the distance threshold τ from a given node (e.g., line 4 and line 5 in Algorithm 5). Note that we have $\tau_1 \le \tau_2 \le \dots \le \tau_D$ (cf. Figure 2). After we use the shortest path algorithm with the largest distance threshold τ_D for any node a in the road network, i.e., call $SP_{\tau_D}(a)$, $SPD(a)$ (cf. Equation 7, where p_i is replaced by a, with the largest distance threshold τ_D) contains enough information of shortest path distances for other distance thresholds τ_1, τ_2,..., τ_{D-1}. Therefore, we can reuse this $SPD(a)$ and avoid calling the shortest path algorithms $SP_{\tau_1}(a)$, $SP_{\tau_2}(a)$,..., $SP_{\tau_{D-1}}(a)$.

Integration of the ASPS approach with our CA and NS methods. Here, we mainly discuss how to integrate the ASPS approach with our NS method, namely NS$^{(\text{ASPS})}$. By adopting the similar concept, we can also integrate this approach with the CA method (i.e., CA$^{(\text{ASPS})}$). In the NS$^{(\text{ASPS})}$ method, once we have computed the shortest path distances $SPD(a)$ and $SPD(b)$ for an edge $\widehat{e} = (a, b)$, we can share $SPD(a)$ and $SPD(b)$ for all $L + 1$ datasets (based on the core idea 1), i.e., $P, R_1, R_2, \dots, R_L$, and D distance thresholds (based on the core idea 2) in order to evaluate the corresponding edge-edge count functions (cf. Equation 10), without re-invoking the shortest path algorithm. Algorithm 6 shows the pseudocode of the NS$^{(\text{ASPS})}$ method for generating the network K-function plot.

Algorithm 6 Combine the ASPS Approach with the NS Method for Generating the Network K-function Plot

```
1:  procedure NS^(ASPS)($G = (V, E)$, Datasets $\{P, R_1, R_2, \dots, R_L\}$,
        distance thresholds $\tau_1, \tau_2, \dots, \tau_D$)
2:      Let $R_0 = P$ and $K_{ld}$ be the network K-function value using
        the $l^{\text{th}}$ dataset and $\tau_d$ distance threshold.
3:      //Initialization
4:      for $l \leftarrow 0$ to $L$ do
5:          for $d \leftarrow 1$ to $D$ do
6:              $K_{ld} \leftarrow 0$
7:      for each edge $\widetilde{e} = (a, b) \in E$ do
8:          $SPD(a) \leftarrow SP_{\tau_D}(G, a)$
9:          $SPD(b) \leftarrow SP_{\tau_D}(G, b)$
10:         for $l \leftarrow 0$ to $L$ do
11:             for $d \leftarrow 1$ to $D$ do
12:                 for each edge $e = (u, v) \in E$ do
13:                     //Use Algorithm 4
14:                     $C \leftarrow NS_{\text{EDGE}}(\widetilde{e}, e, SPD(a), SPD(b), \tau_d)$
15:                     $K_{ld} \leftarrow K_{ld} + C$
16:         Clear $SPD(a)$, $SPD(b)$
17:     Return $K_{ld}$, where $0 \le l \le L$ and $1 \le d \le D$
```

Since this algorithm ensures that we call the shortest path algorithm two times for each edge $\widetilde{e} = (a, b)$, the time complexity of shortest path computations (lines 8 to 9 throughout the loop (in line 7)) is at most $O(|E|T_{\text{SP}})$. Based on the discussion in Section 3.3, the time complexity of lines 10 to 15 throughout the loop is at most $O(nLD|E|)$ once the data points in each edge are sorted in advance for every dataset (cf. footnote 1). As such, we conclude that the time complexity of the NS$^{(\text{ASPS})}$ method and the CA$^{(\text{ASPS})}$ method (by adopting the similar concept) is $O(|E|T_{\text{SP}} + nLD|E| + Ln\log n)$ and $O\big(|E|T_{\text{SP}} + nLD|E|\log\big(\frac{n}{|E|}\big) + Ln\log n\big)$, respectively (cf. Theorem 3).

THEOREM 3. *The time complexity of the NS$^{(ASPS)}$ method and the CA$^{(ASPS)}$ method to generate the network K-function plot is* $O(|E|T_{SP}+$

$nLD|E| + Ln\log n)$ and $O\big(|E|T_{SP} + nLD|E|\log\big(\frac{n}{|E|}\big) + Ln\log n\big)$, respectively.

Hence, the CA$^{(\text{ASPS})}$ and NS$^{(\text{ASPS})}$ methods can further reduce the time complexity for generating the network K-function plot (cf. Problem 1) compared with the CA and NS methods, respectively. As a remark, we further discuss how to parallelize our methods in Section 7.5.

3.5 Space Complexity of Our Methods

In this section, we investigate the space complexity of all our methods, i.e., CA, NS, CA$^{(\text{ASPS})}$, and NS$^{(\text{ASPS})}$. Note that all these methods need to access the road network $G = (V, E)$ (with $O(|V|+|E|)$ space), access the L datasets (with $O(nL)$ space), and adopt the shortest path algorithm (with $O(\mathcal{S}_{\text{SP}})$ space). All these methods take at least $O(|V| + |E| + nL + \mathcal{S}_{\text{SP}})$ space. In the following, we consider the additional space consumption for all these methods.

CA method. Recall that we only need to augment the count values, $|P(p_j, u)|$ and $|P(p_j, v)|$, for each data point p_j in each edge $e = (u, v)$ (cf. Figure 5). Therefore, we need to store additional $O(nL)$ count values for these L datasets. As such, the space complexity of this method remains in $O(|V| + |E| + nL + \mathcal{S}_{\text{SP}})$.

NS method. Recall that we need to maintain at most four sets of data points (cf. Figure 6), i.e., $\ell_{au}(p_i)$, $\ell_{bu}(p_i)$, $\ell_{av}(p_i)$, and $\ell_{bv}(p_i)$, in the edge $e = (u, v)$ so as to compute $C_P^{(\widetilde{e}, e)}(\tau)$ (cf. line 8 in Algorithm 5). Since we can clear all these sets after we consider the next pair of $(\widetilde{e}, e)$, this algorithm takes $O(n)$ additional space (with $|P(e)| \to n$ in the worst case). Therefore, the space complexity remains in $O(|V| + |E| + nL + \mathcal{S}_{\text{SP}})$.

CA$^{(\text{ASPS})}$ and NS$^{(\text{ASPS})}$ methods. Like the SPS method, since we need to maintain and clear the shortest path distances from node a and node b, i.e., $SPD(a)$ and $SPD(b)$ (with size $O(|V|)$), in each iteration (cf. lines 8-9 and line 16 in Algorithm 6), these two methods, CA$^{(\text{ASPS})}$ and NS$^{(\text{ASPS})}$, take $O(|V| + nL)$ and $O(|V| + n)$ additional space, respectively. As such, the space complexity of these two methods remains in $O(|V| + |E| + nL + \mathcal{S}_{\text{SP}})$.

Based on the above discussion, we conclude the space complexity of all methods in Theorem 4.

Theorem 4. *The space complexity of using the CA, NS, CA$^{(ASPS)}$, and NS$^{(ASPS)}$ methods to generate the network K-function plot is $O(|V| + |E| + nL + \mathcal{S}_{SP})$.*

Therefore, all our methods remain the same space complexity compared with the existing RQS and SPS methods (cf. Table 1).

4 EXPERIMENTAL EVALUATION

In this section, we first discuss the experimental settings in Section 4.1. Then, we investigate the practical efficiency of different methods for computing a single network K-function in Section 4.2. After that, we further test the practical efficiency of different methods for generating a network K-function plot (cf. Problem 1) in Section 4.3. Lastly, we conduct the case study in Section 4.4 to demonstrate how to use the network K-function plot to examine the cluster properties for large-scale location datasets.

4.1 Experimental Settings

We choose four large-scale location datasets with different categories, including traffic accidents, 911 calls, 311 calls, and crime events, to conduct the efficiency experiments, which are summarized in Table 2. For each location dataset, we adopt the OSMnx software package[2] [13] to extract the corresponding road network from the OpenStreetMap [4].

Table 2: Datasets.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	n	Category
London [5]	19,984	75,610	822,382	Traffic accidents
Detroit [3]	16,294	51,819	1,931,000	911 calls
San Francisco [6]	6,488	21,013	4,771,272	311 calls
Chicago [2]	23,320	66,075	7,358,988	Crime events

In our experiments, we compare all our methods, CA, NS, CA$^{(\text{ASPS})}$, and NS$^{(\text{ASPS})}$, with the state-of-the-art methods, RQS and SPS (cf. Table 1). We implemented all these methods with C++[3] 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 each method and only report the response time that is smaller than one day (i.e., 86,400 sec).

4.2 Experiments for Computing a Single Network K-function

Although our CA and NS methods can theoretically reduce the time complexity for computing a single network K-function compared with the RQS and SPS methods (cf. Table 1), we do not know the practical efficiency improvement of our methods over these methods. To investigate this issue, we conduct the following experiments to test the response time for the RQS, SPS, CA, and NS methods. As a remark, since we focus on computing a single network K-function in this section, we omit the CA$^{(\text{ASPS})}$ and NS$^{(\text{ASPS})}$ methods, which only improve the efficiency for using multiple datasets and multiple thresholds (i.e., the setting of generating network K-function plot), for testing.

Response time of all methods using the default threshold τ: In this experiment, we follow [30] and choose the default threshold τ as 1000m for computing the network K-function. Table 3 summarizes the response time of all methods. Since our CA method can significantly reduce the time complexity for computing the network K-function (cf. Lemma 3), this method can achieve 6.41x to 28.58x speedup compared with the RQS and SPS methods. Furthermore, since our NS method can further remove the $\log\big(\frac{n}{|E|}\big)$ factor (cf. Theorem 2) compared with the CA method, this method can achieve the smallest response time for all datasets (with additional 2.12x to 5.86x speedup over the CA method).

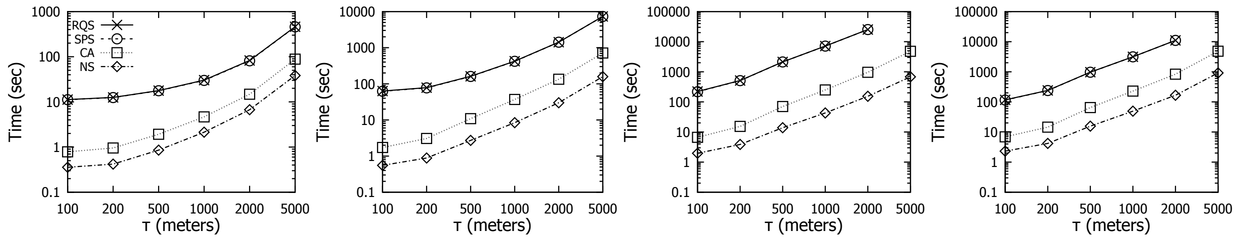
Table 3: Response time (sec) of all methods for computing the network K-function, using the default threshold τ = 1000m.

Dataset	RQS	SPS	CA	NS
London	31.02	30.33	4.69	**2.21**
Detroit	425.02	421.77	37.06	**8.79**
San Francisco	7145.53	7132.01	251.35	**42.88**
Chicago	3146.51	3140.63	229.03	**49.38**

Response time of all methods with different thresholds τ: We proceed to test how the threshold τ affects the response time of each

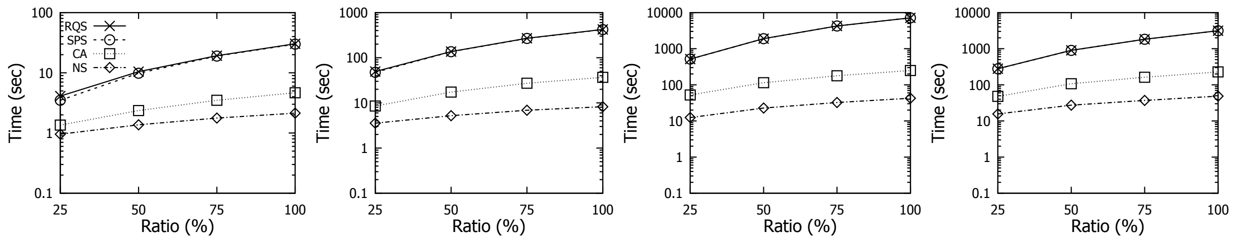
[2]https://github.com/gboeing/osmnx

[3]The implementation of all methods are available in the Github link: https://anonymous.4open.science/r/Network-K-function-44D9/



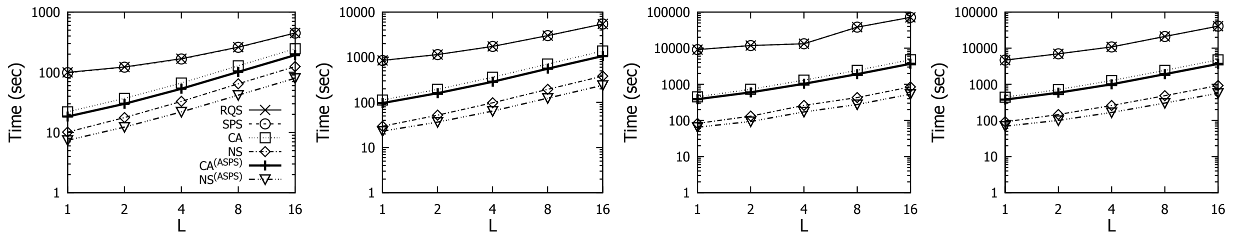
(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 11: Response time for computing a single network K-function, varying the threshold τ from 100 to 5000 meters.



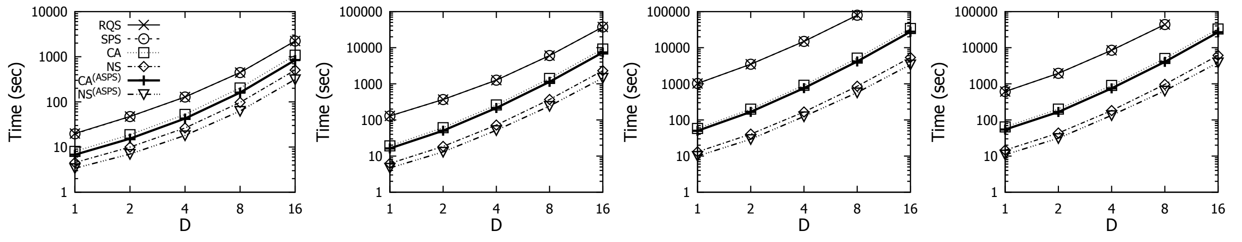
(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 12: Response time for computing a single network K-function, varying the dataset size.



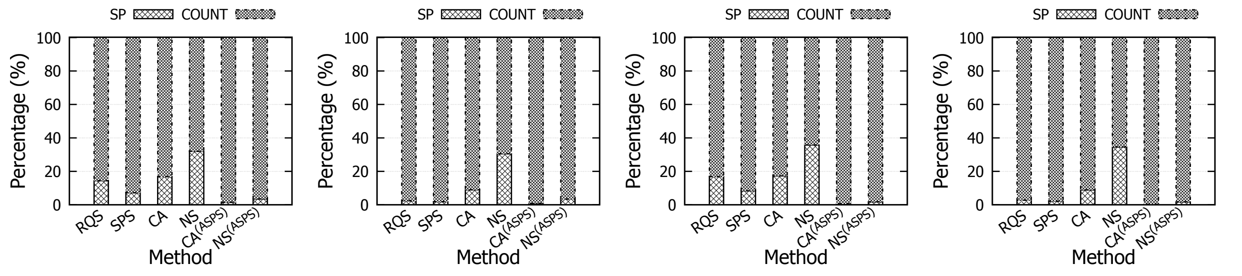
(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 13: Response time for generating a network K-function plot with $D = 5$, varying the number of location datasets L.



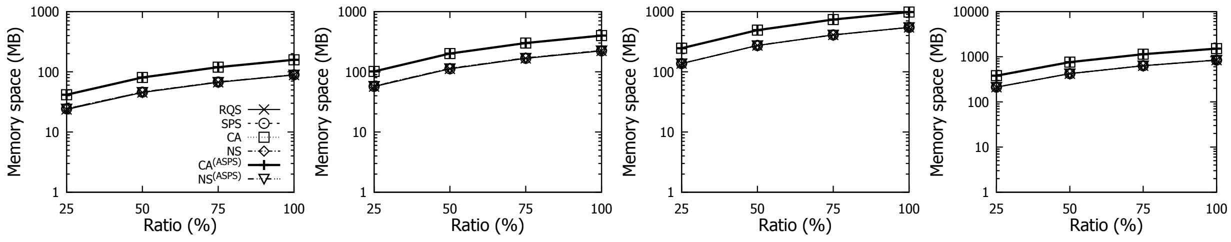
(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 14: Response time for generating a network K-function plot with $L = 5$, varying the number of distance thresholds D.



(a) London ($D = 4$) (b) Detroit ($D = 4$) (c) London ($D = 16$) (d) Detroit ($D = 16$)

Figure 15: Distributions of response time for different methods with $L = 5$ in the London (a and c) and Detroit datasets (b and d), using $D = 4$ (a and b) and $D = 16$ (c and d).



(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 16: Memory space consumption for generating a network K-function plot with $L = 5$ and $D = 5$, varying the dataset size.

method. To conduct this experiment, we choose six threshold values, which are 100, 200, 500, 1000, 2000, and 5000 meters, for testing. In Figure 11, once we increase the threshold τ, each method needs to process more data points. Therefore, the response time of each method increases. Since both our CA and NS methods can reduce the time complexity for computing the network K-function, these methods can achieve significant speedup compared with the RQS and SPS methods. In particular, the best method, NS, can achieve 11.95x to 165.85x speedup no matter which threshold τ we choose.

Response time of all methods with different dataset sizes: We investigate how the dataset size affects the response time of each method. In this experiment, we first randomly sample each dataset with different ratios, i.e., 25%, 50%, 75%, and 100% (the original one), and then run all methods for each sampled dataset with the default threshold $\tau = 1000$ meters. Figure 12 shows the response time of each method. Observe that our methods are more scalable to the dataset size, which can achieve up to 165.85x speedup compared with the RQS and SPS methods. The main reason is that our methods can significantly reduce the time complexity for computing the network K-function.

4.3 Experiments for Generating a Network K-function Plot

Recall that domain experts need to generate a network K-function plot (cf. Problem 1) by evaluating multiple network K-functions (with L randomly generated datasets and D distance thresholds). In this section, we test how these parameters, i.e., L and D, affect the response time of all methods. Furthermore, we also test the memory space consumption of all methods for generating a network K-function plot.

Response time of all methods with different numbers of datasets L: To conduct this experiment, we fix the number of distance thresholds $D = 5$, by choosing 100, 200, 300, 400, and 500 meters as the thresholds, and randomly generate $L = 1$, 2, 4, 8, and 16 additional datasets for testing (cf. Problem 1). Since the CA, NS, $\text{CA}^{(\text{ASPS})}$, and $\text{NS}^{(\text{ASPS})}$ methods can reduce the worst-case time complexity for generating a network K-function plot (cf. Table 1), all our methods can achieve up to 137.59x speedup compared with the existing methods, RQS and SPS (cf. Figure 13). Furthermore, our methods, $\text{CA}^{(\text{ASPS})}$ and $\text{NS}^{(\text{ASPS})}$, can achieve additional 15% to 40% speedup compared with the CA and NS methods, respectively. The smaller improvements of using the ASPS approach (e.g., the gaps between the NS and $\text{NS}^{(\text{ASPS})}$ methods in Figure 13) indicate that the main bottleneck of generating a network K-function plot is to count the data points in a road network from each data point rather than computing the shortest paths.

Response time of all methods with different numbers of thresholds D: We proceed to test how the number of thresholds D affects the response time of each method. In this experiment, we set the initial threshold value to be 100 and increase the threshold by 100 to generate D thresholds (e.g., $D = 4$ indicates that we adopt the four thresholds, $\tau_1 = 100$, $\tau_2 = 200$, $\tau_3 = 300$, and $\tau_4 = 400$, for testing). Furthermore, we fix the number of randomly generated datasets L to be 5 and vary the number of thresholds D from 1 to 16. In Figure 14, observe that all our methods can significantly improve the efficiency compared with the existing methods, RQS and SPS, due to the smaller time complexity of our methods (cf. Table 1). Specifically, the best method, $\text{NS}^{(\text{ASPS})}$, can achieve up to 141.62x speedup compared with the existing methods.

Distributions of response time for different methods: Recall that all methods need to (1) compute the shortest path distances (SP) and (2) count the number of data points (COUNT). Here, we investigate the distributions of response time in these two components, i.e., SP and COUNT, for different methods. To conduct this experiment, we fix L to be 5, set D to be 4 and 16, and choose two location datasets, which are London and Detroit, for testing. In Figure 15, observe that both the CA and NS methods contain the smaller percentages of response time in the COUNT component compared with the RQS and SPS methods. The main reason is that these two methods can reduce the time complexity for counting the data points in a road network. With the ASPS approach, which can significantly reduce the number of shortest path computations, both the $\text{CA}^{(\text{ASPS})}$ and $\text{NS}^{(\text{ASPS})}$ methods can significantly reduce the percentages of response time in the SP component compared with the CA and NS methods. Furthermore, the distributions of the same dataset (e.g., London dataset in Figure 15a and Figure 15c) are similar no matter which D we adopt.

Memory space consumption of all methods with different dataset sizes: We proceed to investigate the memory space consumption of all methods. In this experiment, we first sample 25%, 50%, 75%, and 100% (the original one) of data points from each dataset. Then, we measure the memory space consumption (in terms of MB) for generating a network K-function plot in each subset of the dataset, by fixing the parameters L and D to be 5. Observe from Figure 16 that all methods have the similar memory space consumption for different sample ratios. The main reason is that these methods have the same space complexity (cf. Table 1),

4.4 Case Study: Understanding the Cluster Properties for Crime Events in Chicago

In this section, we conduct the case study for using the network K-function plot to understand the cluster properties for crime events using the largest Chicago dataset. To conduct this experiment, we fix the default L to be 5 and vary the threshold τ_d from 100 to 1600. In Figure 17, observe that each network K-function value for the Chicago dataset, i.e., $K_P(\tau_d)$, is within the smallest and the largest K-function values of L randomly generated datasets when the thresholds τ_d are from 100 to 700 meters. Therefore, this indicates that the clusters are not meaningful (same as random distribution) if their sizes are within 100 to 700 meters. In contrast, the Chicago dataset has the meaningful clusters with sizes from 700 to 1600 meters. As such, once domain experts (e.g., criminologists [34]) adopt the cluster detection algorithms for finding the cluster locations in the Chicago dataset, they can verify whether the results are meaningful based on Figure 17 (e.g., they can ignore the results with sizes smaller than 700 meters).

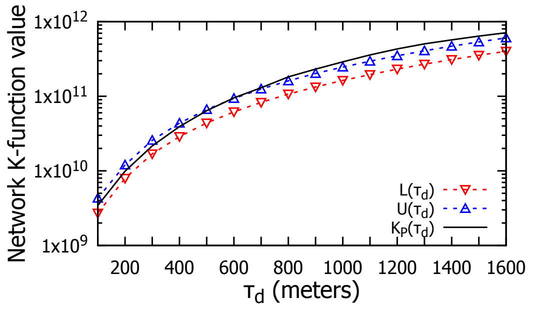


Figure 17: Network K-function plot for the Chicago dataset.

5 RELATED WORK

Network K-function has been extensively used to analyze location data in different communities, including criminology [20, 34], transportation [14, 57], ecology [41, 50], and urban planning [23, 55]. Due to its wide range of applications, the commonly-used software package, ArcGIS, can also support this operation (based on the SANET plugin [37, 38]). However, network K-function is a time-consuming operation, which involves $O(n^2)$ distance computation in the worst case. Worse still, domain experts need to compute multiple network K-function values in order to generate the network K-function plot (cf. Problem 1), which further deteriorates the inefficiency issue. In this section, we review four camps of research studies, which are mostly related to this work.

Spatial queries in a road network: To compute the network K-function (cf. Equation 1), we need to find those data points p_j in the road network that is within the distance τ from each data point p_i. As such, we can regard this as the spatial range query problem in the road network. Although many efficient algorithms [25, 27, 33, 42, 51, 53, 63] have been developed for solving the spatial range query problem and its variants (e.g., kNN query), none of these methods, to the best of our knowledge, can theoretically reduce the worst-case time complexity for these problems, let alone to reduce the time complexity for computing the network K-function.

Shortest path methods: Recall that computing the network K-function frequently invokes the shortest path algorithms. Therefore, an efficient shortest path method can be adopted to further improve the efficiency for generating the network K-function plot. In the literature, even though many shortest path methods, including hierarchical indexing [24, 64], hub labeling [9, 29, 32, 48], and shortest path sharing [31, 52], have been developed, most of these methods [9, 29, 31, 32, 48, 52] mainly focus on improving the efficiency of the single source single target shortest path query (rather than the spatial range query). As a remark, since we can replace the shortest path algorithm in the existing methods (e.g., RQS and SPS) and our methods (e.g., CA, NS, CA$^{(ASPS)}$, and NS$^{(ASPS)}$), developing an efficient shortest path algorithm is orthogonal to this work.

Kernel density visualization methods: Recently, Zheng et al. [60–62], Phillips et al. [43–45], and Chan et al. [16–18] have developed efficient algorithms for generating kernel density visualization (KDV), which is another important point pattern analysis operation. Although these methods can normally reduce the time complexity for generating KDV with non-trivial accuracy guarantees, these methods cannot be directly used for computing the network K-function. The main reasons are that (1) the network K-function (cf. Equation 1) is different from the density functions that are used in these research studies (e.g., Equation 1 in [17]) and (2) most of these research studies [16, 18, 43–45, 60–62] only focus on planar KDV (i.e., do not consider the road network).

Clustering and hotspot detection methods: In the literature, many effective algorithms have been developed for finding clusters and hotspots in a given location dataset. Some representative examples include k-means clustering [28], DBScan clustering [21], kernel clustering [15, 54], kernel density visualization [16], etc. However, unlike the network K-function, these methods cannot determine whether the clusters/hotspots in the dataset are meaningful/significant. As such, many research studies in the geospatial community [12, 26, 36, 49, 55] simultaneously adopt both the network K-function and other clustering/hotspot detection methods to thoroughly understand the location datasets.

6 CONCLUSION

In this paper, we study the network K-function, which has been extensively used in many geospatial applications for understanding hotspots and clusters [12, 14, 20, 30]. However, due to the high time complexity for computing the network K-function, none of the existing methods can be scalable to support large-scale location datasets. To overcome this issue, we develop the count augmentation (CA) and neighbor sharing (NS) methods that can theoretically reduce the worst-case time complexity for computing the network K-function. Furthermore, by incorporating the advanced shortest path sharing (ASPS) approach into our methods, CA and NS, we further achieve the lowest worst-case time complexity for generating the network K-function plot (cf. Problem 1). Our experiment results on four large-scale location datasets show that our methods can achieve up to 165.85x speedup compared with the state-of-the-art methods.

In the future, we will develop the QGIS and ArcGIS plugins that integrate our efficient methods for supporting the network K-function. Furthermore, we will examine the possibility to further improve the theoretical and practical efficiency for generating the network K-function plot. Moreover, we plan to extend our methods to improve the efficiency for other clustering methods (e.g., k-means clustering [59]) in road networks.

7 APPENDIX

7.1 Proof of Lemma 1

Proof. In Figure 5, observe that we can either use the route $p_i \to u \to p_j$ or the route $p_i \to v \to p_j$ to reach p_j from p_i. Since the shortest path distances $dist_G(a,u)$, $dist_G(a,v)$, $dist_G(b,u)$, and $dist_G(b,v)$ are available, we can find the shortest path distances $dist_G(p_i,u)$ and $dist_G(p_i,v)$ (dashed lines in Figure 5) in $O(1)$ time, using the following equations.

$$dist_G(p_i,u) = \min\begin{cases} dist_G(p_i,a)+dist_G(a,u) \\ dist_G(p_i,b)+dist_G(b,u)\end{cases} \tag{24}$$

$$dist_G(p_i,v) = \min\begin{cases} dist_G(p_i,a)+dist_G(a,v) \\ dist_G(p_i,b)+dist_G(b,v)\end{cases} \tag{25}$$

There are four possible cases for $dist_G(p_i,u)$ and $dist_G(p_i,v)$, which are (1) $dist_G(p_i,u) > \tau$ and $dist_G(p_i,v) > \tau$, (2) $dist_G(p_i,u) \le \tau$ and $dist_G(p_i,v) > \tau$, (3) $dist_G(p_i,u) > \tau$ and $dist_G(p_i,v) \le \tau$, and (4) $dist_G(p_i,u) \le \tau$ and $dist_G(p_i,v) \le \tau$. Here, we claim that we can use $O(\log|P(e)|)$ time to calculate the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ in these four cases.

Case 1 ($dist_G(p_i,u) > \tau$ and $dist_G(p_i,v) > \tau$): In Figure 5, we observe that:

$$\begin{aligned} & dist_G(p_i,p_j) \\ = \ & \min(dist_G(p_i,u)+dist_G(u,p_j), dist_G(p_i,v)+dist_G(v,p_j)) > \tau \end{aligned}$$

Based on Equation 11, we have $C_P^{(p_i,e)}(\tau) = 0$.

Case 2 ($dist_G(p_i,u) \le \tau$ and $dist_G(p_i,v) > \tau$): Since we have $dist_G(p_i,v) > \tau$, those data points with $dist_G(p_i,p_j) \le \tau$, i.e., red points in Figure 18, must have the shortest path that passes through the node u. Therefore, we need to use the binary search method to find p_j^*, where $dist_G(p_i,p_j^*)$ is just smaller than τ, i.e., $dist(u,p_j^*) \le \tau - dist_G(p_i,u)$ (cf. Figure 18). Then, we have $C_P^{(p_i,e)}(\tau) = |P(p_j^*,u)|$. As such, the time complexity for this case is $O(\log|P(e)|)$.

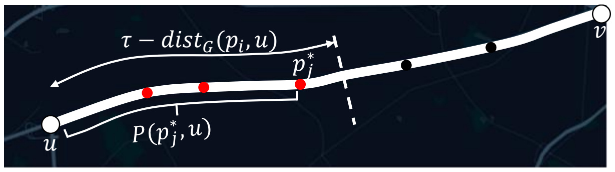


Figure 18: Case 2 of the CA method.

Case 3 ($dist_G(p_i,u) > \tau$ and $dist_G(p_i,v) \le \tau$): We omit the proof of this case, since the idea is the same as Case 2.

Case 4 ($dist_G(p_i,u) \le \tau$ and $dist_G(p_i,v) \le \tau$): Like Case 2, we can utilize the binary search method to find both p_L and p_R (cf. Figure 19), where $dist_G(u,p_L) \le \tau - dist_G(p_i,u)$ and $dist_G(v,p_R) \le \tau - dist_G(p_i,v)$, respectively, with $O(\log|P(e)|)$ time. Based on these two data points, we have $C_P^{(p_i,e)}(\tau) = \max(|P(p_L,u)| + |P(p_R,v)|, |P(e)|)$.

Since the time complexity of these four cases is $O(\log|P(e)|)$ and we only need $O(1)$ time to obtain $dist_G(p_i,u)$ and $dist_G(p_i,v)$, the time complexity for computing the (p_i,e)-count function $C_P^{(p_i,e)}(\tau)$ is $O(\log|P(e)|)$ time. □

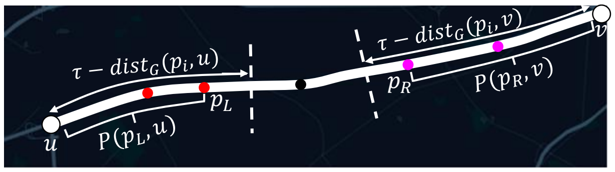


Figure 19: Case 4 of the CA method.

7.2 Proof of Lemma 2

Proof. Recall that $C_P^{(p_i,e)}(\tau)$ (cf. Equation 11) is the inner summation term of $C_P^{(\tilde{e},e)}(\tau)$ (cf. Equation 10), i.e.,

$$C_P^{(\tilde{e},e)}(\tau) = \sum_{p_i \in P(\tilde{e})} C_P^{(p_i,e)}(\tau)$$

Since it takes $O(\log|P(e)|)$ time to compute $C_P^{(p_i,e)}(\tau)$ (cf. Lemma 1), we conclude that the time complexity of computing $C_P^{(\tilde{e},e)}(\tau)$ is $O(|P(\tilde{e})|\log|P(e)|)$. □

7.3 Proof of Lemma 3

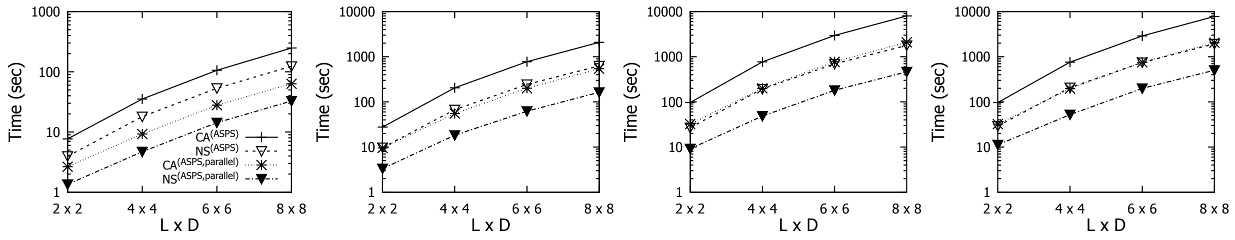
Proof. Recall that we need to sort all data points in $P(e)$ for each edge e in advance (cf. footnote 1), which takes $O(n\log n + |E|)$ time. In lines 3 to 6, this method needs to scan all data points for each edge e in E to augment the count values (cf. Equations 12 and 13) for each data point p_j, which takes $O(n+|E|)$ time. Then, we can find the shortest path distances from nodes a and b in the edge $\tilde{e} = (a,b)$ to all other nodes that are within the distance τ, which takes $O(T_{SP})$ time for each edge $\tilde{e}$. As such, lines 9 and 10 take $O(|E|T_{SP})$ time for all iterations. Based on Lemma 2, computing each $(\tilde{e},e)$-count function $C_P^{(\tilde{e},e)}(\tau)$ (cf. line 12) takes $O(|P(\tilde{e})|\log|P(e)|)$ time. Therefore, the time complexity of Algorithm 3 is:

$$\begin{aligned} & O\Big(n + |E| + |E|T_{SP} + \sum_{\tilde{e}\in E}\sum_{e\in E}|P(\tilde{e})|\log|P(e)| + n\log n\Big) \\ = \ & O\Big(n + |E|T_{SP} + \Big(\sum_{\tilde{e}\in E}|P(\tilde{e})|\Big)\Big(\sum_{e\in E}\log|P(e)|\Big) + n\log n\Big) \\ = \ & O\Big(|E|T_{SP} + n\sum_{e\in E}\log|P(e)| + n\log n\Big) \\ = \ & O\Big(|E|T_{SP} + n|E|\log\Big(\frac{n}{|E|}\Big) + n\log n\Big) \end{aligned}$$

In the last equality, we use the fact that $\sum_{e\in E}\log|P(e)| \le |E|\log\left(\frac{n}{|E|}\right)$, which has been proved in [17] (in Theorem 2). Hence, we conclude that the time complexity of Algorithm 3 is $O\big(|E|T_{SP} + n|E|\log\left(\frac{n}{|E|}\right) + n\log n\big)$ (cf. Lemma 3). □

7.4 Computing a Network K-function with Various Data Distributions

In this section, we further investigate how a data distribution can affect the efficiency of all methods for computing a network K-function. To construct the dataset based on a data distribution, we first randomly choose 10,000 initial points in a road network. Then, we generate 1,000 data points around each initial point, where the distance from each data point to the initial point is denoted by the multiplication of the threshold τ with the positive random value g (i.e., $\tau \times g$). In this experiment, we adopt the default threshold $\tau =$



(a) London (b) Detroit (c) San Francisco (d) Chicago

Figure 20: Response time for generating a network K-function plot with six cores, varying the number $L \times D$.

1000m and choose the Gaussian distribution (the mean $\mu = 1$ and the standard deviation $\sigma = 0.5$) and the exponential distribution (the default parameter $\lambda = 1$) for g in order to construct two synthetic datasets (with 10,000,000 (10,000 $\times$ 1,000) data points) in the largest Chicago road network (cf. Table 2). Table 4 shows the response time of all methods for computing the network K-function (using the default threshold $\tau = 1000$m) in these synthetic datasets. Observe that our methods can achieve 9.98x to 42.54x speedup, no matter which data distribution we adopt.

Table 4: Response time (sec) of all methods for computing the network K-function in each synthetic dataset (generated by a data distribution), using the default threshold $\tau = 1000$m.

Distribution	RQS	SPS	CA	NS
Gaussian	5580.44	5561.29	553.81	**130.73**
Exponential	5520.94	5498.17	550.96	**129.94**

7.5 Parallelization of Our Methods for Generating a Network K-function Plot

In order to further improve the efficiency for generating a network K-function plot, which involves computing multiple network K-functions with $L + 1$ datasets and D distance thresholds (cf. Problem 1), we discuss how to parallelize our methods, $\text{NS}^{(\text{ASPS})}$ and $\text{CA}^{(\text{ASPS})}$. Using $\text{NS}^{(\text{ASPS})}$ as an example, Algorithm 6 shares $SPD(a)$ and $SPD(b)$ (line 8 and line 9, respectively) to evaluate the $(\widehat{e}, e)$-count functions (cf. Equation 10) for all datasets and distance thresholds. As such, we adopt the round-robin approach to assign each thread for handling the (l, d)-pairs in lines 10 and 11 of Algorithm 6. Since different threads do not need to share the computational resources, this approach can parallelize our $\text{NS}^{(\text{ASPS})}$ method. The similar idea can also be applied for parallelizing the $\text{CA}^{(\text{ASPS})}$ method. Here, we term the parallel versions of $\text{CA}^{(\text{ASPS})}$ and $\text{NS}^{(\text{ASPS})}$ to be $\text{CA}^{(\text{ASPS, parallel})}$ and $\text{NS}^{(\text{ASPS, parallel})}$, respectively.

To verify the efficiency of our methods, we further conduct the experiment in an Intel PC with six cores. In this experiment, we vary $L \times D$ from 2×2, 4×4, 6×6, and 8×8.[4] Figure 20 shows the response time of all methods. Observe that $\text{CA}^{(\text{ASPS, parallel})}$ and $\text{NS}^{(\text{ASPS, parallel})}$ can further achieve 2.6x to 3.99x speedup compared with the $\text{CA}^{(\text{ASPS})}$ and $\text{NS}^{(\text{ASPS})}$ methods, respectively.

[4]Like the previous experiment for varying different numbers of thresholds D in Section 4.3, we set the initial threshold to be 100 and increase the threshold by 100 to generate D thresholds. As an example, $D = 4$ indicates that we adopt four thresholds with $\tau_1 = 100$, $\tau_2 = 200$, $\tau_3 = 300$, and $\tau_4 = 400$.

REFERENCES

[1] [n. d.]. ArcGIS. http://pro.arcgis.com/en/pro-app/tool-reference/spatial-analyst/how-kernel-density-works.htm.

[2] [n. d.]. Chicago Data Portal (last accessed: 3rd December 2021). https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2.

[3] [n. d.]. City of Detroit Open Data Portal (last accessed: 3rd December 2021). https://data.detroitmi.gov/datasets/detroitmi::911-calls-for-service/about.

[4] [n. d.]. OpenStreetMap (last accessed: 3rd December 2021). https://www.openstreetmap.org/.

[5] [n. d.]. Road Safety Data (last accessed: 3rd December 2021). https://data.gov.uk/dataset/cb7ae6f0-4be6-4935-9277-47e5ce24a11f/road-safety-data.

[6] [n. d.]. San Francisco Open Data (last accessed: 3rd December 2021). https://data.sfgov.org/City-Infrastructure/311-Cases/vw6y-z8j6.

[7] [n. d.]. SANET. http://sanet.csis.u-tokyo.ac.jp/.

[8] [n. d.]. spNetwork: Spatial Analysis on Network - R Project. https://cran.r-project.org/web/packages/spNetwork/spNetwork.pdf.

[9] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*. 349–360. https://doi.org/10.1145/2463676.2465315

[10] Amira K. Al-Aamri, Graeme Hornby, Li-Chun Zhang, Abdullah A. Al-Maniri, and Sabu S. Padmadas. 2021. Mapping road traffic crash hotspots using GIS-based methods: A case study of Muscat Governorate in the Sultanate of Oman. *Spatial Statistics* 42 (2021), 100458.

[11] Qi Wei Ang, Adrian Baddeley, and Gopalan Nair. 2012. Geometrically Corrected Second Order Analysis of Events on a Linear Network, with Applications to Ecology and Criminology. *Scandinavian Journal of Statistics* 39, 4 (2012), 591–617.

[12] Adrian Baddeley, Gopalan Nair, Suman Rakshit, Greg McSwiggan, and Tilman M. Davies. 2021. Analysing point patterns on networks — A review. *Spatial Statistics* 42 (2021), 100435.

[13] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126 – 139.

[14] Jiannan Cai, Min Deng, Qiliang Liu, Zhanjun He, Jianbo Tang, and Xuexi Yang. 2019. Nonparametric Significance Test for Discovery of Network-Constrained Spatial Co-Location Patterns. *Geographical Analysis* 51, 1 (2019), 3–22. https://doi.org/10.1111/gean.12155 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/gean.12155

[15] Daniele Calandriello and Lorenzo Rosasco. 2018. Statistical and Computational Trade-Offs in Kernel K-Means. In *NeurIPS*. 9379–9389.

[16] Tsz Nam Chan, Reynold Cheng, and Man Lung Yiu. 2020. QUAD: Quadratic-Bound-based Kernel Density Visualization. In *SIGMOD*. 35–50. https://doi.org/10.1145/3318464.3380561

[17] Tsz Nam Chan, Zhe Li, Leong Hou U, Jianliang Xu, and Reynold Cheng. 2021. Fast Augmentation Algorithms for Network Kernel Density Visualization. *Proc. VLDB Endow.* 14, 9 (2021), 1503–1516.

[18] Tsz Nam Chan, Man Lung Yiu, and Leong Hou U. 2019. KARL: Fast Kernel Aggregation Queries. In *ICDE*. 542–553. https://doi.org/10.1109/ICDE.2019.00055

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition.* MIT Press. http://mitpress.mit.edu/books/introduction-algorithms

[20] Matthias Eckardt and Jorge Mateu. 2018. Point Patterns Occurring on Complex Structures in Space and Space-Time: An Alternative Network Approach. *Journal of Computational and Graphical Statistics* 27, 2 (2018), 312–322.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php

[22] Yaxin Fan, Xinyan Zhu, Bing She, Wei Guo, and Tao Guo. 2018. Network-constrained spatio-temporal clustering analysis of traffic collisions in Jianghan District of Wuhan, China. *PLOS ONE* 13, 4 (04 2018), 1–23. https://doi.org/10.1371/journal.pone.0195093

[23] Carlos Garrocho-Rangel, José Antonio Álvarez Lobato, and Tania Chávez. 2013. Calculating Intraurban Agglomeration of Economic Units with Planar and Network K-Functions: A Comparative Analysis. *Urban Geography* 34, 2 (2013), 261–286.

[24] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. 2008. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *WEA*. 319–333. https://doi.org/10.1007/978-3-540-68552-4_24

[25] Abdeltawab M. Hendawi, Jie Bao, Mohamed F. Mokbel, and Mohamed H. Ali. 2015. Predictive tree: An efficient index for predictive queries on road networks. In *ICDE*. 1215–1226. https://doi.org/10.1109/ICDE.2015.7113369

[26] Alexander Hohl, Minrui Zheng, Wenwu Tang, Eric Delmelle, and Irene Casas. 2017. Spatiotemporal point pattern analysis using Ripley's K function. *Geospatial Data Science Techniques and Applications* (2017), 155–175.

[27] Haibo Hu, Dik Lun Lee, and Victor C. S. Lee. 2006. Distance Indexing on Road Networks. In *VLDB*. 894–905. http://dl.acm.org/citation.cfm?id=1164204

[28] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognit. Lett.* 31, 8 (2010), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011

[29] Ruoming Jin, Ning Ruan, Yang Xiang, and Victor E. Lee. 2012. A highway-centric labeling approach for answering distance queries on large sparse graphs. In *SIGMOD*. 445–456. https://doi.org/10.1145/2213836.2213887

[30] David S. Lamb, Joni A. Downs, and Chanyoung Lee. 2016. The network K-function in context: examining the effects of network structure on the network K-function. *Transactions in GIS* 20, 3 (2016), 448–460. https://doi.org/10.1111/tgis.12157 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12157

[31] Lei Li, Mengxuan Zhang, Wen Hua, and Xiaofang Zhou. 2020. Fast Query Decomposition for Batch Shortest Path Processing in Road Networks. In *ICDE*. 1189–1200. https://doi.org/10.1109/ICDE48307.2020.00107

[32] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An Experimental Study on Hub Labeling based Shortest Path Algorithms. *Proc. VLDB Endow.* 11, 4 (2017), 445–457. https://doi.org/10.1145/3186728.3164141

[33] Zijian Li, Lei Chen, and Yue Wang. 2019. G*-Tree: An Efficient Spatial Index on Road Networks. In *ICDE*. IEEE, 268–279. https://doi.org/10.1109/ICDE.2019.00032

[34] Yongmei Lu and Xuwei Chen. 2007. On the false alarm of planar K-function when analyzing urban crime distributed along streets. *Social Science Research* 36, 2 (2007), 611–632. https://doi.org/10.1016/j.ssresearch.2006.05.003

[35] Ömür Kaygisiz, Şebnem Düzgün, Ahmet Yildiz, and Metin Senbil. 2015. Spatio-temporal accident analysis for accident prevention in relation to behavioral factors in driving: The case of South Anatolian Motorway. *Transportation Research Part F: Traffic Psychology and Behaviour* 33 (2015), 128–140. https://doi.org/10.1016/j.trf.2015.07.002

[36] Jianhua Ni, Tianlu Qian, Changbai Xi, Yikang Rui, and Jiechen Wang. 2016. 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 (2016). https://www.mdpi.com/1660-4601/13/8/833

[37] Atsuyuki Okabe, Kei-Ichi Okunuki, and Shino Shiode. 2006. SANET: A Toolbox for Spatial Analysis on a Network. *Geographical Analysis* 38, 1 (2006), 57–66.

[38] Atsuyuki Okabe, Kei-Ichi Okunuki, and Shino Shiode. 2006. The SANET Toolbox: New Methods for Network Spatial Analysis. *Transactions in GIS* 10, 4 (2006), 535–550.

[39] A. Okabe and K. Sugihara. 2012. *Spatial Analysis Along Networks: Statistical and Computational Methods.* Wiley. https://books.google.com.hk/books?id=48GRqj51_W8C

[40] Atsuyuki Okabe and Ikuho Yamada. 2001. The K-Function Method on a Network and Its Computational Implementation. *Geographical Analysis* 33, 3 (2001), 271–290. https://doi.org/10.1111/j.1538-4632.2001.tb00448.x

[41] E. Okwori, M. Viklander, and A. Hedström. 2021. Spatial heterogeneity assessment of factors affecting sewer pipe blockages and predictions. *Water Research* 194 (2021), 116934. https://doi.org/10.1016/j.watres.2021.116934

[42] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. 2003. Query Processing in Spatial Network Databases. In *VLDB*. 802–813. https://doi.org/10.1016/B978-012722442-8/50076-8

[43] Jeff M. Phillips. 2013. ϵ-Samples for Kernels. In *SODA*. 1622–1632. https://doi.org/10.1137/1.9781611973105.116

[44] Jeff M. Phillips and Wai Ming Tai. 2018. Improved Coresets for Kernel Density Estimates. In *SODA*. 2718–2727. https://doi.org/10.1137/1.9781611975031.173

[45] Jeff M. Phillips and Wai Ming Tai. 2018. Near-Optimal Coresets of Kernel Density Estimates. In *SOCG*. 66:1–66:13. https://doi.org/10.4230/LIPIcs.SoCG.2018.66

[46] Suman Rakshit, Adrian Baddeley, and Gopalan Nair. 2019. Efficient Code for Second Order Analysis of Events on a Linear Network. *Journal of Statistical Software, Articles* 90, 1 (2019), 1–37. https://doi.org/10.18637/jss.v090.i01

[47] Suman Rakshit, Gopalan Nair, and Adrian Baddeley. 2017. Second-order analysis of point patterns on a network using any distance metric. *Spatial Statistics* 22 (2017), 129–154.

[48] Michael N. Rice and Vassilis J. Tsotras. 2010. Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions. *Proc. VLDB Endow.* 4, 2 (2010), 69–80. https://doi.org/10.14778/1921071.1921074

[49] Yikang Rui, Zaigui Yang, Tianlu Qian, Shoaib Khalid, Nan Xia, and Jiechen Wang. 2016. Network-constrained and category-based point pattern analysis for Suguo retail stores in Nanjing, China. *International Journal of Geographical Information Science* 30, 2 (2016), 186–199.

[50] Peter Spooner, Ian Lunt, Atsu Okabe, and Shino Shiode. 2004. Spatial Analysis of Roadside Acacia Populations on a Road Network using the Network K-Function. *Landscape Ecology* 19 (07 2004), 491–499.

[51] Weiwei Sun, Chunan Chen, Baihua Zheng, Chong Chen, and Peng Liu. 2015. An Air Index for Spatial Query Processing in Road Networks. *IEEE Trans. Knowl. Data Eng.* 27, 2 (2015), 382–395. https://doi.org/10.1109/TKDE.2014.2330836

[52] Jeppe Rishede Thomsen, Man Lung Yiu, and Christian S. Jensen. 2012. Effective caching of shortest paths for location-based services. In *SIGMOD*. 313–324. https://doi.org/10.1145/2213836.2213872

[53] Haojun Wang and Roger Zimmermann. 2011. Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks. *IEEE Trans. Knowl. Data Eng.* 23, 7 (2011), 1065–1078. https://doi.org/10.1109/TKDE.2010.171

[54] Shusen Wang, Alex Gittens, and Michael W. Mahoney. 2019. Scalable Kernel K-Means Clustering with Nystrom Approximation: Relative-Error Bounds. *Journal of Machine Learning Research* 20 (2019), 12:1–12:49. http://jmlr.org/papers/v20/17-517.html

[55] Teng Wang, Yandong Wang, Xiaoming Zhao, and Xiaokang Fu. 2018. Spatial distribution pattern of the customer count and satisfaction of commercial facilities based on social network review data in Beijing, China. *Computers, Environment and Urban Systems* 71 (2018), 88–97. https://doi.org/10.1016/j.compenvurbsys.2018.04.005

[56] Zhensheng Wang and Ke Nie. 2019. Measuring Spatial Patterns of Health Care Facilities and Their Relationships with Hypertension Inpatients in a Network-Constrained Urban System. *International Journal of Environmental Research and Public Health* 16, 17 (2019). https://www.mdpi.com/1660-4601/16/17/3204

[57] Ikuho Yamada and Jean-Claude Thill. 2004. Comparison of planar and network K-functions in traffic accident analysis. *Journal of Transport Geography* 12, 2 (2004), 149–158. https://doi.org/10.1016/j.jtrangeo.2003.10.006

[58] Ikuho Yamada and Jean-Claude Thill. 2007. Local Indicators of Network-Constrained Clusters in Spatial Point Patterns. *Geographical Analysis* 39, 3 (2007), 268–292. https://doi.org/10.1111/j.1538-4632.2007.00704.x

[59] Man Lung Yiu and Nikos Mamoulis. 2004. Clustering Objects on a Spatial Network. In *SIGMOD*. 443–454. https://doi.org/10.1145/1007568.1007619

[60] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. 2013. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*. 433–444.

[61] Yan Zheng, Yi Ou, Alexander Lex, and Jeff M. Phillips. 2021. Visualization of Big Spatial Data Using Coresets for Kernel Density Estimates. *IEEE Trans. Big Data* 7, 3 (2021), 524–534. https://doi.org/10.1109/TBDATA.2019.2913655

[62] Yan Zheng and Jeff M. Phillips. 2015. L∞ Error and Bandwidth Selection for Kernel Density Estimates of Large Data. In *SIGKDD*. 1533–1542. https://doi.org/10.1145/2783258.2783357

[63] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, Lizhu Zhou, and Zhiguo Gong. 2015. G-Tree: An Efficient and Scalable Index for Spatial Search on Road Networks. *IEEE Trans. Knowl. Data Eng.* 27, 8 (2015), 2175–2189. https://doi.org/10.1109/TKDE.2015.2399306

[64] Andy Diwen Zhu, Hui Ma, Xiaokui Xiao, Siqiang Luo, Youze Tang, and Shuigeng Zhou. 2013. Shortest path and distance queries on road networks: towards bridging theory and practice. In *SIGMOD*. 857–868. https://doi.org/10.1145/2463676.2465277