MASS: A Complexity-Optimal Solution for Product Kernel Density Visualization

Yue Zhong
College of Computer Science and Software Engineering, Shenzhen University
Shenzhen, China
2310273008@email.szu.edu.cn

Tsz Nam Chan*
College of Computer Science and Software Engineering, Shenzhen University
Shenzhen, China
edisonchan@szu.edu.cn

Leong Hou U
Department of Computer and Information Science, University of Macau
Macao, China
ryanlhu@um.edu.mo

Dingming Wu
College of Computer Science and Software Engineering, Shenzhen University
Shenzhen, China
dingming@szu.edu.cn

Wei Tu
Ruisheng Wang
School of Architecture and Urban Planning, Shenzhen University
Shenzhen, China
{tuwei,ruiswang}@szu.edu.cn

Joshua Zhexue Huang
College of Computer Science and Software Engineering, Shenzhen University
Shenzhen, China
zx.huang@szu.edu.cn

Abstract

Product Kernel Density Visualization (product KDV) has been widely used in various fields, including urban planning, epidemiology, traffic science, and crime science. However, product KDV is a time-consuming tool, which does not scale to support large-scale location datasets and high resolution sizes. Even worse, there is a lack of research studies for handling this tool. To tackle the efficiency issues of product KDV, we first develop the new indexing structure, called MAtrix of Search Space (MASS), and then develop two complexity-reduced methods, called MASS_{CR} (the basic version) and MASS_{OPT} (the advanced version). Note that MASS_{OPT} is the first optimal solution (with $O(XY + n)$ time) for generating exact product KDV, i.e., there will be no exact solution with the lower time complexity for supporting this tool in the future. Experimental results with four large-scale location datasets (up to 37.4 million data points) verify that our methods can achieve 3.08x to 135.31x speedups compared with existing methods for supporting this tool. The implementation of all methods can be found in https://github.com/YovelaZ/product_KDV/.

CCS Concepts

• **Theory of computation** → **Design and analysis of algorithms**.

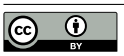
Keywords

Product KDV, Data structures, Optimal time complexity

ACM Reference Format:

Yue Zhong, Tsz Nam Chan, Leong Hou U, Dingming Wu, Wei Tu, Ruisheng Wang, and Joshua Zhexue Huang. 2026. MASS: A Complexity-Optimal Solution for Product Kernel Density Visualization. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770855.3817698

*Corresponding author.



KDD 126, August 09-13, 2026, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2259-2/2026/08
https://doi.org/10.1145/3770855.3817698

1 Introduction

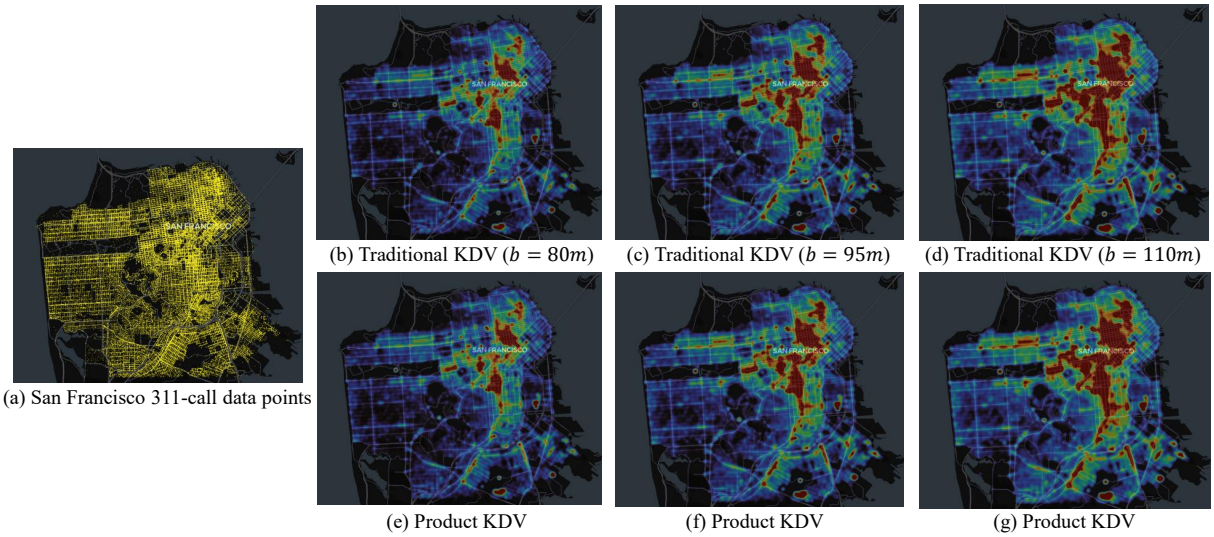
Heatmap [11] is an important tool for visualizing spatial data. Among most of the tools, Kernel Density Visualization (KDV) [18, 39] is a prevalent tool for heatmap generation that has been widely adopted in different fields for spatial analysis, including urban planning, epidemiology, traffic science, and crime science. Urban planners [42, 44] adopt KDV to analyze mobility patterns of different geographical regions. Epidemiologists [20, 31] adopt KDV to perform disease outbreak detection. Traffic experts [26, 35] and crime researchers [33, 34] adopt KDV to discover traffic/traffic accident hotspots and crime hotspots, respectively, in different cities and countries. Due to its wide range of applications, many software suites can also support KDV. Some representative examples include Scikit-learn [36], statsmodels [10], Scipy [8], QGIS [4], ArcGIS [6], Deck.gl [7], and Seaborn [9].

To generate traditional KDVs [18] (see Figure 1b to Figure 1d) for a location dataset P (see Figure 1a), we need to color each pixel $\mathbf{q}$ based on computing the kernel density function $\mathcal{D}_P^T(\mathbf{q})$ (using the Epanechnikov kernel function as an example) for each pixel $\mathbf{q}$.

$$\mathcal{D}_P^T(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot \begin{cases} 1 - \frac{1}{b^2} d(\mathbf{q},\mathbf{p})^2 & \text{if } d(\mathbf{q},\mathbf{p}) \le b \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where w, $d(\mathbf{q},\mathbf{p})$, and b represent the normalization constant, Euclidean distance, and the bandwidth parameter, respectively. Note that this bandwidth parameter can significantly affect the heatmap visualization (see Figure 1b to Figure 1d).

However, one major drawback of using traditional KDV is that this tool suffers from the time complexity of $O(XYn)$, where $X \times Y$ and n denote the resolution size and the number of location data points (see the yellow points in Figure 1a). Hence, this tool is not scalable to generate high-resolution heatmaps for a large-scale location dataset. Although the state-of-the-art method, called SLAM [18], can alleviate this efficiency issue, by successfully reducing the time complexity of generating the traditional KDV from



(a) San Francisco 311-call data points
(b) Traditional KDV ($b = 80m$)
(c) Traditional KDV ($b = 95m$)
(d) Traditional KDV ($b = 110m$)
(e) Product KDV ($b_x = 73m$ and $b_y = 73m$)
(f) Product KDV ($b_x = 88m$ and $b_y = 88m$)
(g) Product KDV ($b_x = 103m$ and $b_y = 103m$)

Figure 1: Generating traditional and product KDVs for the San Francisco 311-call location dataset.

Table 1: Theoretical results of exact methods for generating product KDV.

Method	Time complexity	Space complexity	Ref.	Remark
SLAM	$O(Y(X+n))$ or $O(X(Y+n))$	$O(XY+n)$	Section 2.2 (Modified from [18])	N/A
MASS_{CR}	$O((XY+n)\log XY)$ (Theorem 1)	$O(XY+n)$ (Theorem 2)	Section 3 and Section 8.1	N/A
MASS_{OPT}	$O(XY+n)$ (Theorem 3)	$O(XY+n)$ (Theorem 4)	Section 4 and Section 8.1	**Optimal**

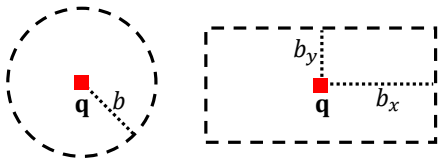
$O(XYn)$ to $O(Y(X+n))/O(X(Y+n))$, the time complexity still depends on either the Xn term or the Yn term. Therefore, this method (1) can still be inefficient for large $X \times Y$ and large n and (2) cannot achieve real-time performance (< 0.5 seconds) for supporting exploratory/interactive heatmap visualization.

Recently, many domain experts [24, 25, 27, 29, 41, 43] propose to generate KDV by multiplying a kernel function along each axis (a.k.a. product KDV), where the kernel density function $\mathcal{D}_P(\mathbf{q})$ (with the Epanechnikov kernel function as an example) is represented as follows.

$$\mathcal{D}_P(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot \begin{cases} 1 - \frac{1}{b_x^2} d(\mathbf{q}_x, \mathbf{p}_x)^2 & \text{if } d(\mathbf{q}_x, \mathbf{p}_x) \le b_x \\ 0 & \text{otherwise} \end{cases} \cdot \begin{cases} 1 - \frac{1}{b_y^2} d(\mathbf{q}_y, \mathbf{p}_y)^2 & \text{if } d(\mathbf{q}_y, \mathbf{p}_y) \le b_y \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Here, b_x and b_y denote the bandwidth parameters along the x-axis and y-axis, respectively, and $\mathbf{q}_x$ ($\mathbf{p}_x$) and $\mathbf{q}_y$ ($\mathbf{p}_y$) denote the x-coordinate and y-coordinate of the pixel $\mathbf{q}$ (the data point $\mathbf{p}$), respectively.

Compared with traditional KDVs (see Figure 1b to Figure 1d), we can observe that the corresponding product KDVs (see Figure 1e to Figure 1g) can achieve similar heatmap visualizations no matter which bandwidth parameter b of the traditional KDV we adopt.[1] However, unlike traditional KDV, there is a lack of fast algorithms for improving the efficiency of generating product KDV, which still takes the time complexity of $O(XYn)$. Although we can simply extend SLAM [18] for reducing the time complexity of generating product KDV from $O(XYn)$ to $O(Y(X+n))/O(X(Y+n))$ (will be discussed in Section 2.2), there is no obvious benefit for using the product KDV.



(a) Traditional KDV (b) Product KDV

Figure 2: The search space between the traditional KDV and the product KDV.

In this paper, we have the following observation. Note that $\mathcal{D}_P^T(\mathbf{q})$ (see Equation 1) depends on the circular search space $\{\mathbf{p} \in P : d(\mathbf{q}, \mathbf{p}) \le b\}$ (see Figure 2a), while $\mathcal{D}_P(\mathbf{q})$ (see Equation 2) depends on the rectangular search space $\{\mathbf{p} \in P : d(\mathbf{q}_x, \mathbf{p}_x) \le b_x$ and $d(\mathbf{q}_y, \mathbf{p}_y) \le b_y\}$ (see Figure 2b). With this major difference, we ask a question. *Can we develop a new solution that can further reduce the time complexity for supporting product KDV?* To provide a certain answer to this question, we develop a new index structure, called MAtrix of Search Space (MASS), and augment some information for this structure. Based on MASS, we further develop a new method, called MASS_{CR}, which significantly reduces the time complexity to $O((XY+n)\log XY)$. By further incorporating the new interval decomposition approach into MASS_{CR}, we establish the complexity-optimal method, called MASS_{OPT}, which

[1] More visualization results can be found in the supplementary material (Section 1 in [49]).

only takes $O(XY + n)$ time for generating product KDV. Table 1 summarizes the theoretical results of all methods. **Due to the optimality of MASS$_{\text{OPT}}$, there will be no other solution that can achieve theoretically lower time complexity compared with this method.** Experimental results on four large-scale location datasets (up to 37 million data points) verify that our methods can achieve 3.08x to 135.31x speedups compared with existing methods. Our case study also shows that **product KDV (with MASS$_{\text{OPT}}$) can achieve real-time performance (< 0.5 seconds) for the London crime dataset [2] (with 4.346 million data points), while traditional KDV (with SLAM) can no longer achieve this performance.**

This paper is structured as follows. We first discuss the preliminaries in Section 2. Then, we illustrate our methods, called MASS$_{\text{CR}}$ and MASS$_{\text{OPT}}$, in Section 3 and Section 4, respectively. After that, we present the experiments in Section 5. Then, we provide the literature review in Section 6. Lastly, we conclude this paper in Section 7. Appendix can be found in Section 8.

2 Preliminaries

In this section, we first formally define the problem of product KDV in Section 2.1. Then, we extend the state-of-the-art traditional KDV solution, called SLAM [18], for solving this problem in Section 2.2.

2.1 Problem Statement

Recall that we need to color each pixel $\mathbf{q}$ based on the product kernel density function $\mathcal{D}_P(\mathbf{q})$ in order to generate a product KDV (e.g., Figure 1e to Figure 1g) based on location data points (e.g., Figure 1a), which is formally defined in Problem 1.

Problem 1. *Given a resolution size $X \times Y$, a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with size n, we need to color each pixel $\mathbf{q}$ by computing the product kernel density function $\mathcal{D}_P(\mathbf{q})$, where*

$$\mathcal{D}_P(\mathbf{q}) = \sum_{\mathbf{p} \in P} w \cdot K_{b_x}(\mathbf{q}_x, \mathbf{p}_x) \cdot K_{b_y}(\mathbf{q}_y, \mathbf{p}_y) \tag{3}$$

where $K_{b_x}(\mathbf{q}_x, \mathbf{p}_x)$ and $K_{b_y}(\mathbf{q}_y, \mathbf{p}_y)$ denote the kernel function along the x-axis with the bandwidth value b_x and the kernel function along the y-axis with the bandwidth value b_y, respectively. Some widely used kernel functions can be found in Table 2.[2]

Table 2: Some widely used kernel functions, where q, p, and b can be $\mathbf{q}_x$ or $\mathbf{q}_y$, $\mathbf{p}_x$ or $\mathbf{p}_y$, and b_x or b_y, respectively.

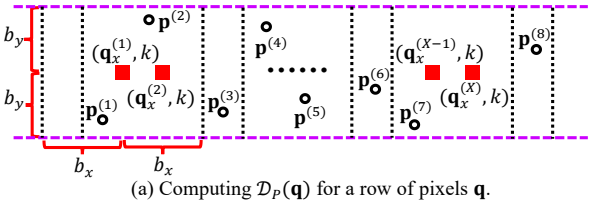
Kernel	$K_b(q, p)$	Ref.
Uniform	$\begin{cases} \frac{1}{b} & \text{if } d(q,p) \le b \\ 0 & \text{otherwise} \end{cases}$	[27, 40]
Triangular	$\begin{cases} 1 - \frac{1}{b} d(q,p) & \text{if } d(q,p) \le b \\ 0 & \text{otherwise} \end{cases}$	[28, 32]
Epanechnikov	$\begin{cases} 1 - \frac{1}{b^2} d(q,p)^2 & \text{if } d(q,p) \le b \\ 0 & \text{otherwise} \end{cases}$	[24, 27]

In this paper, we mainly focus on the commonly used Epanechnikov kernel function (see Equation 2). Those techniques for supporting other kernel functions will be discussed in Section 8.1.

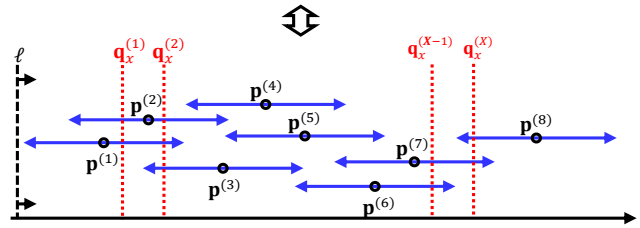
[2]The problem studied in this paper applies only to polynomial kernels, such as those listed in Table 2.

2.2 A State-of-the-art Solution: SLAM

Here, we discuss how to extend SLAM [18] for generating the product KDV. Consider computing $\mathcal{D}_P(\mathbf{q})$ (see Equation 2) for a row of X red pixels (see Figure 3a), i.e., $\mathbf{q}_y^{(1)} = \mathbf{q}_y^{(2)} = ... = \mathbf{q}_y^{(X)} = k$. Observe from Equation 2 that only those data points $\mathbf{p}$ (i.e., all white circles, which are $\mathbf{p}^{(1)}$, $\mathbf{p}^{(2)}$,..., $\mathbf{p}^{(8)}$ in Figure 3a) with $d(k, \mathbf{p}_y) = |k - \mathbf{p}_y| \le b_y$ can possibly contribute to $\mathcal{D}_P(\mathbf{q})$. Therefore, SLAM first retains those data points $\mathbf{p}$ in P that are within the envelope $|k - \mathbf{p}_y| \le b_y$ (covered by the pink dashed lines in Figure 3a). Since $\mathcal{D}_P(\mathbf{q})$ also has the constraint $d(\mathbf{q}_x, \mathbf{p}_x) \le b_x$ (see Equation 2), we note that those data points $\mathbf{p}$ inside the envelope with $\mathbf{p}_x - b_x \le \mathbf{q}_x \le \mathbf{p}_x + b_x$ can contribute to $\mathcal{D}_P(\mathbf{q})$. As such, SLAM then maintains the interval $[\mathbf{p}_x - b_x, \mathbf{p}_x + b_x]$ for each data point $\mathbf{p}$ (see those blue intervals in Figure 3b) and converts the computation of $\mathcal{D}_P(\mathbf{q})$ for a row of pixels $\mathbf{q}$ to solving the interval stabbing problem. By adopting the sweep line approach (see ℓ in Figure 3b), SLAM takes $O(X + n)$ time for computing $\mathcal{D}_P(\mathbf{q})$ for a row of pixels. Hence, the time complexity of SLAM for generating the product KDV is $O(Y(X + n))$ (or $O(X(Y + n))$ if we consider a column of pixels).



(a) Computing $\mathcal{D}_P(\mathbf{q})$ for a row of pixels $\mathbf{q}$.



(b) Solving the interval stabbing problem.

Figure 3: Computing $\mathcal{D}_P(\mathbf{q})$ for a row of pixels $\mathbf{q}$ is equivalent to solving the interval stabbing problem.

3 MASS$_{\text{CR}}$: A Complexity-Reduced Solution

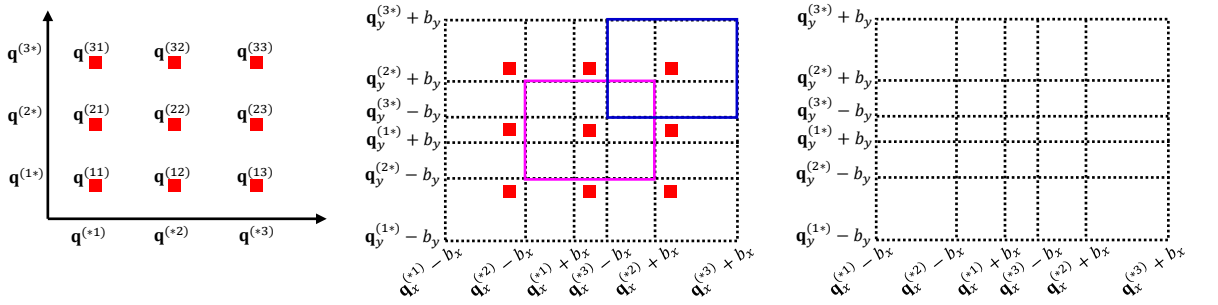
In this section, we first discuss the representation of the product kernel density function $\mathcal{D}_P(\mathbf{q})$ in Section 3.1. Then, we construct the index structure, called MAtrix of Search Space (MASS), in Section 3.2. Next, we discuss how to augment some information for MASS in Section 3.3. After that, we construct the prefix matrices for MASS and discuss how these matrices can be adopted for efficiently evaluating $\mathcal{D}_P(\mathbf{q})$ in Section 3.4. Lastly, we illustrate our solution, MASS$_{\text{CR}}$, based on the above concepts in Section 3.5.

3.1 Representation of $\mathcal{D}_P(\mathbf{q})$

Observe from Equation 2 that only those data points $\mathbf{p}$ that are within the search space $\mathbb{S}(\mathbf{q})$ (see Figure 2b), as formally stated in Equation 4, can contribute to $\mathcal{D}_P(\mathbf{q})$.

$$\mathbb{S}(\mathbf{q}) = \{\mathbf{p} \in P : d(\mathbf{q}_x, \mathbf{p}_x) \le b_x \text{ and } d(\mathbf{q}_y, \mathbf{p}_y) \le b_y\} \tag{4}$$

Hence, we have



(a) A 3 × 3-resolution plane (b) Search space for each pixel (c) Construction of a 5 × 5-matrix of search space

Figure 4: Construction of the index structure, called the matrix of search space (MASS), for a 3×3-resolution plane, where $\mathbf{q}^{(uv)}$, $\mathbf{q}^{(u*)}$, and $\mathbf{q}^{(*v)}$ denote the pixel with the u^{th} row and the v^{th} column, any pixel in the u^{th} row, and any pixel in the v^{th} column, respectively. As an example, the pink rectangle and the blue rectangle denote the search space $\mathbb{S}(\mathbf{q}^{(22)})$ and the search space $\mathbb{S}(\mathbf{q}^{(33)})$, respectively.

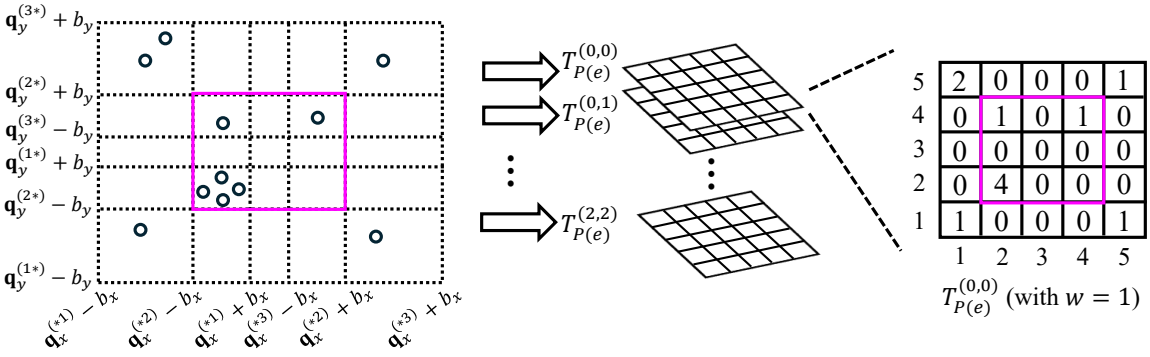


Figure 5: Augment the polynomial terms $T^{(i,j)}_{P(e)}$ ($0 \le i \le 2$ and $0 \le j \le 2$) for each entry e, where $P(e)$ denotes those white data points $\mathbf{p}$ that are inside the entry e. For example, the entry e with the position $(2, 2)$ for $T^{(0,0)}_{P(e)}$ (with $w = 1$) stores the value 4.

$$
\begin{aligned}
&\mathcal{D}_P(\mathbf{q}) \\
&= \sum_{\mathbf{p}\in\mathbb{S}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x^2} d(\mathbf{q}_x, \mathbf{p}_x)^2\right) \cdot \left(1 - \frac{1}{b_y^2} d(\mathbf{q}_y, \mathbf{p}_y)^2\right) \\
&= \left(1 - \frac{\mathbf{q}_x^2}{b_x^2}\right)\left(1 - \frac{\mathbf{q}_y^2}{b_y^2}\right) T^{(0,0)}_{\mathbb{S}(\mathbf{q})} + \frac{2\mathbf{q}_x}{b_x^2}\left(1 - \frac{\mathbf{q}_y^2}{b_y^2}\right) T^{(1,0)}_{\mathbb{S}(\mathbf{q})} - \frac{1}{b_x^2}\left(1 - \frac{\mathbf{q}_y^2}{b_y^2}\right) T^{(2,0)}_{\mathbb{S}(\mathbf{q})} \\
&+ \frac{2\mathbf{q}_y}{b_y^2}\left(1 - \frac{\mathbf{q}_x^2}{b_x^2}\right) T^{(0,1)}_{\mathbb{S}(\mathbf{q})} + \frac{4\mathbf{q}_x\mathbf{q}_y}{b_x^2 b_y^2} T^{(1,1)}_{\mathbb{S}(\mathbf{q})} - \frac{2\mathbf{q}_y}{b_x^2 b_y^2} T^{(2,1)}_{\mathbb{S}(\mathbf{q})} \\
&- \frac{1}{b_y^2}\left(1 - \frac{\mathbf{q}_x^2}{b_x^2}\right) T^{(0,2)}_{\mathbb{S}(\mathbf{q})} - \frac{2\mathbf{q}_x}{b_x^2 b_y^2} T^{(1,2)}_{\mathbb{S}(\mathbf{q})} + \frac{1}{b_x^2 b_y^2} T^{(2,2)}_{\mathbb{S}(\mathbf{q})} \qquad (5)
\end{aligned}
$$

where each $T^{(i,j)}_{\mathbb{S}(\mathbf{q})}$ ($0 \le i \le 2$ and $0 \le j \le 2$) denotes the polynomial term (see Equation 6) which only depends on those data points in the dataset P that are inside the search space $\mathbb{S}(\mathbf{q})$.

$$T^{(i,j)}_{\mathbb{S}(\mathbf{q})} = \sum_{\mathbf{p}\in\mathbb{S}(\mathbf{q})} w \cdot \mathbf{p}_x^i \mathbf{p}_y^j \qquad (6)$$

Therefore, this work aims to efficiently obtain $T^{(i,j)}_{\mathbb{S}(\mathbf{q})}$ for each pixel $\mathbf{q}$ in order to efficiently compute $\mathcal{D}_P(\mathbf{q})$.

3.2 Construction of MASS

Observe from Figure 2b that the search space for this pixel $\mathbf{q}$ is the rectangular region $[\mathbf{q}_x - b_x, \mathbf{q}_x + b_x][\mathbf{q}_y - b_y, \mathbf{q}_y + b_y]$. Therefore, by considering the search space for each pixel $\mathbf{q}^{(uv)}$ in Figure 4a, e.g., $\mathbb{S}(\mathbf{q}^{(22)})$ (the pink rectangle) and $\mathbb{S}(\mathbf{q}^{(33)})$ (the blue rectangle) in Figure 4b, we can construct the matrix of search space, i.e., MASS (see Figure 4c). In Lemma 1, we formally state that we can construct a $(2X-1)\times(2Y-1)$ MASS for a $(X\times Y)$-resolution plane in $O(XY)$ time. The proof of this lemma can be found in Section 2.1 of [49].

Lemma 1. *Given a resolution size $X \times Y$, a bandwidth parameter (along the x-axis) b_x, and a bandwidth parameter (along the y-axis) b_y, we can construct the corresponding MASS with $(2X-1)\times(2Y-1)$ entries in $O(XY)$ time.*

3.3 Augmentation for MASS

Recall that we need to efficiently evaluate $T^{(i,j)}_{\mathbb{S}(\mathbf{q})}$ (see Equation 6) in order to efficiently evaluate $\mathcal{D}_P(\mathbf{q})$ (see Equation 5). Therefore, we aim to maintain the augmented polynomial terms $T^{(i,j)}_{P(e)}$ (see Equation 7), where $0 \le i \le 2$ and $0 \le j \le 2$, for each entry e of MASS so that we can use them to efficiently compute $T^{(i,j)}_{\mathbb{S}(\mathbf{q})}$.

$$T^{(i,j)}_{P(e)} = \sum_{\mathbf{p}\in P(e)} w \cdot \mathbf{p}_x^i \cdot \mathbf{p}_y^j \qquad (7)$$

where $P(e)$ denotes the set of data points inside the entry e.

Figure 5 shows an example for how we augment $T^{(i,j)}_{P(e)}$ in each entry e of MASS. Initially, we set the polynomial terms $T^{(i,j)}_{P(e)}$, where $0 \le i \le 2$ and $0 \le j \le 2$, for each entry e to be 0. Consider each data point $\mathbf{p}$ in the dataset P. We need to utilize the binary search method along the x-axis and the y-axis to identify the correct entry e. Then, we can update this entry e by adding $w \cdot \mathbf{p}_x^i \cdot \mathbf{p}_y^j$ for maintaining

each value of $T_{P(e)}^{(i,j)}$ (see Equation 7). In Lemma 2, we state that the time complexity of augmenting the polynomial terms $T_{P(e)}^{(i,j)}$ for each entry e with respect to all data points $\mathbf{p}$ in P is $O(XY + n \log XY)$ time. The proof of this lemma can be found in Section 2.2 of [49].

Lemma 2. *Given a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ and an index structure MASS, augmenting the polynomial terms $T_{P(e)}^{(i,j)}$ for each entry e with respect to all data points $\mathbf{p}$ in P is $O(XY + n \log XY)$ time.*

3.4 Prefix Matrices for MASS

Once we have augmented $T_{P(e)}^{(i,j)}$ for MASS, we can compute $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$ (see Equation 6) based on Equation 8.

$$T_{\mathbb{S}(\mathbf{q})}^{(i,j)} = \sum_{e:P(e)\subseteq\mathbb{S}(\mathbf{q})} T_{P(e)}^{(i,j)} \tag{8}$$

Since the search space $\mathbb{S}(\mathbf{q})$ forms the rectangular region, all those entries e with $P(e) \subseteq \mathbb{S}(\mathbf{q})$ must also form the rectangular region. Using the search space $\mathbb{S}(\mathbf{q}^{(22)})$ (see the pink rectangles in Figures 4 and 5) as an example, this search space covers all entries in the pink rectangular region. Hence, we can conclude that **computing $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$ (see Equation 8) is equivalent to counting numbers in a matrix (i.e., MASS with $T_{P(e)}^{(i,j)}$) with any rectangular region.**

Note that this problem can be solved efficiently based on the prefix matrix structure [30]. Using Figure 6b as an example, this is the prefix matrix for MASS with $T_{P(e)}^{(0,0)}$ (with $w = 1$) in Figure 6a. Observe that the entry e with the index (u, v) of this prefix matrix represents the count of all numbers from the entries with the indices in the range $[1, u][1, v]$ of MASS (with $T_{P(e)}^{(0,0)}$). As an example, the entry with the index $(3, 2)$ that is colored in green in Figure 6b stores the value 5 since the rectangle with the light blue outline in Figure 6a covers the count value $1 + 0 + 0 + 0 + 4 + 0$. With this property, we can compute $T_{\mathbb{S}(\mathbf{q})}^{(0,0)}$ with any rectangular search space $\mathbb{S}(\mathbf{q})$ by accessing at most four entries in the prefix matrix. As an example for computing $T_{\mathbb{S}(\mathbf{q}^{(22)})}^{(0,0)}$ in Figure 5, which is 6, we only need to first use the binary search method for searching, based on $S(\mathbf{q}^{(22)})$ in Figure 5, the correct indices, i.e., $(4, 4)$, $(4, 1)$, $(1, 4)$, and $(1, 1)$ in Figure 6, and then access the corresponding (yellow) entries of the prefix matrix, where $7 - 1 - 1 + 1 = 6$. Similar concepts can be extended to other (i, j)-pairs.

5	2	0	0	0	1
4	0	1	0	1	0
3	0	0	0	0	0
2	0	4	0	0	0
1	1	0	0	0	1
	1	2	3	4	5

(a) $T_{P(e)}^{(0,0)}$ (with $w = 1$).

5	3	8	8	9	11
4	1	6	6	7	8
3	1	5	5	5	6
2	1	5	5	5	6
1	1	1	1	1	2
	1	2	3	4	5

(b) Prefix matrix for $T_{P(e)}^{(0,0)}$.

Figure 6: An example of prefix matrix for MASS with the polynomial term $T_{P(e)}^{(0,0)}$ (with $w = 1$) in Figure 5, which can be extended to other (i, j)-pairs.

Here, we conclude in Lemma 3 that, given the index structure (MASS), computing $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$ (including the construction of the prefix matrices and accessing them) for all pixels $\mathbf{q}$ takes $O(XY \log XY)$ time. The proof of this lemma can be found in Section 2.3 of [49].

Lemma 3. *Given an index structure, MASS, with augmented polynomial terms $T_{P(e)}^{(i,j)}$ (see Equation 7), $0 \le i \le 2$ and $0 \le j \le 2$, for each entry e, the time complexity of computing $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$ (see Equation 8) for all pixels $\mathbf{q}$ and (i, j)-pairs is $O(XY \log XY)$.*

3.5 Illustration of MASS_{CR}

Based on the discussion in Section 3.1 to Section 3.4, we present the pseudocode of our MASS-based complexity-reduced solution, called MASS_{CR}, in Algorithm 1.

Algorithm 1 MASS-based Complexity-Reduced Solution (MASS_{CR})

```
1: procedure MASS_CR(Point set P = {p_1, p_2, ..., p_n}, resolution size X × Y)
2:     Create the pixel plane ℙ with X × Y pixels.
3:     Construct MASS 𝕄 with size (2X − 1) × (2Y − 1).
4:     Augment 𝕄 with T_{P(e)}^{(i,j)} for each entry e and (i, j)-pair.
5:     Construct the prefix matrices for 𝕄 with all (i, j)-pairs.
6:     for each q ∈ ℙ do
7:         Compute T_{𝕊(q)}^{(i,j)} with all (i, j)-pairs.
8:         Compute 𝒟_P(q).                          ▷ Color q in ℙ.
9:     Return ℙ
```

Here, we conclude in Theorem 1 that the time complexity for using MASS_{CR} to generate a product KDV is $O((XY + n) \log XY)$. The proof of this theorem can be found in Section 2.4 of [49].

Theorem 1. *Given a resolution size $X \times Y$ and a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with size n, the time complexity of MASS_{CR} for generating a product KDV is $O((XY + n) \log XY)$.*

In Theorem 2, we further state that the space complexity of MASS_{CR} is $O(XY + n)$, which is proved in Section 2.5 of [49].

Theorem 2. *Given a resolution size $X \times Y$ and a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with size n, the space complexity of MASS_{CR} for generating a product KDV is $O(XY + n)$.*

4 MASS_{OPT}: A Complexity-Optimal Solution

Although MASS_{CR} can achieve the lower time complexity for generating a product KDV, this solution requires the binary search method for (1) finding the correct position of each data point in order to maintain the augmented polynomial terms $T_{P(e)}^{(i,j)}$ in each entry e of MASS (see Figure 5a) and (2) finding the correct indices of the prefix matrices based on the search space $\mathbb{S}(\mathbf{q})$ for each pixel $\mathbf{q}$ (see the rectangles with pink outline in Figures 5 and 6), which incurs the $O(\log XY)$ term for the time complexity (see Theorem 1). However, the lower bound time complexity of generating a product KDV is $\Omega(XY + n)$ (as any algorithm needs to at least access all data points and pixels once in order to obtain enough information). Therefore, MASS_{CR} is still not an optimal solution. Here, we ask a question. *Is it possible to further modify MASS_{CR} by removing the binary search method so that we can develop the optimal solution?* To provide a positive answer for this question, we first discuss the core concept, called interval decomposition, in Section 4.1, and then illustrate the optimal solution, MASS_{OPT}, in Section 4.2.

4.1 Interval Decomposition

Although MASS can exhibit the irregular sizes in different entries (see Figure 4c), we show that it is still possible to get rid of the binary

search method (i.e., use $O(1)$ time) for identifying the correct entry for each data point (e.g., each white data point in Figure 5a) and finding the correct indices of the prefix matrices based on the search space $\mathbb{S}(\mathbf{q})$ (see the rectangles with pink outline in Figures 5 and 6).

Here, we only consider finding the correct position along the x-axis for each data point $\mathbf{p}$. Similar ideas can be extended to y-axis and finding the correct indices in the prefix matrices based on the search space $\mathbb{S}(\mathbf{q})$. In Figure 7, observe that we can decompose all intervals of search space along the x-axis into two sequences, (1) $\{\mathbf{q}_x^{(*1)} - b_x, \mathbf{q}_x^{(*2)} - b_x, ..., \mathbf{q}_x^{(*X)} - b_x\}$ (i.e., red dashed lines) and (2) $\{\mathbf{q}_x^{(*1)} + b_x, \mathbf{q}_x^{(*2)} + b_x, ..., \mathbf{q}_x^{(*X)} + b_x\}$ (i.e., blue dashed lines).

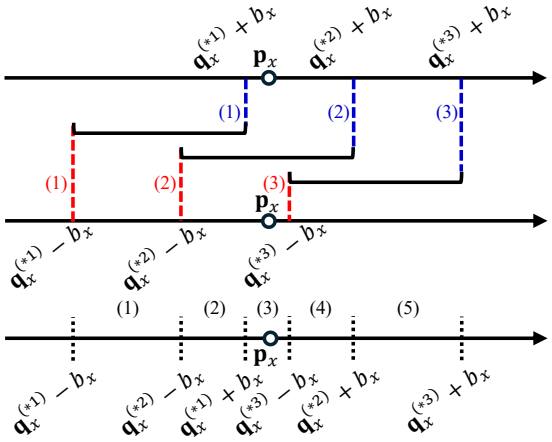


Figure 7: Decomposition of each black interval of search space (or range) in the x-axis into the red dashed line and the blue dashed line, where $\mathbf{p}_x$ should be in the 3rd entry of the x-axis, which just passes through two red dashed lines and one blue dashed line.

With this decomposition, we note that the data point $\mathbf{p}$ is in the α^{th} entry if $\mathbf{p}_x$ just passes through the α^{th} black dashed line (see Figure 7), regardless of which color (red or blue) those lines from 1 to α belong to. Therefore, we can obtain the correct index of the entry **by counting the number of red dashed lines and the number of blue dashed lines that are passed through by $\mathbf{p}_x$ (see Figure 7).** Since we also note that the difference between any two consecutive pixels in the x-axis must have the same value, say δ_x, (i.e., $\mathbf{q}_x^{(*2)} - \mathbf{q}_x^{(*1)} = \mathbf{q}_x^{(*3)} - \mathbf{q}_x^{(*2)} = ... = \mathbf{q}_x^{(*X)} - \mathbf{q}_x^{(*X-1)} = \delta_x$), we also know that the difference between any two consecutive values of the sequence (1) or sequence (2) must also be δ_x. Therefore, we can use $O(1)$ time to identify the correct index $\mathbb{I}_x$ of the entry in the x-axis by adopting the following equation.

$$\mathbb{I}_x = \min\left(\max\left(\left\lfloor \frac{\mathbf{p}_x - (\mathbf{q}_x^{(*1)} - b_x)}{\delta_x} \right\rfloor, 0\right), X\right) + \min\left(\max\left(\left\lfloor \frac{\mathbf{p}_x - (\mathbf{q}_x^{(*1)} + b_x)}{\delta_x} \right\rfloor, 0\right), X\right) \quad (9)$$

where the values 0 and $2X$ denote that those data points are not inside any entry in MASS (see Figure 5a), which can be discarded.

4.2 Illustration of MASS$_{\text{OPT}}$

Based on the core concept of the interval decomposition, we conclude that the time complexity of augmenting the polynomial terms $T_{P(e)}^{(i,j)}$ for each entry e in MASS with respect to all data points $\mathbf{p}$ in the dataset P (see line 4 in Algorithm 1) can be reduced from $O(XY + n \log XY)$ (see Lemma 2) to $O(XY + n)$ (see Lemma 4).

Lemma 4. *Given a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ and an index structure MASS, using the interval decomposition approach to augment the polynomial terms $T_{P(e)}^{(i,j)}$ for each entry e with respect to all data points $\mathbf{p}$ in P is $O(XY + n)$ time.*

Furthermore, we can also conclude that the time complexity of computing $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$ (see line 7 in Algorithm 1) for all pixels $\mathbf{q}$ and (i, j)-pairs can be reduced from $O(XY \log XY)$ (see Lemma 3) to $O(XY)$ (see Lemma 5), by omitting the binary search method, based on the interval decomposition approach.

Lemma 5. *Given an index structure, MASS, with augmented polynomial terms $T_{P(e)}^{(i,j)}$ (see Equation 7), $0 \le i \le 2$ and $0 \le j \le 2$, for each entry e, the time complexity of computing $T_{\mathbb{S}(\mathbf{q})}^{(i,j)}$, which is based on the interval decomposition approach, for all pixels $\mathbf{q}$ and (i, j)-pairs is $O(XY)$.*

With the above modifications, we can conclude in Theorem 3 that the time complexity of MASS$_{\text{OPT}}$ is $O(XY + n)$, which is the same as the lower bound time complexity $\Omega(XY + n)$ (i.e., achieves the optimal solution).

Theorem 3. *Given a resolution size $X \times Y$ and a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with size n, the time complexity of MASS$_{OPT}$ for generating a product KDV is $O(XY + n)$.*

Recall that the interval decomposition approach aims to adopt some mathematical formula (e.g., Equation 9) to identify the correct index of the entry in $O(1)$ time, which does not construct any additional data structure. The space complexity of MASS$_{\text{OPT}}$ remains the same as MASS$_{\text{CR}}$, which is $O(XY + n)$ (see Theorem 4).

Theorem 4. *Given a resolution size $X \times Y$ and a location dataset $P = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ with size n, the space complexity of MASS$_{OPT}$ for generating a product KDV is $O(XY + n)$.*

5 Experimental Evaluation

In this section, we first discuss the experimental setup in Section 5.1. Then, we present the time efficiency experiments and space efficiency experiments in Section 5.2 and Section 5.3, respectively, using the default Epanechnikov kernel. Lastly, we conduct a case study to compare traditional KDV and product KDV using the London crime dataset in Section 5.4. Experiments with other kernels are presented in Section 8.2. Some additional experiments can also be found in Section 3 of [49].

5.1 Experimental Setup

To conduct the efficiency experiments, we compare four existing methods, namely SCAN, RQS$_{\text{kd}}$, RQS$_{\text{range}}$, and SLAM, with our methods, namely MASS$_{\text{CR}}$ (see Section 3) and MASS$_{\text{OPT}}$ (see Section 4). Note that SCAN is the naïve method that does not provide any optimization approach for computing the product kernel density function $\mathcal{D}_P(\mathbf{q})$ (see Equation 3). Both RQS$_{\text{kd}}$ and RQS$_{\text{range}}$, which adopt the kd-tree [12] and the range-tree [21], respectively, belong to the range-query-based approach for efficiently evaluating $\mathcal{D}_P(\mathbf{q})$. SLAM is the state-of-the-art KDV method [18] that is extended to generate the product KDV (see Section 2.2).

[3]Since the New York taxi dataset is extremely large, we only retain those data points from 1st January 2014 to 31st March 2014 that are within the Manhattan region.

Table 3: Datasets.

Name	n	Category
London [2]	4,345,904	Crime events
San Francisco [5]	6,782,645	311 calls
Chicago [1]	8,059,584	Crime events
Manhattan [3][3]	37,459,981	Taxi pickup locations

Table 3 summarizes four large-scale location datasets that are used in our experiments. These datasets belong to three categories, which are (1) crime events, (2) 311 calls, and (3) taxi pickup locations. By default, we set the resolution size to be 1280×960 and utilize the Scott's rule [39] to choose the bandwidth parameter along the x-axis b_x and the bandwidth parameter along the y-axis b_y. All methods were implemented in C++ and executed on an Intel i7 2.4 GHz PC with 16 GB memory. We measure the response time (seconds) and the memory space consumption (MB) of each method and do not report results for methods that take longer than 86,400 seconds or consume more than 16GB of RAM.

5.2 Time Efficiency Experiments

In this section, we conduct the following four experiments to test the time efficiency of different methods using the Epanechnikov kernel. As a remark, since the space consumption of $\text{RQS}_{\text{range}}$ exceeds 16GB for the Manhattan dataset, we omit all results of $\text{RQS}_{\text{range}}$ in this dataset. Due to space limitations, we only adopt two largest datasets, Chicago and Manhattan, to conduct the experiments for varying the bandwidth parameters b_x and b_y, while additional experiments for the London and San Francisco datasets are provided in Section 3.1 of [49].

Varying the resolution size. In this experiment, we choose 320×240, 640×480, 1280×960, and 2560×1920 as the resolution sizes for testing the response time of each method. Figure 8 shows the results of all methods. Observe that the larger the resolution size, the larger the response time of each method. Since the time complexity of MASS_{CR} is lower than the ones of the existing methods, this method can achieve the speedups of 6.43x to 24.06x compared with them. Furthermore, MASS_{OPT} (with the optimal time complexity) can further achieve 1.68x to 2.06x speedups compared with MASS_{CR}.

Varying the dataset size. We proceed to investigate how the dataset size affects the response time of each method. To conduct this experiment, we first choose four sampling ratios, which are 0.25, 0.5, 0.75, and 1 (without sampling), and then randomly sample the original dataset with these ratios to obtain the corresponding reduced datasets. Figure 9 shows the response times of all methods with respect to different reduced datasets. Since the time complexities of other methods (e.g., SLAM and SCAN) depend on the multiplication of n with other parameters of the resolution size linearly (e.g., Xn/Yn for SLAM and XYn for SCAN), the response times of these existing methods are sensitive to the parameter n. In contrast, the time complexities of our methods, MASS_{CR} and MASS_{OPT}, either depend on $n \log XY$ or n solely. Therefore, our methods can achieve 3.08x to 47.57x speedups compared with the existing methods in all datasets. In particular, the optimal method, MASS_{OPT}, can achieve 1.79x to 2.04x speedups compared with MASS_{CR}.

Varying the bandwidth parameter b_x. Here, we examine how b_x affects the response time of each method. To conduct this experiment, we first fix b_y to be the default one and then vary b_x by multiplying its default value with different ratios, including 0.25, 0.5, 1 (default one), 2, and 4. Figures 10a and b show the results of all methods. Observe that the response times of RQS_{kd} and $\text{RQS}_{\text{range}}$ are proportional to b_x. The main reason is that the larger the value of b_x, the larger the search space (see Figure 2b), which indicates that RQS_{kd} and $\text{RQS}_{\text{range}}$ need to process more data points. Unlike these methods, SCAN, SLAM, MASS_{CR}, and MASS_{OPT} do not depend on b_x (e.g., MASS still has the size $(2X-1) \times (2Y-1)$ (see Lemma 1) no matter which b_x we choose.). Therefore, the response times of these methods are insensitive to b_x. With the lower time complexities of MASS_{CR} and MASS_{OPT}, MASS_{CR} achieves 11.64x to 24.06x speedups over the existing methods. In addition, MASS_{OPT} further achieves 1.85x to 2.07x speedups compared with MASS_{CR}.

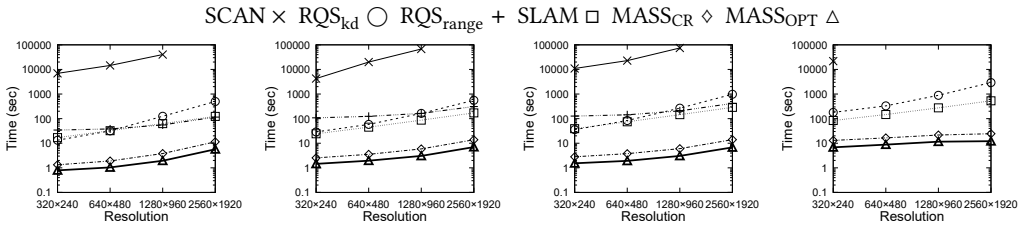
Varying the bandwidth parameter b_y. We further investigate how b_y affects the response time of each method by multiplying its default value with different ratios, including 0.25, 0.5, 1 (default one), 2, and 4, while we fix b_x to be the default one. Figures 10c and d show the results of all methods. Unlike Figures 10a and b, the response time of SLAM is sensitive to b_y since a large b_y results in a large envelope (see Figure 3a), indicating that this method needs to process more data points. Observe that the trends of response times of other methods are similar to the ones of varying b_x (based on the similar reasons in Figures 10a and b). Note that our best method, MASS_{OPT}, achieves 19.6x to 135.31x speedups compared with the existing methods.

5.3 Space Efficiency Experiments

In this section, we examine the space consumption of each method by conducting the following experiments. As the space consumption of $\text{RQS}_{\text{range}}$ is more than 16GB for the Manhattan dataset, we omit the results of $\text{RQS}_{\text{range}}$ in this dataset. Moreover, we also omit some results of SCAN as the response times are more than 86,400 seconds.

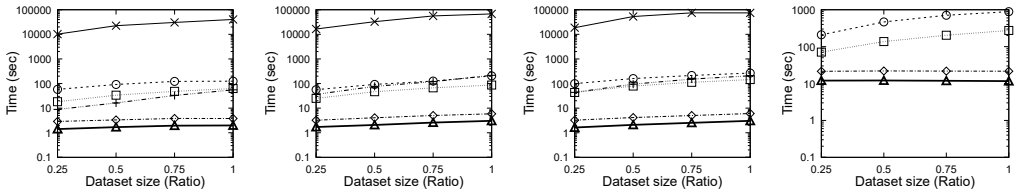
Varying the resolution size. To conduct this experiment, we choose four resolution sizes, which are 320×240, 640×480, 1280×960, and 2560×1920, for testing the memory spaces of all methods. Observe from Figure 11 that the larger the resolution sizes, the larger the memory spaces of our methods, MASS_{CR} and MASS_{OPT}. The main reason is that these two methods also need to maintain the MASS structure (see Figure 4) and several prefix matrices with different polynomial terms (see Figure 6). However, note that the space complexities of MASS_{CR} and MASS_{OPT} are also $O(XY+n)$ (see Theorem 2 and Theorem 4), which are the same as the lower bound space complexity of the product KDV (i.e., $\Omega(XY+n)$, as every algorithm needs to store $X \times Y$ pixels and n data points). We can notice that the memory space overheads of these two methods are not very high compared with existing methods. As a remark, since $\text{RQS}_{\text{range}}$ needs to maintain multiple tree structures, this method consumes the highest space consumptions compared with all methods.

Varying the dataset size. We proceed to investigate how the dataset size affects the memory space consumptions of all methods. In this experiment, we first sample each dataset with four ratios, namely 0.25, 0.5, 0.75, and 1 (the original one), and then measure the memory spaces of all methods for each reduced dataset. Observe from Figure 12 that the larger the dataset sizes, the larger the memory spaces of all methods. With the optimal space complexities of MASS_{CR} and MASS_{OPT}, our methods only have slightly space overheads compared with existing methods. Moreover, our



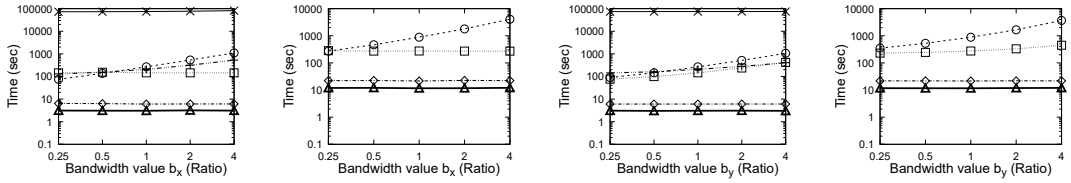
(a) London (b) San Francisco (c) Chicago (d) Manhattan

Figure 8: Response times for computing the product KDV, varying the resolution size.



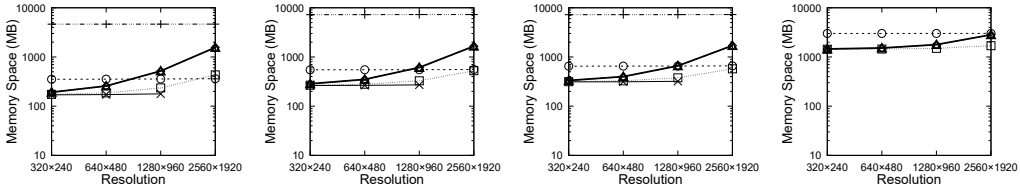
(a) London (b) San Francisco (c) Chicago (d) Manhattan

Figure 9: Response times for computing the product KDV, varying the dataset size.



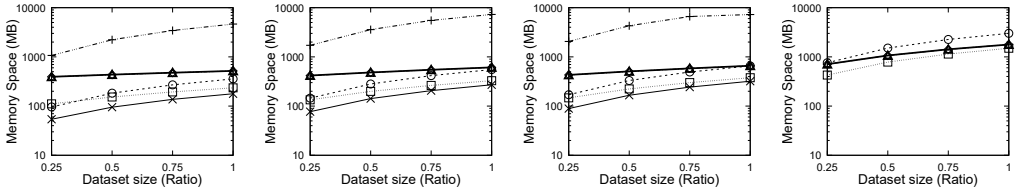
(a) Chicago (varying b_x) (b) Manhattan (varying b_x) (c) Chicago (varying b_y) (d) Manhattan (varying b_y)

Figure 10: Response times for computing the product KDV in the Chicago (a and c) and Manhattan (b and d) datasets, varying the bandwidth parameters, b_x (a and b) and b_y (c and d).



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

Figure 11: Memory space consumption (MB) for computing the product KDV, varying the resolution size.



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

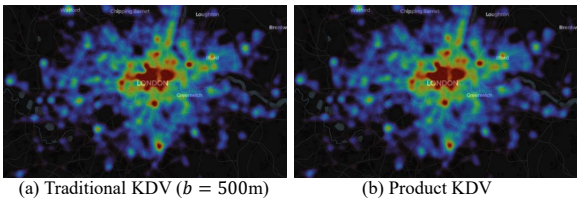
Figure 12: Memory space consumption (MB) for computing the product KDV, varying the dataset size.

methods only need to maintain the MASS structure (and its prefix matrices), which are not very sensitive to (or are more scalable with respect to) the dataset size compared with the existing methods (e.g., RQS$_{kd}$ and RQS$_{range}$).

5.4 Case Study

In this section, we further perform the case study for comparing the visual quality and the efficiency of using traditional KDV and product KDV. In Figure 13, observe that product KDV achieves similar visual quality compared with the traditional KDV.

However, as shown in Figure 14, product KDV (based on MASS$_{OPT}$) can be much faster than traditional KDV (based on the state-of-the-art method, SLAM). Observe that product KDV can still achieve the real-time performance (< 0.5 seconds) even though we choose the resolution size to be 320×240. This result indicates that domain experts can perform exploratory operations (e.g., zoom-in, zoom-out, and panning) for smoothly generating interactive heatmaps. Unfortunately, traditional KDV cannot achieve this goal even though we choose the resolution size to be 80×60.



(a) Traditional KDV ($b = 500$m) (b) Product KDV ($b_x = 493$m and $b_y = 493$m)

Figure 13: Generating the traditional KDV and the product KDV for the London crime dataset (with 4.346 million data points) using the 320×240 resolution size.

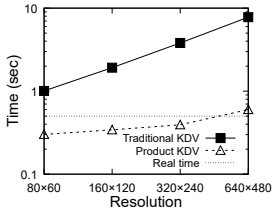


Figure 14: Response times for generating the traditional KDV (with SLAM) and the product KDV (with MASS_{OPT}), varying the resolution size.

6 Related Work

Although efficient algorithms of KDV has been extensively studied in the literature, a lack of research studies, to the best of our knowledge, has been proposed for improving the efficiency of generating product KDV. Here, we review four camps of research studies in KDV (mostly related to this work) and point out the weakness for using (or extending) them for handling the product KDV.

Indexing approach. Recall from Figure 2 that only those data points that are within the search space can contribute to the kernel density function. Therefore, KDV can be regarded as solving multiple range queries. Note that many indexing structures, including kd-tree [12] and range-tree [21], can be used for solving range queries, and thus can be adopted for generating KDV. Although we have extended these indexing structures, RQS_{kd} and $\text{RQS}_{\text{range}}$, for solving the product KDV problem (see Section 5.1), the time efficiency results of all these solutions are still significantly inferior compared with MASS_{CR} and MASS_{OPT} (see Figures 8 to 10). The main reason is that RQS_{kd} and $\text{RQS}_{\text{range}}$ are unable to reduce the time complexity of solving the product KDV problem (imagine that the bandwidth values, b_x and b_y, are very large, these methods need to access all data points for each pixel.). Furthermore, each of these methods also needs to construct a tree structure for each location dataset, which, therefore, suffers from relatively higher space consumption compared with our methods, especially for $\text{RQS}_{\text{range}}$.

Data sampling approach. Many data sampling methods [37, 38, 45–47] have been proposed for improving the efficiency of generating KDV. Although these methods can reduce the time complexity for generating approximate KDV with a non-trivial probablistic approximation guarantees using the kernel density function $D_P^T(\mathbf{q})$ (see Equation 1), it is still unknown whether these data sampling methods can be extended to support the product kernel density function $D_P(\mathbf{q})$ (see Equation 2) with those theoretical results. Recently, Zhong et al. [48] propose the data sampling (or compression) method for supporting the more complex variant of KDV, called spatiotemporal KDV. However, this method cannot reduce the time complexity for supporting this operation. Unlike these existing methods, MASS_{OPT} is the first method that can achieve the optimal time complexity for generating an exact product KDV.

Function approximation approach. Some researchers propose to utilize some simple functions, e.g., linear function [17, 19] and quadratic function [13], to approximate the more complicated kernel functions in order to boost the efficiency of computing $D_P(\mathbf{q})$. However, these function approximation methods cannot achieve an exact solution for generating KDV. Even worse, they cannot reduce the worst-case time complexity. Therefore, we do not modify these methods for generating product KDV.

Computational sharing approach. Recently, Chan et al. [14, 18] propose two computational sharing methods to reduce the time complexity for generating exact KDV (using $D_P(\mathbf{q})$). Among these two methods, the sweep-line based method, SLAM [18], is the state-of-the-art one that has the lowest time complexity. Although we can extend SLAM to generate product KDV with the time complexity of $O(Y(X+n))/O(X(Y+n))$ (see Section 2.2 and Table 1), this method is still inferior compared with MASS_{OPT} (with the optimal time complexity $O(XY+n)$). Hence, in practice, MASS_{OPT} can achieve significant improvement compared with SLAM (see Section 5.2).

7 Conclusion

In this paper, we study the problem of Product Kernel Density Visualization (product KDV), which has been widely used in crime science, traffic science, epidemiology, and urban planning. Like traditional KDV (based on Equation 1), product KDV is computationally expensive, which is not scalable to large-scale location datasets and large resolution sizes. However, unlike traditional KDV, a lack of research studies, to the best of our knowledge, has been proposed for handling the efficiency issues of product KDV. To overcome this issue, we first propose the new indexing structure, called MAtrix of Search Space (MASS) and then develop two efficient methods, namely MASS_{CR} (the basic one) and MASS_{OPT} (the advanced one), which can reduce the time complexities for generating product KDV. Note that MASS_{OPT} further achieves $O(XY+n)$ time, which reaches the lower bound time complexity of generating product KDV (i.e., $\Omega(XY+n)$), indicating that this method is optimal. Our experimental results (based on four large-scale location datasets, with up to 37.4 million data points) verify that our methods can achieve 3.08x to 135.31x speedups compared with existing methods.

In the future, we will investigate whether some data sampling methods (e.g., [45, 47, 48]) can be combined with our methods to further reduce the time complexities and achieve non-trivial approximation guarantees for supporting product KDV. Furthermore, we will also extend our methods to improve the efficiency of other GIS tools, e.g., DBSCAN [22, 23] and spatiotemporal KDV [15, 16, 48].

Acknowledgments

This work was supported by the Natural Science Foundation of China under grants 23IAA00610, 62572326, 62372308, and 42471496, Scientific Foundation for Youth Scholars of Shenzhen University, the Science and Technology Development Fund Macau SAR (0003/2023/RIC, 0052/2023/RIA1, 0011/2025/RIC, 001/2024/SKL for SKL-IOTSC), and the Center for Scientific Research and Development in Higher Education Institutes, MOE (2024HT013). This work was performed in part at SICC which is supported by SKL-IOTSC, University of Macau.

References

[1] [n. d.]. Chicago Data Catalog. https://catalog.data.gov/dataset/crimes-2001-to-present.
[2] [n. d.]. DATA.POLICE.UK. https://data.police.uk/data/.
[3] [n. d.]. NYC Yellow Taxi Trip Data. https://data.cityofnewyork.us/Transportation/2014-Yellow-Taxi-Trip-Data/gkne-dk5s.
[4] [n. d.]. QGIS. https://docs.qgis.org/2.18/en/docs/user_manual/plugins/plugins_heatmap.html.
[5] [n. d.]. San Francisco Open Data. https://data.sfgov.org/City-Infrastructure/311-Cases/vw6y-z8j6.
[6] 2026. ArcGIS. https://www.arcgis.com/index.html.
[7] 2026. DECK.GL. https://deck.gl/.
[8] 2026. Scipy. https://scipy.org/.
[9] 2026. seaborn: statistical data visualization. https://seaborn.pydata.org/.
[10] 2026. statsmodels. https://www.statsmodels.org/stable/index.html.
[11] Adrian Baddeley, Ege Rubak, and Rolf Turner. 2015. *Spatial Point Patterns: Methodology and Applications with R.* Chapman and Hall/CRC Press, London. https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/9781482210200/
[12] Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (1975), 509–517.
[13] 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
[14] Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SAFE: A Share-and-Aggregate Bandwidth Exploration Framework for Kernel Density Visualization. *Proc. VLDB Endow.* 15, 3 (2022), 513–526.
[15] Tsz Nam Chan, Pak Lon Ip, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SWS: A Complexity-Optimized Solution for Spatial-Temporal Kernel Density Visualization. *Proc. VLDB Endow.* 15, 4 (2022), 814–827.
[16] Tsz Nam Chan, Pak Lon Ip, Bojian Zhu, Leong Hou U, Dingming Wu, Jianliang Xu, and Christian S. Jensen. 2025. Large-scale Spatiotemporal Kernel Density Visualization. In *ICDE*. 99–113. https://doi.org/10.1109/ICDE65448.2025.00015
[17] Tsz Nam Chan, Leong Hou U, Reynold Cheng, Man Lung Yiu, and Shivansh Mittal. 2022. Efficient Algorithms for Kernel Aggregation Queries. *IEEE Trans. Knowl. Data Eng.* 34, 6 (2022), 2726–2739. https://doi.org/10.1109/TKDE.2020.3018376
[18] Tsz Nam Chan, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. SLAM: Efficient Sweep Line Algorithms for Kernel Density Visualization. In *SIGMOD*. ACM, 2120–2134. https://doi.org/10.1145/3514221.3517823
[19] 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
[20] Heather Coatsworth, Catherine A. Lippi, Chalmers Vasquez, Jasmine B. Ayers, Caroline J. Stephenson, Christy Waits, Mary Florez, André B.B. Wilke, Isik Unlu, Johana Medina, Sadie J. Ryan, John A. Lednicky, John C. Beier, William Petrie, and Rhoel R. Dinglasan. 2022. A molecular surveillance-guided vector control response to concurrent dengue and West Nile virus outbreaks in a COVID-19 hotspot of Florida. *The Lancet Regional Health - Americas* 11 (2022), 100231. https://doi.org/10.1016/j.lana.2022.100231
[21] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. 2008. *Computational geometry: algorithms and applications, 3rd Edition.* Springer. https://www.worldcat.org/oclc/227584184
[22] 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 *SIGKDD*. AAAI Press, 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php
[23] Junhao Gan and Yufei Tao. 2017. On the Hardness and Approximation of Euclidean DBSCAN. *ACM Trans. Database Syst.* 42, 3 (2017), 14:1–14:45. https://doi.org/10.1145/3083897
[24] Eduardo García-Portugués and Andrea Meilán-Vila. 2025. Kernel density estimation with polyspherical data and its applications. *J. Amer. Statist. Assoc.* just-accepted (2025), 1–25.
[25] Margherita Gerolimetto and Stefano Magrini. 2023. Distribution dynamics: a spatial perspective. *Spatial Economic Analysis* 18, 1 (2023), 64–88.
[26] Somaye Ghezelbash, Reza Ghezelbash, and Mohsen Kalantari. 2024. Developing a spatio-temporal interactions model for car crashes using a novel data-driven AHP-TOPSIS. *Applied Geography* 162 (2024), 103151. https://doi.org/10.1016/j.apgeog.2023.103151
[27] Michael Govorov, Giedrė Beconytė, and Gennady Gienko. 2025. Exploration-Based Statistical Learning for Selecting Kernel Density Estimates of Spatial Point Patterns. *Transactions in GIS* 29, 2 (2025), e70051. https://doi.org/10.1111/tgis.70051 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.70051 e70051 8350596.
[28] Timothy Hart and Paul Zandbergen. 2014. 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 (2014), 305–323. https://doi.org/10.3390/s80603601
[29] Amanda S. Hering and Sean Bair. 2014. Characterizing spatial and chronological target selection of serial offenders. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 63, 1 (2014), 123–140. http://www.jstor.org/stable/24771835
[30] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. 1997. Range Queries in OLAP Data Cubes. In *SIGMOD*. ACM, 73–88. https://doi.org/10.1145/253260.253274
[31] Yu Lan, Isabel Rancu, Melanie H Chitwood, Benjamin Sobkowiak, Kate Nyhan, Hsien-Ho Lin, Chieh-Yin Wu, Barun Mathema, Tyler S Brown, Caroline Colijn, et al. 2025. Integrating genomic and spatial analyses to describe tuberculosis transmission: a scoping review. *The Lancet Microbe* (2025).
[32] Jonas Lukasczyk, Ross Maciejewski, Christoph Garth, and Hans Hagen. 2015. Understanding hotspots: a topological visual analytics approach. In *SIGSPATIAL*. 36:1–36:10. https://doi.org/10.1145/2820783.2820817
[33] Shupeng Lyu, Chen Qian, and Ching-Hung Lee. 2025. Shifting landscape of terrorism: A 50-year spatiotemporal analysis. *Risk Analysis* 45, 8 (2025), 2375–2389. https://doi.org/10.1111/risa.70018 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/risa.70018
[34] Renee J Mitchell, Hunter M Boehme, and Cannon Fulmer. 2025. Experimental evidence shows police leaders may make strategic decisions depending on visuals presented: results from a multi-armed survey experiment. *Journal of Experimental Criminology* (2025), 1–22.
[35] Nuri Park, Juneyoung Park, Yang-Jun Joo, and Mohamed Abdel-Aty. 2025. Micro-level hotspot identification at intersections using traffic conflict analysis. *Accident Analysis & Prevention* 220 (2025), 108167. https://doi.org/10.1016/j.aap.2025.108167
[36] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[37] 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
[38] 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
[39] D. W. Scott. 1992. *Multivariate Density Estimation: Theory, Practice, and Visualization.* Wiley. https://books.google.com.hk/books?id=7crCUS_F2ocC
[40] John RJ Thompson. 2024. Iterative smoothing for change-point regression function estimation. *Journal of Applied Statistics* 51, 16 (2024), 3431–3455.
[41] Brian E. Vestal, Nichole E. Carlson, and Debashis Ghosh. 2021. Filtering spatial point patterns using kernel densities. *Spatial Statistics* 41 (2021), 100487. https://doi.org/10.1016/j.spasta.2020.100487
[42] Hao Wang. 2025. Ranking dynamics of urban mobility. *Journal of Transport Geography* 128 (2025), 104331. https://doi.org/10.1016/j.jtrangeo.2025.104331
[43] Zachary D. Weller. 2018. spTest: An R Package Implementing Nonparametric Tests of Isotropy. *Journal of Statistical Software* 83, 4 (2018), 1–24. https://doi.org/10.18637/jss.v083.i04
[44] Xiping Yang, Zhixiang Fang, Yang Xu, Ling Yin, Junyi Li, and Zhiyuan Zhao. 2021. Revealing temporal stay patterns in human mobility using large-scale mobile phone location data. *Transactions in GIS* 25, 4 (2021), 1927–1948. https://doi.org/10.1111/tgis.12750 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12750
[45] 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.
[46] 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
[47] 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
[48] Yue Zhong, Tsz Nam Chan, Leong Hou U, Dingming Wu, Wei Tu, Ruisheng Wang, and Joshua Zhexue Huang. 2025. A Fast and Accurate Block Compression Solution for Spatiotemporal Kernel Density Visualization. In *SIGKDD*. ACM, 4098–4109.
[49] Yue Zhong, Tsz Nam Chan, Leong Hou U, Dingming Wu, Wei Tu, Ruisheng Wang, and Joshua Zhexue Huang. 2026. MASS: A Complexity-Optimal Solution for Product Kernel Density Visualization. https://github.com/YovelaZ/product_KDV/blob/main/supplementary_material.pdf.

8 Appendix

8.1 Other Kernels

In previous sections, we mainly focus on the Epanechnikov kernel (see Equation 2). Here, we further investigate how to extend MASS (see Section 3 and Section 4) to support other kernel functions in Table 2, including the uniform kernel and the triangular kernel.

Uniform kernel. Consider the uniform kernel in the product kernel density function $\mathcal{D}_P(\mathbf{q})$ (see Equation 3). We have

$$\mathcal{D}_P(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot \begin{cases} \frac{1}{b_x} & \text{if } d(\mathbf{q}_x, \mathbf{p}_x) \le b_x \\ 0 & \text{otherwise} \end{cases} \cdot \begin{cases} \frac{1}{b_y} & \text{if } d(\mathbf{q}_y, \mathbf{p}_y) \le b_y \\ 0 & \text{otherwise} \end{cases}$$

By adopting the concept of Section 3.1, we can represent $\mathcal{D}_P(\mathbf{q})$ by the following expression.

$$\mathcal{D}_P(\mathbf{q}) = \frac{1}{b_x b_y} T^{(0,0)}_{\mathbb{S}(\mathbf{q})} \tag{10}$$

Hence, those techniques in MASS_{CR} and MASS_{OPT} can be directly extended to support the generation of product KDV with this $\mathcal{D}_P(\mathbf{q})$ (see Equation 10).

Triangular kernel. Consider the triangular kernel in the product kernel density function $\mathcal{D}_P(\mathbf{q})$ (see Equation 3). We have

$$\mathcal{D}_P(\mathbf{q}) = \sum_{\mathbf{p}\in P} w \cdot \begin{cases} 1 - \frac{1}{b_x} d(\mathbf{q}_x, \mathbf{p}_x) & \text{if } d(\mathbf{q}_x, \mathbf{p}_x) \le b_x \\ 0 & \text{otherwise} \end{cases} \cdot \begin{cases} 1 - \frac{1}{b_y} d(\mathbf{q}_y, \mathbf{p}_y) & \text{if } d(\mathbf{q}_y, \mathbf{p}_y) \le b_y \\ 0 & \text{otherwise} \end{cases}$$

Based on Equation 4, we can further represent $\mathcal{D}_P(\mathbf{q})$ based on the following expression.

$$\mathcal{D}_P(\mathbf{q}) = \sum_{\mathbf{p}\in \mathbb{S}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x} d(\mathbf{q}_x, \mathbf{p}_x)\right) \cdot \left(1 - \frac{1}{b_y} d(\mathbf{q}_y, \mathbf{p}_y)\right)$$

Unlike $d(\mathbf{q}_x, \mathbf{p}_x)^2$ and $d(\mathbf{q}_y, \mathbf{p}_y)^2$ (i.e., the square value of each Euclidean distance function) in Equation 5, which can be easily expanded based on simple mathematical operations, it is comparatively difficult to expand $d(\mathbf{q}_x, \mathbf{p}_x)$ and $d(\mathbf{q}_y, \mathbf{p}_y)$, respectively.

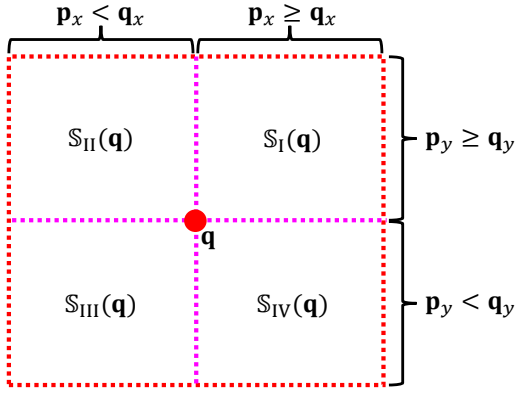


Figure 15: The search space $\mathbb{S}(\mathbf{q})$ can be further divided into four subspaces, which are $\mathbb{S}_{\text{I}}(\mathbf{q})$, $\mathbb{S}_{\text{II}}(\mathbf{q})$, $\mathbb{S}_{\text{III}}(\mathbf{q})$, and $\mathbb{S}_{\text{IV}}(\mathbf{q})$, using the pixel $\mathbf{q}$ as the center.

To handle this issue, observe from Figure 15 that we divide the search space $\mathbb{S}(\mathbf{q})$ into four subspaces, namely $\mathbb{S}_{\text{I}}(\mathbf{q})$, $\mathbb{S}_{\text{II}}(\mathbf{q})$, $\mathbb{S}_{\text{III}}(\mathbf{q})$, and $\mathbb{S}_{\text{IV}}(\mathbf{q})$, where

$$\mathbb{S}_{\text{I}}(\mathbf{q}) = \{\mathbf{p} \in \mathbb{S}(\mathbf{q}) : \mathbf{p}_x \ge \mathbf{q}_x \text{ and } \mathbf{p}_y \ge \mathbf{q}_y\} \tag{11}$$

$$\mathbb{S}_{\text{II}}(\mathbf{q}) = \{\mathbf{p} \in \mathbb{S}(\mathbf{q}) : \mathbf{p}_x < \mathbf{q}_x \text{ and } \mathbf{p}_y \ge \mathbf{q}_y\} \tag{12}$$

$$\mathbb{S}_{\text{III}}(\mathbf{q}) = \{\mathbf{p} \in \mathbb{S}(\mathbf{q}) : \mathbf{p}_x < \mathbf{q}_x \text{ and } \mathbf{p}_y < \mathbf{q}_y\} \tag{13}$$

$$\mathbb{S}_{\text{IV}}(\mathbf{q}) = \{\mathbf{p} \in \mathbb{S}(\mathbf{q}) : \mathbf{p}_x \ge \mathbf{q}_x \text{ and } \mathbf{p}_y < \mathbf{q}_y\} \tag{14}$$

Hence, we can further express $\mathcal{D}_P(\mathbf{q})$ based on the above four subspaces.

$$\begin{aligned} \mathcal{D}_P(\mathbf{q}) = & \sum_{\mathbf{p}\in \mathbb{S}_{\text{I}}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x}(\mathbf{p}_x - \mathbf{q}_x)\right) \cdot \left(1 - \frac{1}{b_y}(\mathbf{p}_y - \mathbf{q}_y)\right) \\ & + \sum_{\mathbf{p}\in \mathbb{S}_{\text{II}}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x}(\mathbf{q}_x - \mathbf{p}_x)\right) \cdot \left(1 - \frac{1}{b_y}(\mathbf{p}_y - \mathbf{q}_y)\right) \\ & + \sum_{\mathbf{p}\in \mathbb{S}_{\text{III}}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x}(\mathbf{q}_x - \mathbf{p}_x)\right) \cdot \left(1 - \frac{1}{b_y}(\mathbf{q}_y - \mathbf{p}_y)\right) \\ & + \sum_{\mathbf{p}\in \mathbb{S}_{\text{IV}}(\mathbf{q})} w \cdot \left(1 - \frac{1}{b_x}(\mathbf{p}_x - \mathbf{q}_x)\right) \cdot \left(1 - \frac{1}{b_y}(\mathbf{q}_y - \mathbf{p}_y)\right) \end{aligned}$$

Based on some simple mathematical derivations, we have

$$\begin{aligned} \mathcal{D}_P(\mathbf{q}) = & \left(1 + \frac{\mathbf{q}_x}{b_x}\right)\left(1 + \frac{\mathbf{q}_y}{b_y}\right) T^{(0,0)}_{\mathbb{S}_{\text{I}}(\mathbf{q})} - \frac{1}{b_x}\left(1 + \frac{\mathbf{q}_y}{b_y}\right) T^{(1,0)}_{\mathbb{S}_{\text{I}}(\mathbf{q})} \\ & - \frac{1}{b_y}\left(1 + \frac{\mathbf{q}_x}{b_x}\right) T^{(0,1)}_{\mathbb{S}_{\text{I}}(\mathbf{q})} + \frac{1}{b_x b_y} T^{(1,1)}_{\mathbb{S}_{\text{I}}(\mathbf{q})} \\ & + \left(1 - \frac{\mathbf{q}_x}{b_x}\right)\left(1 + \frac{\mathbf{q}_y}{b_y}\right) T^{(0,0)}_{\mathbb{S}_{\text{II}}(\mathbf{q})} + \frac{1}{b_x}\left(1 + \frac{\mathbf{q}_y}{b_y}\right) T^{(1,0)}_{\mathbb{S}_{\text{II}}(\mathbf{q})} \\ & - \frac{1}{b_y}\left(1 - \frac{\mathbf{q}_x}{b_x}\right) T^{(0,1)}_{\mathbb{S}_{\text{II}}(\mathbf{q})} - \frac{1}{b_x b_y} T^{(1,1)}_{\mathbb{S}_{\text{II}}(\mathbf{q})} \\ & + \left(1 - \frac{\mathbf{q}_x}{b_x}\right)\left(1 - \frac{\mathbf{q}_y}{b_y}\right) T^{(0,0)}_{\mathbb{S}_{\text{III}}(\mathbf{q})} + \frac{1}{b_x}\left(1 - \frac{\mathbf{q}_y}{b_y}\right) T^{(1,0)}_{\mathbb{S}_{\text{III}}(\mathbf{q})} \\ & + \frac{1}{b_y}\left(1 - \frac{\mathbf{q}_x}{b_x}\right) T^{(0,1)}_{\mathbb{S}_{\text{III}}(\mathbf{q})} + \frac{1}{b_x b_y} T^{(1,1)}_{\mathbb{S}_{\text{III}}(\mathbf{q})} \\ & + \left(1 + \frac{\mathbf{q}_x}{b_x}\right)\left(1 - \frac{\mathbf{q}_y}{b_y}\right) T^{(0,0)}_{\mathbb{S}_{\text{IV}}(\mathbf{q})} - \frac{1}{b_x}\left(1 - \frac{\mathbf{q}_y}{b_y}\right) T^{(1,0)}_{\mathbb{S}_{\text{IV}}(\mathbf{q})} \\ & + \frac{1}{b_y}\left(1 + \frac{\mathbf{q}_x}{b_x}\right) T^{(0,1)}_{\mathbb{S}_{\text{IV}}(\mathbf{q})} - \frac{1}{b_x b_y} T^{(1,1)}_{\mathbb{S}_{\text{IV}}(\mathbf{q})} \end{aligned}$$

where $T^{(i,j)}_{\mathbb{S}_r(\mathbf{q})}$ denotes the polynomial term (similar to Equation 6), where $0 \le i \le 1$ and $0 \le j \le 1$, that depends on the subspace $\mathbb{S}_r$ (where r can be I, II, III, or IV).

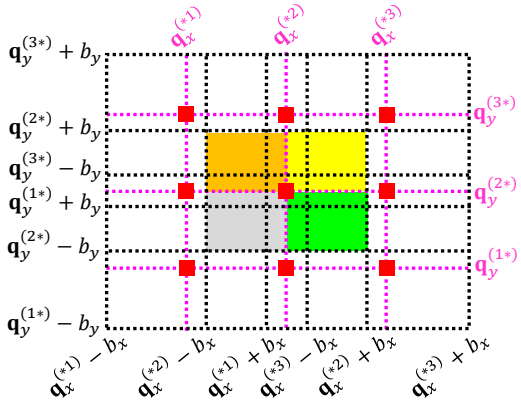
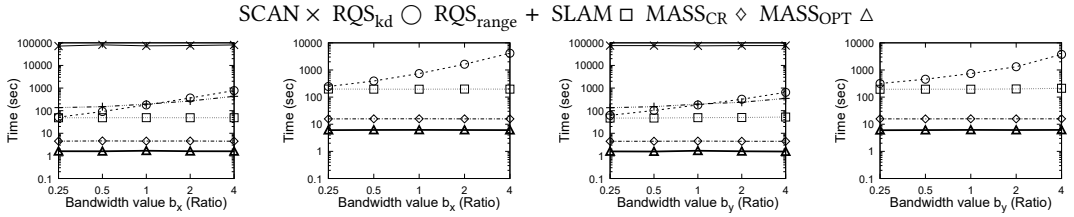


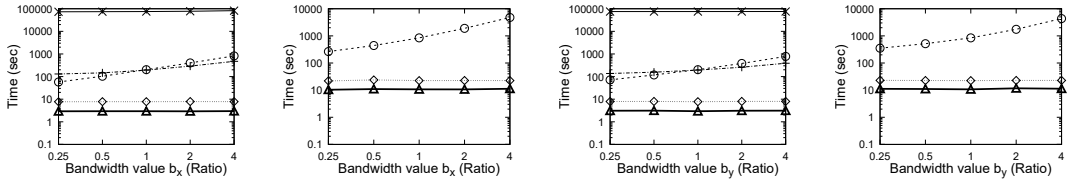
Figure 16: Modification of MASS, by adding X dotted lines in the x-axis and Y dotted lines in the y-axis, for supporting the triangular kernel, where the yellow, orange, grey, and green regions denote the subspaces $\mathbb{S}_{\text{I}}(\mathbf{q}_{22})$, $\mathbb{S}_{\text{II}}(\mathbf{q}_{22})$, $\mathbb{S}_{\text{III}}(\mathbf{q}_{22})$, and $\mathbb{S}_{\text{IV}}(\mathbf{q}_{22})$, respectively.

In order to efficiently evaluate each $T^{(i,j)}_{\mathbb{S}_r(\mathbf{q})}$, we need to modify the MASS structure by adding X dotted lines in the x-axis and Y



(a) Chicago (varying b_x) (b) Manhattan (varying b_x) (c) Chicago (varying b_y) (d) Manhattan (varying b_y)

Figure 17: Response times for computing the product KDV with the uniform kernel function in the Chicago and Manhattan datasets, varying the bandwidth parameters, b_x (a and b) and b_y (c and d).



(a) Chicago (varying b_x) (b) Manhattan (varying b_x) (c) Chicago (varying b_y) (d) Manhattan (varying b_y)

Figure 18: Response times for computing the product KDV with the triangular kernel function in the Chicago and Manhattan datasets, varying the bandwidth parameters, b_x (a and b) and b_y (c and d).

dotted lines in the y-axis (based on the concept of search space division in Figure 15). Hence, this new MASS structure contains $(3X-1)\times(3Y-1)$ entries. Note that all techniques from Section 3.3 to Section 3.5 can be extended to compute each $T^{(i,j)}_{\mathbb{S}_r(\mathbf{q})}$. The main difference is that we only need to handle four rectangular regions (instead of one rectangular region) for computing $\mathcal{D}_P(\mathbf{q})$ (e.g., handle the yellow, orange, grey, and green regions for computing $\mathcal{D}_P(\mathbf{q}_{22})$ with the pixel $\mathbf{q}_{22}$ in Figure 16). Therefore, we conclude that MASS$_{CR}$ still takes $O((XY+n)\log XY)$ time and $O(XY+n)$ space for generating a product KDV using the triangular kernel function, i.e., Theorem 1 and Theorem 2 hold, respectively. Furthermore, since all those pink dashed lines on the x-axis also exhibit the sequence $\{\mathbf{q}_x^{(*1)}, \mathbf{q}_x^{(*2)}, ..., \mathbf{q}_x^{(*X)}\}$ and the difference between any two consecutive pixels along the x-axis must have the same value δ_x, we conclude that we can use $O(1)$ time to count the number of pink dashed lines that are passed through by any data point or pixel. Hence, we can easily extend the interval decomposition approach (see Section 4.1), and thus MASS$_{OPT}$ (see Section 4.2), for supporting the triangular kernel with the same time complexity (i.e., $O(XY+n)$ time in Theorem 3) and the same space complexity (i.e., $O(XY+n)$ space in Theorem 4).

8.2 Experiments with Other Kernels

In this section, we proceed to investigate the time efficiency of computing the product KDV with other kernels, including the uniform kernel and the triangular kernel (see Table 2). To conduct the experiments, we follow the same settings in Section 5.2 for varying the bandwidth parameter b_x and the bandwidth parameter b_y for testing the time efficiency. Additional time efficiency experiments with other kernels can be found in Section 3.2 of [49].

Uniform kernel. Figures 17a and b show the results of all methods by varying b_x. Note that we can seamlessly reuse all methods (without significant modifications and processing overheads) to support the uniform kernel function. The trends of response times in Figures 17a and b are similar to the ones of Figures 10a and b. However, since MASS$_{CR}$ and MASS$_{OPT}$ only need to maintain a single term $T^{(0,0)}_{P(e)}$ (based on Equation 10) for each entry e of the MASS structure (instead of nine terms for the Epanechnikov kernel in Figure 5), the response times of these two methods are smaller than the ones of the Epanechnikov kernel. With the lower time complexities of MASS$_{CR}$ and MASS$_{OPT}$, we can observe that MASS$_{CR}$ can achieve 10.62x to 12.58x speedups compared with all existing methods. Moreover, MASS$_{OPT}$ further achieves 2.57x to 2.87x speedups compared with MASS$_{CR}$.

Figures 17c and d further show the results of all methods by varying b_y. Based on the similar reasons for the experiment of varying b_x, we can observe that (1) the trends of response times in Figures 17c and d are also similar to Figures 10c and d and (2) MASS$_{CR}$ and MASS$_{OPT}$ can achieve 10.63x to 34.26x speedups compared with all existing methods.

Triangular kernel. Figures 18a and b show the results of all methods by varying b_x. Since SLAM cannot support the triangular kernel, we do not include this method for comparison. Furthermore, the space consumption of RQS$_{range}$ is more than 16GB in the Manhattan dataset. Therefore, the results of this method are also omitted in this dataset. Like the results of the Epanechnikov kernel (and the uniform kernel), the trends of response times in Figures 18a and b are similar to the ones of Figures 10a and b (Figures 17a and b). Although MASS$_{CR}$ and MASS$_{OPT}$ need to maintain a larger MASS structure for the triangular kernel (see Figure 16), they still achieve 7.46x to 424.2x speedups over existing methods due to their lower time complexities.

Figures 18c and d further show the response times of all methods by varying b_y. Based on the similar reasons in the experiment of varying b_x, we note that the trends of response times in Figures 18c and d are similar to the ones of Figures 10c and d (Epanechnikov kernel) and Figures 17c and d (uniform kernel). In addition, MASS$_{CR}$ and MASS$_{OPT}$ can still achieve 9.06x to 383.64x speedups compared with the existing methods.