2025 届研究生硕士学位论文

分 类 号：_____ 学校代码：10269

密　　级：_____ 学　　号：51255902035

華東師範大學

East China Normal University

硕士学位论文

MASTER'S DISSERTATION

论文题目：基于核密度估计的时空路网数据可视化研究

院　　系：软件工程学院

专　　业：软件工程

研 究 方 向：软件科学与技术

学位申请人：邵煜

指 导 教 师：林学民教授

2025 年 4 月

Dissertation for Master's Degree in 2025

University code: 10269

Student ID: 51255902035

East China Normal University

Title: Research on Spatio-Temporal Road Network Data Visualization Based on Kernel Density Estimation

Department/School: Software Engineering Institude

Major: Software Engineering

Research Direction: Software Science and Technology

Candidate: Yu Shao

Supervisor: Prof. Xuemin Lin

May, 2025

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《基于核密度估计的时空路网数据可视化研究》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名 **日期** 年 月 日

华东师范大学学位论文著作权使用声明

《基于核密度估计的时空路网数据可视化研究》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的著作权归本人所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（ ）1.经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文*，于 年 月 日解密，解密后适用上述授权。

（ ）2.不保密，适用上述授权。

导师签名 **作者签名**

日期 年 月 日

*“涉密”学位论文应是已经华东师范大学学位管理办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

硕士学位论文答辩委员会成员名单

姓名	职称	单位	备注
王凯	副教授	上海交通大学	主席
杨世宇	教授	广州大学	
韩莉	副教授	华东师范大学	

摘　要

核密度估计法（Kernel Density Estimation, KDE）已成为一种常见的数据可视化手段，已被广泛地应用于金融风险预测、犯罪聚集分析及交通监控等多个领域中。核密度估计法能够将离散的数据点拟合成连续的分布，实现对任意位置的值预测，并可在后续通过归一化映射到有限范围内实现着色并可视化。

然而大多数现有的工作仅考虑了空间数据，并使用多维空间上的欧式距离作为距离度量，具有一定的局限性。在本论文中，我们提出了一种新的计算模型——时间网络核密度估计（Temporal Network Kernel Density Estimation，TN-KDE），将传统的核密带度估计法扩展到有时间数据的路网图中。在时间网络核密度估计中，每个数据点同时包含坐标信息和时间信息，并且使用最短路径距离作为距离度量，使其更加符合路网上的实际情况。

为了解决这一问题，本论文设计了一种全新的算法：区间森林法（Range Forest Solution，RFS），用于高效地在时空路网图中计算核密度值。为了进一步支持区间森林法的插入操作，本论文还设计了一种动态算法：动态区间森林法（Dynamic Range Forest Solution，DRFS）。此外，我们还提出了一种叫做线段点共享（Lixel Sharing，LS）的优化技术，可在相邻的若干线段点中共享大量相似的计算。实验结果表明，精确的 RFS 算法相比于最优算法和基线算法分别有 6 倍和 89 倍的性能提升，近似的 DRFS 算法则可以进一步减少 40% 的时间开销，且核密度误差不超过 5%。

最后，考虑到大量重复的最短路径计算带来的算法瓶颈，本论文还探讨了分布式计算最短路径的可能性，并给出了多个常见的分布式计算平台上的最短路径算法实现。

关键词: 核密度估计法，数据可视化，时空数据，最短路径

ABSTRACT

Kernel Density Estimation (KDE) has become a popular method for visual analysis in various fields, such as financial risk forecasting, crime clustering, and traffic monitoring. KDE can fit discrete data points into a continuous distribution, enabling the prediction of values at any place. Subsequently, through normalization, the values can be mapped to a finite range for coloring and visualization purposes.

However, most existing works only consider planar distance and spatial data. In this thesis, we introduce a new model, called TN-KDE, that applies KDE-based techniques to road networks with temporal data. In TN-KDE, each data point contains coordinate information and time information, using the shortest path distance as the distance metric, which more accurately reflects the actual conditions on road networks.

Specifically, we introduce a novel solution, Range Forest Solution (RFS), which can efficiently compute KDE values on spatiotemporal road networks. To support the insertion operation, we present a dynamic version, called Dynamic Range Forest Solution (DRFS). We also propose an optimization called Lixel Sharing (LS) to share similar KDE values between two adjacent lixels. The experimental results show that the exact RFS method achieves 6 times and 89 times compared to the state-of-the-art and baseline algorithm, respectively. The approximate DRFS method can further reduce time overhead by 40%, while the error does not exceed 5%.

Finally, considering the computational bottleneck of massive shortest path calculation, this thesis also explores the possibility of distributed computation of shortest paths. We provide implementations of shortest path algorithms on several common distributed computing platforms.

Keywords: *Kernel Density Estimation; Data Visualization; Spatio-Temporal Data; Shortest Path*

目录

图目录

表目录

第一章　绪论

1.1　研究背景与概述

核密度估计法（Kernel Density Estimation，KDE）是一种常见的非参数化数据估计方法，广泛应用于统计学、机器学习以及数据分析领域。不同于传统的参数化估计方法需要假设数据服从某种特定的概率分布（例如正态分布），并据此估计分布的参数（如均值和方差），核密度估计法则能直接从数据样本中计算概率密度函数，而无需对数据的分布形式做出任何假设。这种方法提供了一种更加灵活的方式来探索数据的内在分布特点，尤其适用于那些难以用简单的参数模型描述的数据集。非参数化的估计方法可以直接用于将离散的数据点转化为平滑的数据分布，分析数据的分布形式，并通过可视化的方式检测热点 [1, 2]。

核密度估计法的应用场景十分广泛。在金融监测方面，核密度估计法可以将离散的数据点转化为连续分布，并据此估计未来的金融指标和风险 [3–5]；在犯罪聚集分析方面，可以对分析犯罪事件在地理位置上的分布特点，帮助执法机构寻找潜在的罪犯据点或制定更加高效的警力配置策略 [6–8]；在交通事故分析方面，通过对各个时段的交通流进行可视化，可以快速识别不同时间的事故高发区域，为管理部门调配人力提供参考 [9–11]。

核密度估计法通过对有限的离散数据集加权得到平滑的分布来估计其他位置的数据，即核密度。核密度的一般计算公式如下：

$$F(q) = w \cdot \sum_{o_i \in O} K\Big(\frac{dist(q, o_i)}{b}\Big), \tag{1.1}$$

其中，q 是待查询核密度的位置；$F(q)$ 为查询点的核密度；$o_i \in O$ 是已知的数据点；$dist(q, o_i)$ 是查询点 q 到数据点 o_i 的距离，与带宽范围 b 相除后得到归一化的距离；$K(\cdot)$ 是核函数，用于计算权重；w 是放缩因子，用于将最后的结果变换到可视化范围（例如 $0 \sim 255$）。

直观来看，核密度估计法是对数据点进行加权统计的一种方法，其基本思想

是：对于每一个查询点，距离该查询点越近的数据点对最终的核密度估计贡献越大；反之，距离越远的数据点影响越小，当查询点与某个数据点之间的距离超过一定的带宽范围 b 时，该数据点将不再对查询点处的核密度估计产生影响。

这种基于距离的权重分配方式具体是通过核函数来实现的。核函数是一类定义在 $[0,1]$ 上的递减函数，代表不同距离下的权重分配规则，即数据点在其邻域内的影响力分布。常见的核函数包括多项式核函数（例如三角核函数 [12, 13]，Epanechnikov 核函数 [14, 15]）和超越核函数（例如 Gaussian 核函数 [16, 17]，余弦核函数 [18]）。表 1.1 总结了这些常见的核函数以及他们的表达式。

表 1.1　常见的核函数及其表达式

核函数名称	核函数表示
三角核函数 [12, 13]	$1-\frac{1}{b}dist(q,o_i)$
Epanechnikov 核函数 [14, 15]	$1-\frac{1}{b^2}dist(q,o_i)^2$
Gaussian 核函数 [16, 17]	$exp(-\frac{1}{b^2}dist(q,o_i)^2)$
余弦核函数 [18]	$\cos(\frac{1}{b}dist(q,o_i))$

不同的核函数会带来不同的核密度估计结果，因此选择合适的核函数对于准确描述数据分布至关重要。这些核函数的主要特点如下：

- 三角核函数。这是一种简单的核函数，表现为一个线性递减函数。当距离从 0 增加到带宽范围 b 时，权重从 1 线性减少到 0。三角核函数因其简单性和高效性，在许多场景中广泛应用，且这种线性的表达式在很多计算中都非常高效。
- Epanechnikov 核函数。这种核函数表现为二次多项式形式，在带宽范围内具有最优的均方误差性质。Epanechnikov 核函数在接近带宽边界时迅速下降至零，使得核密度的估计更加平滑。相比于三角核函数，Epanechnikov 核函数的表达式较为复杂，但它依然属于多项式核函数，在计算上有许多优化空间。
- Gaussian 核函数。这是最常用的核函数之一，假设数据服从高斯分布。它的特点是权重随距离呈指数衰减，即距离越远，权重下降得越快。Gaussian 核函数在处理连续数据和平滑估计方面表现优异，且更加符合实际情况。

- 余弦核函数。这种核函数使用余弦函数来定义权重，值的变化较为平滑，适用于需要更柔和过渡的应用场景。此外，许多相似度计算都需要用到余弦函数，因此余弦核函数还可用来计算与相似度相关的核密度。

核密度估计中，数据点的权重大小由距离所决定，反映了带宽范围内数据点的综合影响。通常情况下，距离较近的数据点会被赋予更高的权重，从而对核密度估计产生更显著的影响，相对而言，距离较远的数据点会被赋予更低的权重。因此，不同的距离度量方式会产生不同的核密度，甚至可能会影响到整体的数据分布，因此需要根据实际的数据特点选取合适的距离度量方式。

表 1.2　常见的距离度量方式及其表达式

距离度量方式	距离函数表示
欧几里得距离	$d(x,y)=\sqrt{\sum(x_i-y_i)^2}$
曼哈顿距离	$d(x,y)=\sum\lvert x_i-y_i\rvert$
切比雪夫距离	$d(x,y)=\max\lvert x_i-y_i\rvert$
余弦相似度距离	$d(x,y)=\frac{x\cdot y}{\lVert x\rVert\cdot\lVert y\rVert}$

目前，大部分算法均采用欧几里得距离作为距离度量的方法，欧几里得距离是一种直观且广泛应用的距离计算方法，它基于两点之间的直线距离，表明事件的影响随着直线传播。这种距离度量方式简单直接，适用于许多应用场景。然而，欧几里得距离的根号形式在计算中并不方便且十分低效，如果不使用带有二次方形式的 Epanechnikov 核函数，很容易引入计算误差。

相比之下，曼哈顿距离和切比雪夫距离则仅包含加减法和绝对值的运算，在计算形式上更加简单和高效。在某些特定场景下，它们可以提供比欧几里得距离更加灵活实际的计算方法。例如，曼哈顿距离度量数据点之间的城市街区距离，这在某些网络分析或城市规划中具有实际意义；而切比雪夫距离则着眼于最大尺度的变化，适用于需要快速确定最远点距离的应用。

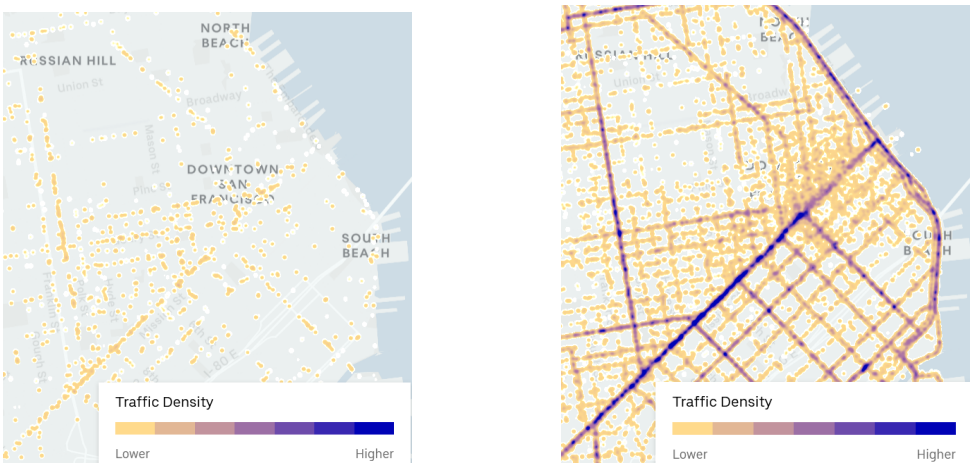
此外，还有一些更加复杂的距离度量方式，例如余弦相似度距离等，常用于计算高维空间中的距离，例如两个文本或图像的高维空间嵌入的距离越近，表明这两个嵌入的相似度越高，即原始文本或图像越接近。

这些距离度量方式及其对应的距离函数如表 1.2所示。这些距离度量方式都可以自由地插入到和密度计算公式中。

1.2 现有工作与挑战

核密度估计法易于理解、实现和可视化，已经成为了许多地理信息系统软件中不可或缺的一部分，例如 ArcGIS [19]，QGIS [20] 和 KDV-Explorer [21] 都支持这一算法。这些工具不仅提供了强大的数据分析能力，还允许用户通过直观的方式理解和展示数据的空间分布特征。然而，这些核密度估计法在以下三种场景中存在局限性：

- 时间维度的聚类。核密度估计法是一种强大的工具，用于分析事件的空间分布，并通过空间距离（如欧几里得距离）来衡量事件的影响和计算核密度。然而，许多现实世界中的事件不仅在空间上相互关联，还与时间密切相关 [9]。例如，交通流量、天气变化、社会动态等都呈现出明显的时间特征。



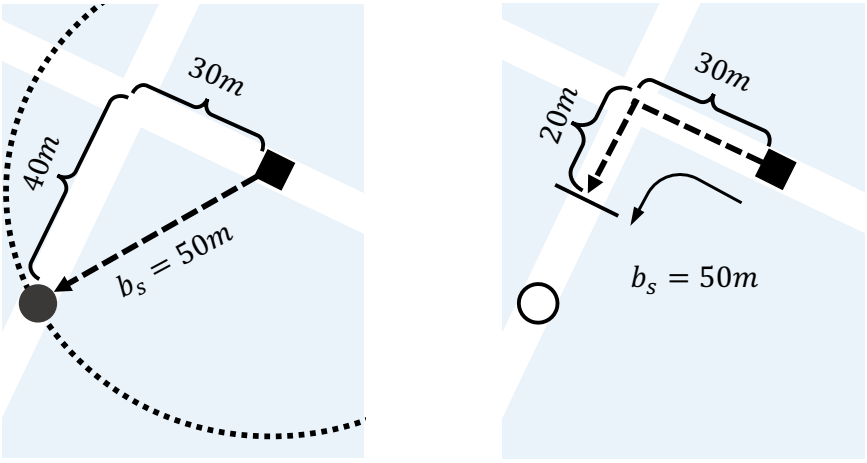
(a) 凌晨 3 点到 6 点的热力图　　(b) 下午 3 点到 6 点的热力图

图 1.1　不同时间窗口下的热力图对比。

在这些情况下，单纯依赖空间距离进行核密度估计可能无法全面捕捉数据的真实特征和变化规律。为了更准确地分析这些复杂现象，我们需要将时间维度也纳入考虑范围。这意味着不仅要关注数据点的地理位置和空间距离，还需评估事件发生的时间间隔及其演变过程。这种方法有助于揭示出数据集在时间维度上的动态变化，以及在时间和空间上不同数据点之间的潜在关联性。

具体来说，用户可能会选择过滤特定时间段内的事件，以揭示数据点在时间维度上的关联 [6]。这种方法不仅可以帮助我们更好地理解事件的时间模式，还能为决策提供有价值的参考信息。例如，Uber Movement 可以展示了特定时间段内的人口流动热图，如图 1.1所示，这是 2020 年第一季度旧金山地区在凌晨 3 点到 6 点以及下午 3 点到 6 点两个不同时间段的交通流动性热图。可以很明显的看到下午 3 点到 6 点的流动性较高，且主干道的密度大于其他街道。这些密度较高的地区会更加拥堵，可以给后续的出租车派单和导航给出一定参考信息。

- 路网图的应用。一些研究发现，在路网图中采用欧几里得距离可能会高估核密度。图 1.2 对比了平面核密度和网络核密度的计算方式（分别采用二维欧几里得距离和最短路径距离）。假设在查询点核数据点在平面上的距离为 50 米，如果我们使用欧几里得距离来计算这两个点之间的距离，那么核密度估计法会认为这两个点非常接近，并且会在它们之间分配较高的权重。然而，实际情况可能由于道路布局的原因，两点间的实际最短路径距离可能为 70 米。这意味着如果使用欧几里得距离进行核密度估计，该数据点的关联性会被高估，甚至会包含不应该存在的数据点。



(a) 平面核密度估计　　(b) 网络核密度估计

图 1.2　采用不同距离度量方式的核密度估计法的差异。平面核密度估计采用欧几里得距离，会低估查询点到数据点的距离，进而高估核密度。

- 多次在线查询。在实际应用中，用户往往需要选取不同的参数，多次提交查询任务来生成一系列核密度估计值，并从中选取最符合需求的结果。参数的选择往往

基于上一次的计算结果，这就要求查询结果是是实时返回的。这种动态调整和实时反馈的需求，使得传统的单次核密度估计方法难以满足现代数据分析的要求。然而，现有的工作主要关注于单次核密度估计的优化，缺少对多次在线查询的优化 [2, 6, 11, 13, 22–24]。

现有的工作在一定程度上已经能够优化核密度估计法，以解决上述问题。为了支持时间维度上的查询，一种常见的方式是根据固定的时间区间（如每小时、每天或每月）来逐个计算核密度 [11]，从而分析事件发生的时间模式及其关联性。这种方法允许我们观察特定时间段内的数据分布特征，并识别出潜在的趋势和规律。然而，这种方法虽然直观易懂，但在处理高分辨率时间序列时可能会面临计算效率的问题。另一种常用方法是在三维时空中划分时空立方体 [7, 9] 并利用时空核函数估计核密度 [6, 25, 26]，其本质是将时间视作另一个维度并进行划分。通过这种方式，不仅可以揭示事件在特定时间窗口内的空间分布，还能捕捉到它们随时间变化的动态特性。

针对路网图上的核密度估计，一个关键改进是使用最短路径距离代替传统的欧几里得距离。[27]，并沿边计算每个位置的核密度值，而不是在欧几里得空间中对每个像素进行计算 [10]。这种方法更加符合实际的道路网络结构，避免了直线距离带来的误差。一些工作还进一步在在每条边上建立列表索引，来减少重复计算，实现更高效的核密度估计 [28]。这些改进通过优化数据结构和计算流程，显著提升了处理大规模路网数据的能力。

为了应对多次及在线查询的需求，传统的工作常采用基于分区的方法（例如 K-d 树 [29–31]、网格化 [8, 9]、快速傅里叶变换 (FFT) [2, 32]、聚类 [33–35] 和分箱 [2, 36, 37] 等），这些方法的核心思想是将整个数据集分割成若干部分，并建立相应的数据结构，以快速检索和整合相邻的数据点。合并计算它们对核密度的贡献 [36]。

1.3 研究贡献与创新

尽管现有的研究在特定问题上进行了一些优化，如时间维度的查询、路网图上的核密度估计以及多次在线查询等，但目前仍缺乏一种单一的方法能够高效地

解决所有这些主要挑战。因此，我们提出了针对时空数据集的时间网络核密度估计（Temporal Network Kernel Density Estimation，TN-KDE），旨在提供一个全面且高效的解决方案，以应对时空数据集中的复杂需求。

首先，我们提出了一种高效的算法，称为区间森林法（Range Forest Solution，RFS），用于高效计算任意时间窗口内的核密度。RFS 采用了基于内存共享的树形索引来同时维护数据点的时间信息和位置信息。这种创新的数据结构能够在高效处理多个时间窗口查询的同时不增加额外的内存开销。

接着，我们还提出了动态区间森林法（Dynamic Range Forest Solution，DRFS），DRFS 扩展了 RFS，通过引入动态结构以支持插入操作。DRFS 还为用户提供了不同等级的量化参数，以达到不同的计算精度。这种灵活的设计允许用户的根据实际需求和限制来动态调整索引的大小，便于用户平衡输出结果的精度和计算时间的开销。

更进一步，我们还注意到相邻查询点的核密度值往往是平滑变化的，且共享了大量相似的计算步骤。针对这一现象，我们设计了一种叫做线段点共享（Lixel Sharing，LS）的技术，允许将相似的计算结果共享到多个查询点上。

最后，我们的框架支持更复杂的核函数，包括指数核函数和余弦核函数，且能够计算精确的核密度。

此外，考虑到 TN-KDE 中会频繁调用最短路径算法的需求，为了提高计算效率和处理大规模数据集的能力，我们还探索了如何将最短路径算法扩展到分布式环境中计算。我们简单介绍了传统图算法是如何移植到分布式环境下，以及相关的底层设计（包括图划分算法和图计算模式），然后给出了常见图计算平台上的分布式最短路径算法的实现，同时对这些代码进行运行时间和扩展性的评估，给出性能对比和选取参考。

本论文的主要贡献和章节安排如下：

- 在第3.1章中，我们正式地定义了空间网络核密度估计（TN-KDE）问题，并介绍了一个基础框架。
- 在第3.2章和第3.3章中，我们分别介绍了两种算法区间森林法（RFS）和动态区间森林法（DRFS），用于高效地计算空间网络核密度，以及所支持的插入操作。

- 在第3.4章中，我们提出了线段点共享（LS）的优化技术，以减少冗余计算并提高效率。同时在第3.5章中，我们分析了许多可以应用于我们框架的非多项式核函数。
- 在第3.6章中，通过对不同规模和类别的真实世界数据集进行测试，展示了我们提出方法的高效性和有效性。
- 在第4.1，4.2，4.3章中，我们介绍了分布式图计算的相关背景，包括图计算的划分算法和图计算的计算模式，并在第4.4章给出了几个常见的计算平台及这些计算平台上的最短路径算法实现。
- 在第4.5章中，我们对这些最短路径算法在不同的计算资源配置下进行测试，主要包括运行时间和扩展性，给出了这些平台的性能对比和选取参考。

第二章　相关工作

2.1　基于划分的方法

核密度估计法的计算过程通常涉及对每一个查询点遍历所有数据点并计算其权重，这一步骤往往非常耗时 [2]，特别是在处理大规模数据集时，逐个计算每个数据点对查询点的影响会导致显著的性能瓶颈。为了解决这一问题，一个常见的做法是对数据点进行聚合，即将位置较为接近的数据点合并成一个聚合集，当某次查询覆盖了聚合集中的所有数据点时，可以整体计算聚合集的对核密度的贡献，从而避免逐个计算带来的时间开销 [36]。

通过这种方法，不仅可以大幅减少计算量，还能有效提升算法的整体效率。不同的划分算法会产生不同的聚合集，这不仅影响到计算效率的提升，还涉及到维护开销的变化。下面将详细介绍几种常见的划分方式及其特点。

- **K-d 树** [29–31]。K-d 树是维护多维空间中的数据点的最常用的数据结构之一，其主要思想是将用一个超平面将数据点沿着某个维度平均分成两半，并在其他维度依次重复执行这一操作。由于每次操作都会将数据点分成两半，最后构成了树状结构，其中每个节点都代表一个超立方体，维护了其中的数据点的信息（特别地，叶节点仅包含一个数据点）。当计算核密度时，如果查询的带宽范围覆盖了整个超立方体或完全不相交，即可直接返回结果，否则继续递归，在更小的子空间中查询。然而，K-d 树仅在低维度（大约 10 维）下有效。对于更高维度，由于分布的稀疏性，K-d 树的效率急剧下降，此时另一个变种算法球树更为合适 [38]。
- **网格化** [8, 9]。不同于 K-d 树这种动态的划分方式，网格化直接将数据空间划分成等间距的网格，每个数据点会被分配到对应的网格中。在计算核密度时，每个网格会被近似成一个点（一般选为网格中心），即实现了连续的数据点离散化操作。由于近似操作，网格化需要选取合适的网格大小：如果网格过大，近似时的误差也会增大；如果网格过小，则会显著增加计算的时间复杂度，尤其是在高维空间中会呈指数增长。
- **快速傅里叶变换** [2, 32]。为了进一步解决网格化带来的额外计算开销，可以使用

快速傅里叶变换加速核密度的计算。具体来说，核密度的计算公式在离散情况下可以视作网格化矩阵和核方程矩阵的卷积，而快速傅里叶变换可以大大提高矩阵卷积的效率。快速傅里叶变换对于大型核和带宽特别有用。然而，对于高维矩阵和稀疏矩阵，其计算成本可能较高 [39]。

- **聚类** [33–35]。聚类和网格化类似，都是将一些数据点合并成一个点进行计算，不同的是聚类采用各种聚类算法进行划分，这使得对于稀疏分布更加友好。并且，现有的各种层级结构聚类算法也可用于生成层级结构的聚合集，进一步加速计算。
- **分桶** [2, 36, 37]。分桶也需要将连续的空间划分成若干桶，但和网格化不同的是，桶的分布是不均匀的，因此可以根据数据的分布动态调整桶的分布。在数据点稀疏或不重要的区域，可以分配较大的桶，而在数据密集且需要更高精度的区域，则采用更小的桶。

2.2 基于时间的方法

许多工作还考虑了对时间数据的分析。考虑到时间数据的复杂性，现有的工作可以分成如下几类难度：

- **静态数据分析** [40, 41]。静态数据是固定不变的，因此时间属性可以被视作一个单独的维度，并同样建立相关的索引。大部分支持高维核密度估计法的算法都可以直接扩展到静态数据上。
- **可持久化数据分析** [42, 43]。可持久化数据可以被更新或修改，形成新的记录，但不能修改已有的数据。这种数据往往用于记录操作信息，例如计算机日志信息或存储库修改信息。只有一部分算法经过修改后才可以实现可持久化数据的维护。
- **流数据分析** [44–46]。流数据是一种特殊的可持久化数据，它只会更新最新的数据信息（而不是在任意位置更新分支）。绝大部分数据都可以视作流数据，因为会有源源不断的新数据产生，并且在时间维度上不会和已有数据相交。流数据的分析是最广泛的，因为它的要求相对较低，且在现实世界中应用广泛。

2.3 核函数的计算

加权聚合是核密度计算优化中的一个重要操作，它可以将多个数据点的核密度合并计算。已有的工作只能实现多项式核函数的核密度聚合计算，而对于像余弦核函数和 Gaussian 核函数这样的非多项式核函数只能拟合到多项式核函数进行计算 [30, 31]，目前还没有精确计算的方式。

2.4 分布式计算平台

如果需要将算法扩展至分布式环境，通常会选择将其移植到特定的计算平台上。这种方法使得用户可以专注于算法本身的实现，而无需担心分布式环境下的各种系统设置问题，例如负载均衡和通讯优化等复杂任务。通过利用这些专门设计的计算平台，用户能够极大地简化开发流程，并加速算法的研发与部署。

在分布式环境中运行算法时，数据分布、任务调度、节点间的通信协调以及故障恢复等部署问题是非常具有挑战性的。这些问题不仅增加了系统的复杂性，还可能引入额外的性能瓶颈。现代的计算平台为了减少这些额外开销所带来的效率降低，一般都内置了强大的调度机制，可以自动管理资源分配，确保各个计算节点之间的负载均衡。此外，这些平台还优化了节点间的通信方式，减少数据传输延迟，提高整体的计算效率。

具体来说，当算法被移植到这些计算平台上时，开发者只需关注核心算法逻辑的实现即可。例如在图计算中，用户只需要定义顶点或边的行为，而平台则负责在后端处理所有的并行化细节，包括如何在多个节点上分配任务、如何优化数据传输，以及如何确保所有节点之间的一致性和同步性。这种高度抽象化的编程模型大大降低了分布式计算的门槛，即使是没有分布式开发背景的用户也能够轻松地将自己的算法并行化，并在大规模数据集上高效运行。

不仅如此，这些计算平台还提供了丰富的接口和工具库，进一步增强了算法的可扩展性和灵活性。无论是用于执行复杂的图分析任务、处理海量的数据流还是进行实时的数据挖掘，这些平台都能提供必要的支持。例如，某些平台专门针对幂律图中的负载均衡问题进行了优化，确保即使数据的分布不平衡，也能保持高

效的计算性能。其他一些平台则提供了对固态硬盘的顺序读写优化，以提高 I/O 操作的速度，这对于需要频繁访问磁盘的计算任务尤为重要。

Bulk Synchronous Parallelism（BSP）模型由 Leslie Valiant 于 1990 年提出 [47]，旨在提供一种高效的并行计算框架，特别适用于大规模图数据的处理。BSP 模型通过将计算过程划分为多个“超级步骤”（supersteps），每个超级步骤包含三个主要阶段：本地计算、通信和同步，能够有效地执行并行任务。这种方法不仅简化了并行编程的复杂性，还提高了计算效率和资源利用率。

基于 BSP 模型，Vertex-Centric（以点为中心）模式应运而生，并引入了“Think-Like-a-Vertex”（TLAV，像顶点一样思考）的设计理念 [48]。在以点为中心的计算模式中，顶点是计算和调度的基本单位。每个顶点都可以进行本地计算并向其他顶点发送消息。用户只需要实现一个基于顶点的计算接口，而平台将在所有顶点上执行用户代码，迭代计算固定次数或直到结果收敛。这一模型进一步简化了大规模图计算的设计与实现，使得开发者可以专注于每个顶点的行为和交互，而不必过多考虑底层的并行处理细节。

以点为中心的计算模式有一些变体，例如单阶段模型（Pregel [49], Pregel+ [50], Flash [51], Ligra [52]），Scatter-Gather 模型（Signal/Collect [53]）和 Scatter-Combine 模型（GRE [54]）。此外，一些已有的并行计算模型，例如 Spark，也可以在 VertexRDD 上提供类似于 Pregel 的接口来模拟 [55]。然而，以点为中心的计算模式面临着一些潜在挑战，包括负载不平衡问题，这在幂律图中尤为突出，以及在大规模集群上部署时显著的通信开销问题。

一些平台进一步扩展，对边或顶点组进行计算，从而分别发展出了 Edge-Centric（以边为中心）和 Block-Centric（以块为中心）的计算模式。PowerGraph [56]、X-Stream [57]、GraphChi [58] 和 Chaos [59] 均支持以边为中心的计算模式，通过对边执行任务来解决幂律图中的负载偏斜问题，并充分利用固态硬盘的顺序读写能力。另一方面，Blogel [60] 和 Grape [61] 支持以块为中心的计算模式，它们将图分割成多个块，使得属于同一块的顶点函数可以在没有通信的情况下进行交互。然而这两个计算模式也有使用上的局限性，以边为中心的计算模式不支持与非邻居顶点的通信，而以块为中心的计算模式在编程实现上较为复杂。

上述所有计算模型产生的输出大小与图的规模成正比，这使得它们不适合可能产生指数级结果的图挖掘问题 [62, 63]。为了解决这一问题，像 Arabesque [64]、Fractal [65]、AutoMine [66]、Peregrine [67] 和 G-thinker [68] 这样的平台采用了 Subgraph-Centric（以子图为中心）的计算模式。在该计算模式中，基本的计算单元是子图（例如三角形或矩形），而不是顶点、边或块。用户需要定义如何构建候选子图，并为每个子图实现一个计算接口。

第三章　基于核密度估计的时空路网数据可视化

3.1　问题定义及已有最优算法

3.1.1　问题定义

首先，我们给出 TN-KDE 问题的形式化定义以及已有的最优算法。本论文所用到的主要符号含义已列在表 3.1 中。

表 3.1　主要符号及对应描述

符号	描述
$G=(V,E)$	图 G，包括点集 V 和边集 E
q	查询的线段点
t	查询的时间
$o_i=(p_i,t_i)$	数据点 o_i，包括位置 p_i 和时间 t_i
$d(u,v)$	点 u 到 v 的最短路径距离
b_s, b_t	空间和时间维度上的带宽范围
L, N	线段点和数据点的总数
n_e	边 e 上的数据点

定义 3.1.1 (路网图)**.** 一个路网图可以被形式化地定义为图 $G=(V,E)$，其中 V 和 $E\subseteq V\times V$ 分别是点集和边集。任意两点 v_a 和 v_b 的距离 $d(v_a,v_b)$ 被定义为两点间的最短路径距离。

为了计算路网图上的核密度，我们根据已有的常见处理方式 [10, 28] 将每条边均匀分成若干相同长度的小段，称为线段点（Linear Pixel，Lixel）。和平面核密度计算中的像素点类似，每个线段点都需要执行一次核密度的计算，并且使用该线段点的中点作为度量距离的位置。

定义 3.1.2 (线段点)**.** 给定一个路网图 $G=(V,E)$, 每条边 $(v_a,v_b)\in E$ 都会被分成若干长度相同的线段点，其中距离间隔为 g，最后不足 g 的部分会被单独作为一个线

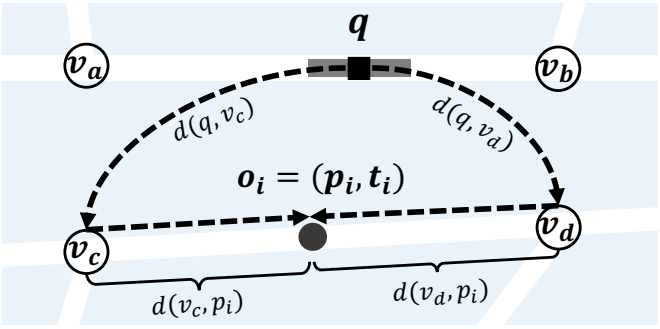


图 3.1　线段点 q（灰色线段）和数据点 o_i（黑色圆点）的可视化示例。

段点。在计算最短路径距离时，使用线段点的中点作为计算位置。

根据该定义，总共的像素点个数为：

$$L = \sum_{(v_a, v_b) \in E} \left\lceil \frac{d(v_a, v_b)}{g} \right\rceil$$

同理，数据点的定义如下：

定义 3.1.3 (数据点)**.** 一个数据点 $o_i = (p_i, t_i) \in O$ 表示在位置 p_i 和时间 t_i 发生事件。边 e 上的总数据点个数为 n_e。图上所有数据点的总数为：

$$N = \sum_{e \in E} n_e$$

图 3.1 简单展示了这些定义。q 是边 (v_a, v_b) 上的一个线段点，用灰色线段表示，其中心用黑色方块标出。o_i 是边 (v_c, v_d) 上的数据点，用黑色圆点标出。

TN-KDE 问题是给定查询的时间和空间带宽范围，在路网图上计算核密度：

定义 3.1.4. 时空路网核密度估计（*Temporal Network Kernel Density Estimation*，*TN-KDE*）给定一个路网图 $G = (V, E)$，一个数据点集合 O，空间带宽范围 b_s 和时间带宽范围 b_t，*TN-KDE* 问题是计算所有线段点的核密度 $F(q)$：

$$\begin{aligned} F(q) &= \sum_{o_i = (p_i, t_i) \in O} f(q, o_i), \\ f(q, o_i) &= K_s\left(\frac{d(q, p_i)}{b_s}\right) K_t\left(\frac{|t - t_i|}{b_t}\right), \end{aligned} \tag{3.1}$$

其中 $K_s(\cdot)$ 和 $K_t(\cdot)$ 是核密度函数，可以从表 *1.1*中任意选取。空间距离是最短路径距离 $d(d, p_i)$，时间距离为时间差 $|t - t_i|$。由于核函数的定义域为 $[0, 1]$，在带宽范围之外的数据点不会被纳入计算，即只考虑最短路径距离不超过 b_s，且发生在时间窗口 $[t - b_t, t + b_t]$ 内的数据点。当所有的线段点都根据该定义计算后，使用放缩因子 w 统一放缩后即可得到热力图（类似于图 *1.1*）。

3.1.2 已有的最优算法

根据我们广泛的调研，目前针对路网图上的核密度计算的最优算法为聚合距离增强算法（Aggregate Distance Augmentation，ADA） [28]。由于 ADA 算法仅支持空间距离，不支持时间距离，我们忽略时间核函数并采用 Epanechnikov 核函数作为空间核函数作为示例，即：

$$f(q, o_i) = 1 - \frac{1}{b_s^2} dist(q, p_i)^2$$

ADA 算法的第一个关键步骤是将所有数据点以边为单位计算核密度: The core step of ADA is to rewrite Equation 3.1 as:

$$F(q) = \sum_{e \in E} F_e(q)$$
$$F_e(q) = \sum_{o_i \in O_e} 1 - \frac{1}{b_s^2} d(q, p_i)^2$$

其中 O_e 包含所有在边 e 上的数据点。该公式只是重新定义了核密度的计算顺序，并没有改变计算复杂度，但之后我们可以仅专注于计算一条边上的数据点所产生的贡献，即 $F_e(q)$。

图 3.2 展示了 ADA 算法的主要思路，这里我们计算边 (v_c, v_d) 上的数据点对线段点 q 的核密度贡献。注意到 q 到所有数据点的位置 p_i 的最短路径一定会通过 v_c 和 v_d，此时最短路径分为两种情况：

- $q \to v_c \to p_i$。此时的最短路径长度为 $d(q, v_c) + d(v_c, p_i)$。
- $q \to v_d \to p_i$。此时的最短路径长度为 $d(q, v_d) + d(v_d, p_i)$。

显然选择第一种路径的数据点一定更靠近 v_c（用蓝色表示），而选择第二种路径的

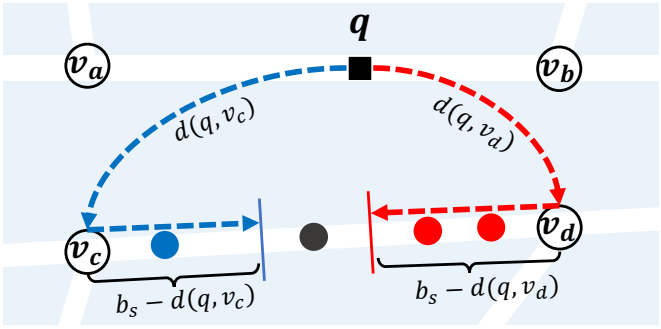


图 3.2　ADA 算法示例。

数据点一定更靠近 v_d（用红色表示）。对于第一种情况来说，这些数据点对核密度的贡献可以表示为：

$$
\begin{aligned}
F_\Gamma(q) &= \sum_{o_i \in O_\Gamma} \left(1 - \frac{[d(q, v_c) + d(v_c, p_i)]^2}{b_s^2}\right) \\
&= \frac{1}{b_s^2} \cdot \begin{bmatrix} -1 \\ -2 \cdot d(q, v_c) \\ b_s^2 - d(q, v_c)^2 \end{bmatrix}^\top \begin{bmatrix} \sum_{o_i \in O_\Gamma} d(v_c, p_i)^2 \\ \sum_{o_i \in O_\Gamma} d(v_c, p_i) \\ |O_\Gamma| \end{bmatrix},
\end{aligned}
$$

其中 O_Γ 是蓝色范围内所有的数据点的聚合集，即边 (v_c, v_d) 上距离 v_c 更近的那一部分。表达式中第一个向量及前面的系数 $\frac{1}{b_s^2}$（称为查询向量 $\mathbf{Q}$）对于某个特定的线段点 q 来说是定值，第二个向量（称为聚合向量 $\mathbf{A}$）则和数据点有关，分别是 $d(v_c, p_i)$ 的零次方和、一次方和、二次方和。

注意到聚合向量 $\mathbf{A}$ 满足结合律、结合律且可逆，因此当计算某个聚合集的聚合向量时，我们不需要每次都将所有的值都计算一遍，只需要计算前缀聚合向量 $\mathbf{A}_1, \mathbf{A}_2, ..., \mathbf{A}_{n_e}$，其中 $\mathbf{A}_i$ 表示 p_1 到 p_i 的数据点所构成的聚合向量，并且在查询时使用二分查询范围带宽的边界 $b_s - d(q, v_c)$ 最远可以包含哪一个数据点，即最大的下标 i 使得 $d(v_c, p_i) \leq b_s - d(q, v_c)$，就可以获取聚合向量 $\mathbf{A_i}$，然后直接使用 $F_\Gamma(q) = \mathbf{Q} \cdot \mathbf{A_i}$ 计算即可。

另一侧通过 v_d 的情况是类似的，只需要对编号反序即可。图 3.2展示了两种路径所产生的不同聚合集，分别用蓝色和红色表示，其中蓝色的聚合集更靠近 v_c，红色的聚合集更靠近 v_d。

该示例有一个隐性的条件，即空间带宽范围 b_s 较小，两个聚合集不会重叠。特别地，当 b_s 很大时，两个聚合集的边界不断靠近，最后会产生重叠。因此，除了原有的边界 $d(v_c, p_i) \leq b_s - d(q, v_c)$，还需要一个边界条件 $d(v_c, p_i) \leq \frac{-d(q,v_c)+d(q,v_d)+d(v_c,v_d)}{2}$，表示如果 q 到 p_i 的最短路径可以同时从 v_c 和 v_d 经过并到达，那么就选择更近的那一侧。另一边 v_d 的边界条件也是同理。

此外，查询向量 $\mathbf{Q}$ 的计算也并非毫无代价。虽然 $d(q, v_c)$ 对于给定的线段点 q 时是固定的，但是下一个线段点 q' 就需要重新计算所有的最短路径距离。共享最短路径算法（Shortest Path Sharing，SPS） [69] 就是用于解决这一问题，它可以在不同的线段点间共享最短路径的计算。具体来说，从 q 出发到 v_c 的最短路径同样存在两条路线：

- $q \to v_a \to v_c$。此时的最短路径长度为 $d(q, v_a) + d(v_a, v_c)$。
- $q \to v_b \to v_c$。此时的最短路径长度为 $d(q, v_b) + d(v_b, v_c)$。

当计算边 (v_a, v_b) 上的所有线段点时，可以提前预处理两个端点 v_a 和 v_b 到其他节点的距离；对于任意的线段点 q，只需要从 $d(q, v_a) + d(v_a, v_c)$ 和 $d(q, v_b) + d(v_b, v_c)$ 中组合出最优的路线即可。

综上所述，下面的引理给出了 ADA 算法的时间复杂度：

引理 3.1.5. *ADA 算法的时间复杂度为 $O(|E| \cdot T_{sp} + L \cdot |E| \cdot \log(\frac{N}{|E|}))$ [28].*

证明. 使用 SPS 算法需要对每条边计算一次最短路径距离，花费 $|E| \cdot T_{sp}$；对于每个线段点和每条边 e，使用二分搜索找到对应的聚合向量需要 $\log(n_e)$，预处理时间为一次性且线性，可忽略不计，这部分时间相加为 $L \cdot \sum_{e \in E} \log(n_e)$。使用算数-几何平均数不等式有：

$$\sum_{e \in E} \log(n_e) = \log\left(\prod_{e \in E} n_e\right) \leq \log\left(\sum_{e \in E} \frac{n_e}{|E|}\right)^{|E|} = \log\left(\frac{N}{|E|}\right)^{|E|} = |E| \cdot \log\left(\frac{N}{|E|}\right)$$

当且仅当所有的 n_e 相等时等号成立，即所有的数据点均匀分布在所有边上。

两部分时间复杂度相加即可得到总时间复杂度为 $O(|E| \cdot T_{sp} + L \cdot |E| \cdot \log(\frac{N}{|E|}))$。

□

3.1.3 算法框架

TN-KDE 问题相比之下更加复杂，由于额外加入时间维度，数据点的聚合集分类会更加复杂。具体来说，时间维度上的距离是 $|t-t_i|$，由于表达式中的绝对值函数，我们无法直接对这个表达式进行聚合，而是需要先分为 $t_i < t$ 和 $t_i \geq t$ 分类计算。例如，对于更靠近 v_c 且 $t_i < t$ 的聚合集，假设空间和时间核函数均采用三角核函数，对应的核密度计算公式会扩展为如下形式：

$$
\begin{aligned}
& F_\Gamma(q) \\
=& \sum_{o_i \in O_\Gamma} \left(1 - \frac{d(q,p_i)}{b_s}\right)\left(1 - \frac{|t-t_i|}{b_t}\right) \\
=& \sum_{o_i \in O_\Gamma} \left(1 - \frac{d(q,v_c)+d(v_c,p_i)}{b_s}\right)\left(1 - \frac{t-t_i}{b_t}\right) \\
=& \frac{1}{b_s b_t} \cdot \begin{bmatrix} -1 \\ b_s - d(q,v_c) \\ -(b_t - t) \\ (b_s - d(q,v_c))(b_t - t) \end{bmatrix}^\top \cdot \begin{bmatrix} \sum_{o_i \in O_\Gamma} d(v_c,p_i)t_i \\ \sum_{o_i \in O_\Gamma} d(v_c,p_i) \\ \sum_{o_i \in O_\Gamma} t_i \\ |O_\Gamma| \end{bmatrix}
\end{aligned}
\tag{3.2}
$$

虽然核密度的计算额外增加了一个维度，但是依然可以拆解为两个向量。同理，第一个查询向量 **Q** 仅和线段点 q 有关，第二个聚合向量 **A** 是聚合集中的所有数据点的属性值相加。算法 1描述了解决 TN-KDE 问题的主要框架。对于一个线段点 q，遍历每一条边 e 上的所有聚合集 Γ，并获取查询向量 **Q** 和 **A**，即可计算 Γ 对核密度的贡献 $F_\Gamma(q)$。将这些核密度累加起来，即可得到结果线段点 q 上的核密度 $F(q)$。

该框架和 ADA 算法类似，主要的难点在于如何高效地获取聚合向量 **A** （算法1第 4 行）。ADA 算法只需简单的二分搜索即可获得聚合集及对应的聚合向量，但该方法无法简单扩展至高维空间，即在 TN-KDE 问题中，时间维度的加入使得数据点和聚合集的维护变得复杂。在本论文中，我们将探讨对每条边的数据点建立高效的索引，以维护这些聚合集而不增加额外的时间复杂度。

Algorithm 1: TN-KDE 问题框架

```
for each lixel q do
    for each edge e = (v_c, v_d) ∈ E do
        Get shared shortest path d(q, v_c) and d(q, v_d)
        for each aggregation Γ on edge e do
            Q ← the query vector
            A ← the aggregated vector
            F_Γ(q) ← Q · A
            F_e(q) ← F_e(q) + F_Γ(q)
    F(q) ← F(q) + F_e(q)
```

3.2 高效的聚合索引算法

首先，我们提出了一个叫做区间森林法（Range Forest Solution，RFS）的算法来维护每条边上的数据点，以支持高效的聚合向量动态查询。

3.2.1 区间森林法

区间树 [70] 是一种高效的树形数据结构，它以层次化的方式维护数据点，其中每一个树节点都代表了若干数据点的聚合集。

首先，所有的数据点会按照他们在边上的相对位置排序（例如根据 $d(v_c, p_i)$ 排序），并且所有数据点都会被存入根节点中。接着，每个节点会被递归地分成两个子节点，每个节点存放一半数据点，直到只剩下一个数据点为止。具体来说，假设树节点 u 存放了数据点 $\{o_l, ..., o_r\}$，那么它的两个子节点 $lc(u)$ 和 $rc(u)$ 会分别存放 $\{o_l, ..., o_{\lfloor (l+r)/2 \rfloor}\}$ 和 $\{o_{\lfloor (l+r)/2 \rfloor + 1}, ..., o_r\}$。这样，每个节点都维护了一个区间内的节点，即 $R(u) = [d(v_c, o_l), d(v_c, o_r)]$。

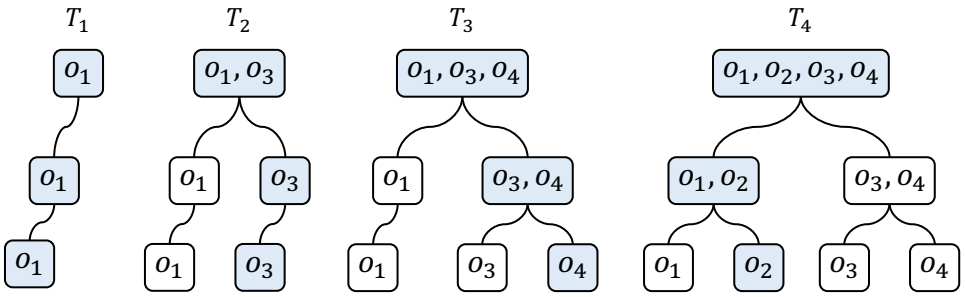


图 3.3　一个包含四个数据点的范围森林的例子，时间次序为 $\{o_1, o_3, o_4, o_2\}$。

图 3.3中的 T_4 就是一个区间树的例子，其中 o_1, o_2, o_3, o_4 四个数据点在边上位置依次从左往右。根节点包含四个数据点，每个树节点都继承了父节点中的一半数据点。

仅仅一棵区间树只能维护空间信息，为了加入时间信息，我们可以将数据点动态插入，并可持久化地记录所有中间状态，构成一系列区间树，也就是一个区间森林。图 3.3是一个区间森林地例子，假设四个数据点的时间顺序为 o_1, o_3, o_4, o_2，每次插入都会在原有的区间树上加入一个数据点，构成一棵新的区间树，其中更新的节点用蓝色表示，白色节点表示不变。注意，树节点所维护的区间 $R(u)$ 是根据最后结果（即 T_4 ）决定且不会变化的，即使其中的数据点还未被插入。

3.2.2 区间森林上的查询

区间森林上的查询包含两个步骤，即**生成**和**探查**。

生成：通过将两棵区间树“相减”，可以得到一棵新的区间树，且其中包含这个时间段内的所有数据点。图 3.4给出了一个包含数据点 o_3 和 o_4 的例子。因此，对于任意一个时间窗口，我们都可以快速获取该时间窗口内所有数据点构成的区间树。

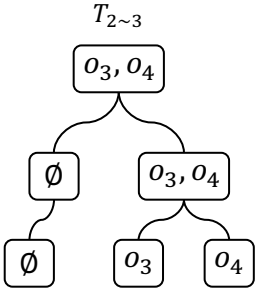


图 3.4　一个区间森林查询的例子。通过查询时间窗口，我们可以生成仅包含两个数据点 o_3 和 o_4 的区间树。

探查：在生成了特定的区间树后，时间维度上的查询就已经结束，接下来只需要将空间查询中的边界条件代入即可。在第三章中已经给出了 v_c 侧的两个边界条件：（1）数据点的位置不能超过带宽范围 b_s 和（2）数据点的位置必须距离 v_c 更近，即：

$$d(v_c, p_i) \leq b_s - d(q, v_c)$$

$$d(v_c, p_i) \leq \frac{-d(q, v_c) + d(q, v_d) + d(v_c, v_d)}{2}$$

另 $R = [0, \min(b_s - d(q, v_c), \frac{-d(q,v_c)+d(q,v_d)+d(v_c,v_d)}{2})]$ 表示查询的区间，我们需要这个区间内所有的数据点，即 $d(v_c, p_i) \in R$，及它们所对应的聚合集。这个步骤可以通过在区间树上递归查询解决，具体步骤如算法 2所示。假设查询的时间范围是 $[T_l, T_r]$，那么我们需要用 T_r 时刻的区间树减去 T_{l-1} 时刻的区间树。注意我们并不需要真的构造这一棵树，而是直接在两棵已有的区间树上遍历。我们用两个变量 u_l 和 u_r 记录这个遍历过程，初始值为两个根节点。每次递归时有三种可能：

- 如果 $R(u_r) \cap R = \varnothing$（第 4 行），说明该区间树节点的所有数据点都不包含在查询的区间内，直接返回零向量；
- 如果 $R(u_r) \subseteq R$（第 6 行），说明所有的数据点都包含在查询的区间内，可以直接返回这些数据点对应的聚合向量，即 $\mathbf{A}(u_r) - \mathbf{A}(u_l)$，这里利用到了聚合向量的计算可逆性；
- 否则，需要在两个子节点中进一步查询（第 8 行）。

Algorithm 2: 区间森林查询

Input: the temporal query range $[T_l, T_r]$
the spatial query range R
Output: the aggregated vector $\mathbf{A}$

```
1 u_l ← root(T_{l-1}), u_r ← root(T_r)
2 return DualDetect(u_l, u_r)
3 Function DualDetect (u_l, u_r):
4     if R(u_r) ∩ R = ∅ then
5         return 0
6     else if R(u_r) ⊆ R then
7         return A(u_r) − A(u_l)
8     else
9         return DualDetect(lc(u_l), lc(u_r)) + DualDetect(rc(u_l), rc(u_r))
```

区间森林的查询非常高效，达到了和传统的二分查询一样的时间复杂度：

引理 3.2.1. *给定一个在边 e 上的查询，算法 2 需要 $O(\log n_e)$ 的时间来获取聚合向量，其中 n_e 表示边 e 上的数据点个数。*

证明. 将连续的时间窗口映射到实际的查询范围 $[T_l, T_r]$ 需要使用二分查询，花费 $O(\log n_e)$。在 *DualDetect* 函数中，每个节点只有三种情况：不覆盖（第 4 行），全

覆盖（第 6 行），和部分覆盖（第 8 行）。对于部分覆盖的节点，它的子节点同样有两种情况：如果左子节点是部分覆盖，那么右子节点不覆盖；如果右子节点是部分覆盖，那么左子节点全覆盖。因此，在该递归操作所对应的区间树中，部分覆盖的节点数量只会减少不会增加，即每一层最多只有一个，所以每一层中最多只会有两个节点会被访问到。因此，查询时间复杂度为区间树的高度。此外，由于区间树是平衡的，所以树的高度为 $\lceil \log n_e \rceil$，时间复杂度为 $O(\log n_e)$。 □

3.2.3 区间森林的构造

如果我们直接如图 3.3所示构造完整的区间森林，虽然查询速度和二分查询类似，但索引会占用大量的内存空间。注意到相邻的两棵区间树存在高度相似的结构（白色节点），只有 $O(\log n_e)$ 个被插入的树节点存在不同。因此，我们可以只记录这些更新的节点，而另外不变的节点可以直接链接到原区间树上。如图 3.5所示，这种共享结构只会改变底层的数据引用，对节点之间的逻辑关系不会有任何影响。因此，算法 2无需任何修改。

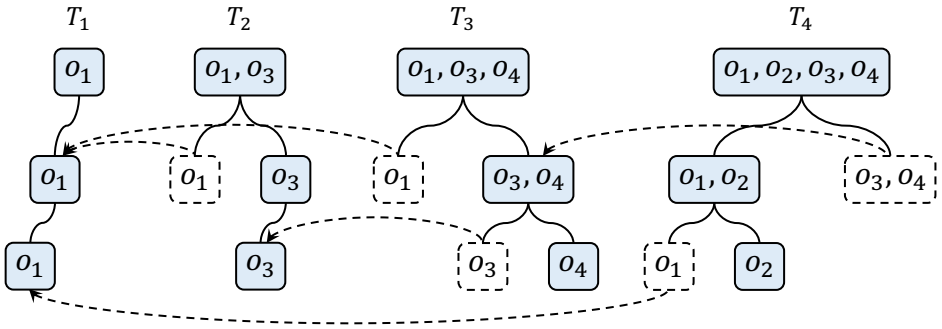


图 3.5 共享内存版本的区间森林。只有被更新的节点（用蓝色标记）是实际存在的，其他没有修改的节点会被链接到前面的区间树节点上。

算法 3 描述了优化版本的区间森林的构造流程。在第 2-4 行，不断有数据点按照时间顺序依次插入最新的区间树中。在每次构造中，树节点会被动态创建（第 6 行）并更新聚合向量（第 7 行）。如果数据点 o_i 将被分配到左子节点 $lc(u_r)$，即 $d(v_c, p_i) \in R(lc(u_l))$，那么不变的右子节点会被链接到原区间树的节点 $rc(u_l)$ 上，并且向左边继续递归；另一侧的更新操作同理。

经过优化后的区间森林构造时间复杂度和排序操作一样，因此可以认为是最

Algorithm 3: 区间森林构造

Input: event set $\{o_i\}$ on the edge e
Output: range forest T_i

sort $\{o_i\}$ by time t_i
for $i \in [1, n_e]$ **do**
    $u_l \leftarrow root(T_{i-1})$
    $root(T_i) \leftarrow DualConstruct(u_l, o_i)$
Function `DualConstruct` (u_l, o_i)**:**
    create a new tree node u_r
    $\mathbf{A}(u_r) \leftarrow \mathbf{A}(u_l) + \mathbf{A}(o_i)$
    if $d(v_c, p_i) \in R(lc(u_r))$ **then**
        link $rc(u_r)$ to $rc(u_l)$
        $lc(u_r) \leftarrow DualConstruct(lc(u_l), o_i)$
    else
        link $lc(u_r)$ to $lc(u_l)$
        $rc(u_r) \leftarrow DualConstruct(rc(u_l), o_i)$
    return u_r

够高效的:

引理 3.2.2. *对于一条边 e，算法 3 可以在 $O(n_e \log n_e)$ 的时间和空间复杂度内构造一个区间森林，其中 n_e 表示边 e 上的数据点个数。*

证明. 区间森林包含 n_e 棵区间树，对于一棵区间树的构造，每次只会往一个方向递归，因此需要递归 $\lceil \log n_e \rceil$ 次，每次操作都是常数复杂度，因此每棵区间树的构造时间均为 $O(\log n_e)$，总时间复杂度为 $O(n_e \log n_e)$。由于每个树节点都是动态创建的，每次递归只会创建一个节点，因此总空间复杂度也为 $O(n_e \log n_e)$。 □

使用和引理 3.1.5相同的技巧，我们可以计算得到 RFS 算法的总时间复杂度和空间复杂度。

引理 3.2.3. *RFS 算法的查询部分时间复杂度为 $O(|E|(T_{sp} + L \cdot \log \frac{N}{|E|}))$，索引构造部分的时间复杂度在最坏情况下是 $O(N \cdot \log N)$。空间复杂度在最坏情况下是 $O(N \cdot \log N)$。*

证明. 对于查询部分，RFS 算法和 ADA 算法的时间复杂度是相同的，即在边 e 需要 $\log n_e$ 的时间复杂度查询，因此使用相同的后续分析方式可以得到时间复杂度

为 $O(|E|(T_{sp} + L \cdot \log \frac{N}{|E|}))$。

对于索引构造部分，每条边 e 上的索引是一个树状结构，因此需要 $O(n_e \log n_e)$ 的时间构造，注意到：

$$\sum_{e \in E} n_e \log n_e \leq \sum_{e \in E} n_e \log N = N \log N$$

因此最坏情况下的构造时间复杂度为 $O(N \log N)$，当且仅当所有的数据点集中在同一条边上时成立。

事实上，由于查询部分和索引构造部分的最坏情况是完全矛盾的（均匀分布或集中分布），这就导致具体的时间复杂度很难量化。另一方面，即使分布不集中，对于索引构造的时间复杂度的影响也只会体现在更小的 $\log N$ 中，不会影响前面的主要变量 N。

对于空间复杂度来说，主要开销是区间森林的索引大小，这部分和索引构造的时间复杂度相同，即最坏情况下为 $N \log N$。 □

3.3 流式数据更新

区间森林算法 RFS 扩展了已有的核密度估计算法，在不额外增加时间复杂度的情况下扩展了带时间窗口的查询。然而，构造出来的区间森林索引是固定不变的，因此需要载入全部数据并确定整体索引结构后才可以进行构建和查询，无法支持动态插入。

考虑到任意的动态插入问题比较复杂，我们考虑一个弱化的，但在现实生活中同样广泛的场景，即流式插入。流数据是一系列按照时间排序后的数据，需要将他们动态地依次插入索引中。流数据保证只会在数据末端插入。

为了支持流数据的插入操作，我们需要将静态的索引改造成动态的索引。因此，我们提出了动态区间森林法（Dynamic Range Tree Solution，DRFS），其中每一个节点不再根据数据点的下标确定区间范围，而是直接根据边上的距离确定区间范围。

3.3.1 动态区间森林法

DRFS 算法的关键是将区间树和区间森林扩展成动态索引。和 RFS 算法类似，在 DRFS 算法中，动态区间树的每一个树节点 u 同样会维护一个区间 $R(u)$。对于根节点来说，它会维护整条边上的数据，即 $R(root) = [0, d(v_c, v_d)]$。接着，每个树节点的左右孩子分别会维护一半的区间（而不是一半的数据）：假设 $R(u) = [l, r]$，那么左孩子的区间为 $R(lc(u)) = [l, (l+r)/2]$，右孩子的区间为 $R(rc(u)) = [(l+r)/2+\delta, r]$。注意这里的 δ 是一个无穷小量，用于避免区分两个区间。在这种顶一下，无需初始数据也可直接在原图上初步建立索引。

图 3.6展示了一个动态区间森林的例子，其中 $\{o_1\}, \{o_2\}, \{o_3, o_4\}$ 分别落在第一、第三、第四个四分之一区间。区间森林上的查询（算法 2）和构造（算法 3）同样适用于这个结构，只需要修改对应的区间 $R(u)$ 即可。

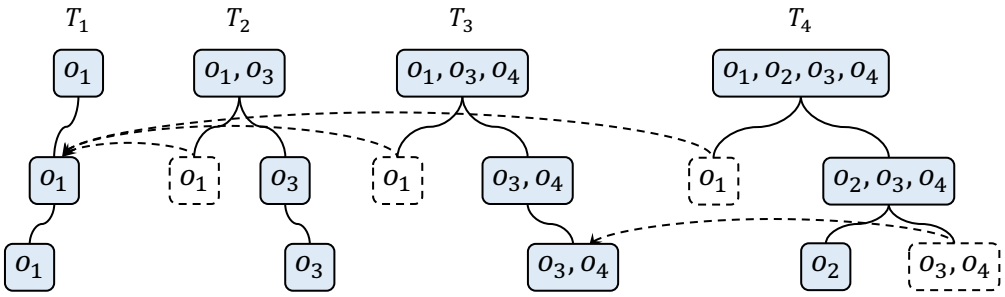


图 3.6　一个动态区间森林的例子。现在区间森林节点的区间是根据边的实际距离而不是节点下标划分的。

DRFS 算法并不是一个精确算法。在图 3.3中，静态区间树的每一个叶节点都仅包含一个数据点，因此 RFS 可以精确的判断带宽范围所在的位置，并且获取精确的聚合向量。然而在图 3.6中，动态区间树的部分叶节点可能包含多个数据点（例如 $\{o_3, o_4\}$）。如果该节点是部分覆盖的，此时却并没有更深的一层节点用于更精准的探查，无论是否返回聚合向量都会导致答案出现误差。

为了尽可能减少这一问题所带来的影响，我们进一步提出了扩展操作，用于扩展动态区间森林的层数。例如在图 3.6中，三层索引不足以精准区分所有数据点，那么扩展操作就可以扩展出第四层，并更新相关索引。在图 3.7中，一个四层的索引就足够划分所有数据点，其中被新扩展出的第四层结构用灰色方框展示。

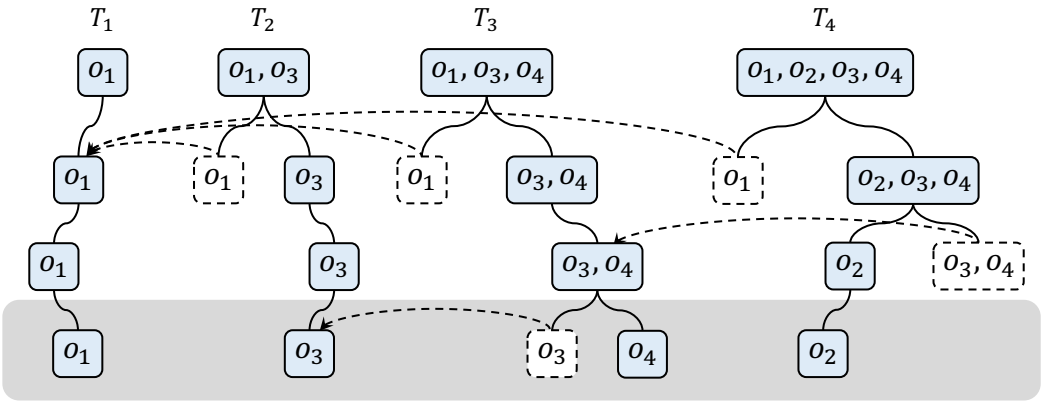


图 3.7　一个动态区间森林扩展的例子。灰色矩形部分就是新扩展的一层。

Algorithm 4: Extension Operation

Input: a range forest, event set $\{o_i\}$
Output: a range forest with an extended layer
for *each event* o_i **do**
    $u \leftarrow$ last updated node $leaf_i$
    call DualConstruct on u to extend a new layer
    save new updated node $leaf_i$

这一扩展操作可以被视作是一个索引上的懒惰操作，在算法 4中有相关的实现。具体来说，在初始构造或上一次扩展的同时，我们需要额外记录所有的叶节点 $leaf_i$。当需要扩展新的一层时，可以直接对这些叶节点调用构造算法中的 *DualConstruct* 函数。这就类似于将原来的构造函数中断，并在需要扩展时恢复。为了继续原来的构造步骤，只需要记录两个变量 o_i 和 u_l，即所需要更新的数据点，以及前一棵区间树上的更新位置。

因此，我们可以预先构造一个默认深度为 H 的动态区间森林，用户可以根据他们的精度和空间需求动态地调整深度参数 H。具体来说，深度越大，得到的结果精度也越高，但所占用的空间也会越大。并且，这种动态扩展的操作相比于直接构造并不会增加额外的时间开销，这就使得这种扩展操作更加灵活。

引理 3.3.1. *DRFS 算法的查询时间复杂度为* $O(|E|(T_{sp} + L \cdot H))$，*其中* H *是森林的深度。无论是通过动态扩展构造，还是直接构造，索引的构造时间和空间复杂度均为* $O(N \cdot H)$。

证明. 根据引理 3.2.3，我们使用固定的树深度 H 代替原有的深度 $\log n_e$，其他部分

的分析不变，因此时间复杂度为 $O(|E|(T_{sp}+L\cdot H))$。

构造一个深度为 H 的区间森林需要 $O(n_e\cdot H)$ 的时间复杂度，使用算法 4 扩展新的一层需要对每一个数据点所对应的区间树更新一次，即 $O(n_e)$。因此，动态构造深度为 $H+1$ 的区间森林所需要的时间为 $O(n_e\cdot H)+O(n_e)=O(n_e\cdot(H+1))$，这和直接构造深度为 $H+1$ 的区间森林的时间复杂度是一样的。而索引的大小也是用固定的树深度 H 代替原有的深度 $\log n_e$，即 $O(N\cdot H)$。 □

3.3.2 动态区间森林结构的量化

真实场景下的应用对于索引大小是高度敏感的。只要精度在可以接受的范围内，一个相对较小的索引会更受欢迎。因此，通过对索引进行量化可以在保证查询精度尽可能保持不变的情况下减少索引的大小。

由于动态区间森林的深度可以自由扩展新的层数以增大 H，或隐藏一些更深的层以减少 H，且而不会带来额外的开销，对高度 H 进行量化是一个有效的手段。

注意到在实际的运行过程中，大部分查询无需到达最深一层即会出现全覆盖或不覆盖的情况，此时即提前返回结果；另一方面，即使某次查询需要达到叶节点才能精确计算，提前结束查询所带来的误差并不会很大。因此，一个相对较小的量化高度 H 就可以快速产生一个相对比较精确的结果。

量化的效果直接由量化高度 H 决定。减少 H 可以显著减少时间和空间消耗。作为交换，叶节点将包含更多事件，从而可能导致更多部分覆盖的情况。在我们的实验中，与精确结果相比，即使 $H=2$ 也能达到超过 90$ 的精度，并且成本仅为原始数据内存的 $2^H=4$ 倍，这揭示了量化的高缩放能力。

3.4 线段点共享优化技术

范围森林法和动态范围森林法在单个线段点上的查询表现出高效的性能，然而不同线段点之间的查询任然是独立的，这是基于核密度估计法的另一个瓶颈。在本章中，我们将介绍一种专门用于多项式核函数的优化方法，称为线段点共享优化。

原始算法中的海量计算来源于不同的线段点会在索引上产生不同的查询。特

别地，如果查询的区间范围足够大，如图 3.8所示，该聚合集将会包含该边上的所有数据点。此时这些数据点对其他线段点的贡献具有相似性。

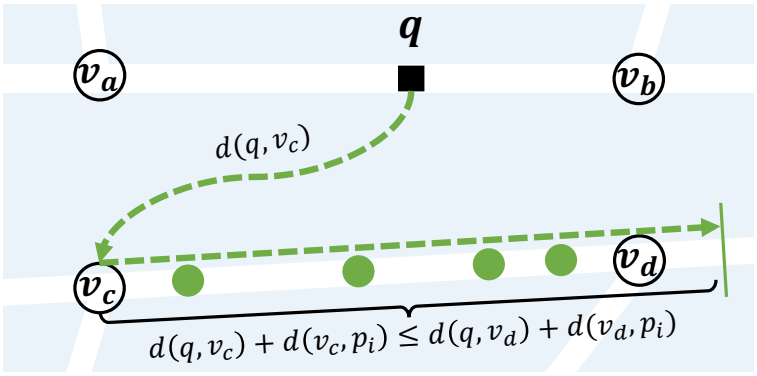


图 3.8　一个支配边的例子。从 q 到所有数据点的最短路径都会经过 v_c。

3.4.1　支配情况的判定

具体来说，如果边 (v_a, v_b) 上的所有线段点都满足该条件，即查询的区间范围能够覆盖整条边 (v_c, v_d)，我们称之为“支配”。例如在图 3.8中，边 (v_c, v_d) 在 v_c 方向上被支配，因为从所有线段点 q_i 到所有数据点的最短路径都会经过 v_c。形式化地说，对于所有线段点 $q_i \in (v_a, v_b)$ 和数据点 $p_i \in (v_c, v_d)$，支配情况有以下两个条件：

$$d(q_i, v_c) + d(v_c, p_i) \leq b_s$$
$$d(q_i, v_c) + d(v_c, p_i) \leq d(q_i, v_d) + d(v_d, p_i)$$

第一个条件表示所有的数据点都在空间带宽范围内。考虑最坏的情况，$d(v_c, p_i)$ 为整条边 $d(v_c, v_d)$ 的长度，$d(q_i, v_c)$ 为环 $v_c \to v_a \to v_b \to v_c$ 的长度的一半，即 $C/2$，其中 $C = d(v_c, v_a) + d(v_a, v_b) + d(v_b, v_c)$ 是环的长度。

第二个条件表示所有的数据点都必须离 v_c 更近，可以被改写为：

$$\max_{q_i}\{d(q_i, v_c) - d(q_i, v_d)\} \leq \min_{p_i}\{d(v_d, p_i) - d(v_c, p_i)\}$$

其中不等号右边最小值函数内的表达式具有单调性，因此 p_{n_e} 取到最小值；然而，不等号左边部分无法直接得到。事实上，有如下引理：

引理 3.4.1. *$d(q_i, v_c) - d(q_i, v_d)$ 的最大值最多只会在 4 个固定的地方出现，即只需要对 4 个 q_i 计算。*

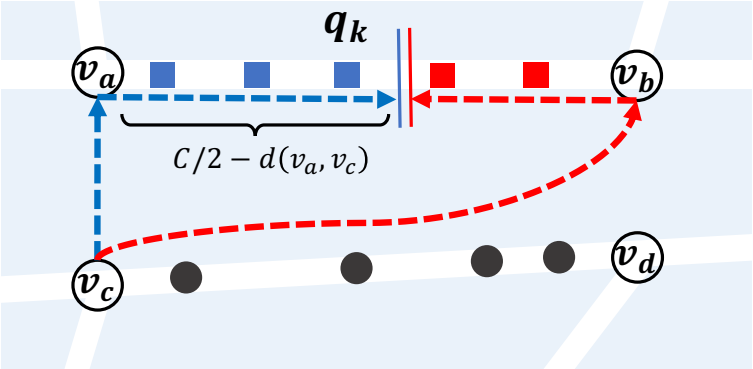


图 3.9　一个支配边的例子。从 v_c 出发到所有线段点的最短路径有两种路线，分别经过 v_a 或 v_b。

证明. 如图 3.9所示，$d(q_i, v_c)$ 的最短路径一定是左边一部分从 v_a 经过（用蓝色表示）而右边一部分从 v_b 经过（用红色表示）。因此，$d(q_i, v_c)$ 的值一定在 $q_1 \sim q_k$ 单调递增，而在 $q_{k+1} \sim q_{l_e}$ 单调递减，其中 k 如图所示两段的分割点，l_e 表示边 (v_a, v_b) 上的线段点数量，且两段的公差分别为 g 和 $-g$。

v_d 一侧的情况同理。$d(q_i, v_d)$ 的值一定在 $q_1 \sim q_{k'}$ 单调递增，而在 $q_{k'+1} \sim q_{l_e}$ 单调递减，其中 k' 是另一个分割点，且两段的公差同样分别为 g 和 $-g$。

此时 $d(q_i, v_c) - d(q_i, v_d)$ 就变成两个双等差数列的差，$q_1 \sim q_{\min\{k,k'\}}$ 部分和 $q_{\max\{k,k'\}+1} \sim q_{l_e}$，两个数列的单调性相同，所以差值为定值。$q_{\min\{k,k'\}+1} \sim q_{\max\{k,k'\}}$ 部分，两个数列的单调性相反，此时结果仍然是一个等差数列，且公差为 $2g$ 或 $-2g$。

综上所述，只需要检验四个位置即可确定最大值，即 k，$k+1$，k' 和 $k'+1$。 □

3.4.2　支配情况的计算

利用引理 3.4.1我们可以快速地判断一条边存在支配情况。如果一条边被支配，那么公式 (3.2) 中的聚合向量 **A** 对所有线段点都是相同的。此时，整个核密度公式

中唯一的变量就是 $d(q, v_c)$，且最高次数为一，即：

$$F_e(q_i) = \alpha \cdot d(q_i, v_c) + \beta$$

并且，相邻两个线段点的核密度值之差为：

$$F_e(q_i) - F_e(q_{i-1}) = \alpha \cdot (d(q_i, v_c) - d(q_{i-1}, v_c)) = \alpha \cdot \Delta(q_i)$$

引理3.4.1已经说明了 $d(q_i, v_c)$ 由两个单调数列组成，且分割点为 k。因此，核密度值的一阶差分 $\Delta(q_i)$ 为定值（除了分割点 k），二阶差分 $\Delta^2(q_i)$ 则为零（除了分割点 k）。图 3.10展示了核密度值及它们的一阶二阶差分，我们只需要在二阶差分上更新两个值，就可以记录所有线段点的核密度值。

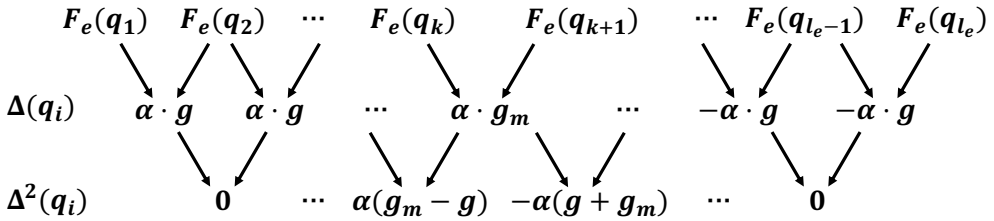


图 3.10　线段点的核密度的一阶差分 $\Delta(q_i)$ 和二阶差分 $\Delta^2(q_i)$ 的可视化表示。二阶差分 $\Delta^2(q_i)$ 除了分界点 k 附近的两个值，其他全为零。

注意到两个数组求和后的差分数组等价于分别差分后再相加，对于其他被支配的边，我们同样只需要再二阶差分上更新，最后依次还原成一阶差分和原数组即可。

最后，我们说明如何定位分割点 k。如图 3.9所示，q_k 是选择通过 v_a 的最远的线段点，因此：

$$k = \left\lceil \frac{C/2 - d(v_a, v_c)}{g} + 0.5 \right\rceil$$

这里额外增加的 0.5 是因为我们使用线段点的中点作为距离度量点，需要保证 q_k 所对应的中点必须在这个范围内。

最后，一旦得到了 k，我们就能得到需要更新的值：

$$g_m = d(q_{k+1}, v_c) - d(q_k, v_c)$$

3.4.3 越界情况的判定

图 3.8展示了如果聚合集包含了所有的数据点，可以进行一些优化。另一方面，如果聚合集不包含任意数据点，我们同样可以跳过这一过程。具体来说，我们有如下判定条件：

$$d(q_i, v_c) + d(v_c, p_i) > b_s$$

$$d(q_i, v_d) + d(v_d, p_i) > b_s$$

即所有的数据点都超出了空间范围带宽。考虑最坏的情况，$d(v_c, p_i) = d(v_d, p_i) = 0$，此时 $d(q_i, v_c)$ 和 $d(q_i, v_d)$ 的最小值只会在两个端点 q_1 或 q_{l_e} 出现。如果这两个线段点都距离很远了，那么整条边都可以被跳过。

3.4.4 带线段点共享优化的框架

算法 5是经过线段点共享优化后的框架。首先，第 3-5 行会分别寻找支配边集合 E_d，越界边集合 E_o，以及其他边 E_q。对于支配边，只需要更新核密度的二阶差分并恢复即可。接着使用原始流程计算其他边的核密度。

Algorithm 5: Lixel Sharing Framework

```
1  for each edge (v_a, v_b) do
2      Get shared shortest path distance
3      E_d ← dominated edges
4      E_o ← out-of-bandwidth edges
5      E_q ← E\(E_d ∪ E_o)
6      for each edge e ∈ E_d do
7          update Δ²(q_i)
8      recover F(q_i) from Δ(q_i) and Δ²(q_i)
9      for each lixel q_i ∈ (v_a, v_b) do
10         for each edge e ∈ E_q do
11             compute F_e(q_i)
12             F(q_i) ← F(q_i) + F_e(q_i)
```

引理 3.4.2. *算法 5的时间复杂度为* $O(|E| \cdot T_{sp} + |E|^2 + L \cdot |E_q| \cdot \log \frac{N}{E_q})$。

证明. 第 2 行计算最短路径花费 $O(|E| \cdot T_{sp})$；第 3-5 行判定支配边花费 $O(|E|)$；第 6-7 行计算支配边的核密度花费 $O(|E_d|)$；第 8 行恢复操作在所有边上总共花费 $O(L)$；第 8-12 行调用传统 RFS 算法会花费 $O(L \cdot (|E_q| \log \frac{N}{|E_q|}))$。因此总时间复杂度为 $O(|E| \cdot T_{sp} + |E|^2 + L \cdot |E_q| \cdot \log \frac{N}{E_q})$。 □

算法 5的效率提升主要由实际需要查询的边集 E_q 的大小所决定。在最坏情况下，没有边被支配或越界，此时 $E_q = E$，即没有任何提升。如果 E_q 较小，则复杂度会有显著的减少。

3.5 非多项式核函数

上述的核密度计算中我们使用最为常见的三角核方程作为示例，由于其可被划分为查询向量 **Q** 和聚合向量 **A** 的特性，使得我们可对核密度的计算进行加速。其他多项式核函数，例如 Epanechnikov 核函数也有相同的性质。然而，许多非多项式核函数，例如指数核函数和余弦核函数，无法直接通过这种方式进行分解。

已有的工作尝试使用多项式核函数来逼近非多项式核函数，例如双线性函数逼近 [31] 和二次函数逼近 [30]。这些方法能够达到较好的精度，然而逼近函数的参数选择极为复杂，并且误差仍然存在且无法收敛。

为了解决这一问题，我们的框架可以直接支持这些非多项式核函数，并且同样可以拆分为查询向量和聚合向量的乘积：

$$F_\Gamma(q) = \mathbf{Q}(q) \cdot \mathbf{A}(\Gamma)$$

只需要替换对应的查询向量、聚合向量，并重载对应的计算方式，即可实现不同核函数的调用。

3.5.1 指数核函数

指数核函数的定义如下：

$$K(x) = e^{-x}$$

为了简化问题，我们只展示空间维度上的变换，即 $K_s(x) = e^{-x}$，$K_t(x) = 1$。此时，核密度为：

$$\begin{aligned} F_\Gamma(q) &= \sum_{o_i \in \Gamma} e^{-\frac{d(q,v_c)+d(v_c,p_i)}{b_s}} \\ &= e^{-d(q,v_c)/b_s} \sum_{o_i \in \Gamma} e^{-d(v_c,p_i)/b_s} \end{aligned}$$

此时，查询向量和聚合向量分别为：

$$\mathbf{Q}(q) = e^{-d(q,v_c)/b_s} \quad \mathbf{A}(\Gamma) \quad = \sum_{o_i \in \Gamma} e^{-d(v_c,p_i)/b_s}$$

3.5.2 余弦核函数

另一个被广泛使用的核函数是余弦核函数：

$$K(x) = \cos(x)$$

余弦核函数的分解需要利用三角函数的变换性质：

$$\begin{aligned} F_\Gamma(q) &= \sum_{o_i \in \Gamma} \cos\left(d(q,v_c)/b_s + d(v_c,p_i)/b_s\right) \\ &= \sum_{o_i \in \Gamma} \left[\cos\frac{d(q,v_c)}{b_s}\cos\frac{d(v_c,p_i)}{b_s} - \sin\frac{d(q,v_c)}{b_s}\sin\frac{d(v_c,p_i)}{b_s}\right] \\ &= \cos\frac{d(q,v_c)}{b_s}\sum_{o_i \in \Gamma}\cos\frac{d(v_c,p_i)}{b_s} - \sin\frac{d(q,v_c)}{b_s}\sum_{o_i \in \Gamma}\sin\frac{d(v_c,p_i)}{b_s} \end{aligned}$$

此时，查询向量和聚合向量分别为：

$$\mathbf{Q}(q) = \begin{bmatrix} \cos\frac{d(q,v_c)}{b_s} \\ -\sin\frac{d(q,v_c)}{b_s} \end{bmatrix}^\top \quad \mathbf{A}(\Gamma) = \begin{bmatrix} \sum_{o_i \in \Gamma}\cos\frac{d(v_c,p_i)}{b_s} \\ \sum_{o_i \in \Gamma}\sin\frac{d(v_c,p_i)}{b_s} \end{bmatrix}$$

3.5.3 高维情况下的核密度公式

无论是空间核函数还是时间核函数，都可以任意选择核函数进行组合：

$$\begin{aligned} F_\Gamma(q) &= \Big[\mathbf{Q}_s(q)\cdot\mathbf{A}_s(\Gamma)\Big]\cdot\Big[\mathbf{Q}_t(q)\cdot\mathbf{A}_t(\Gamma)\Big] \\ &= \left(\sum_i \mathbf{Q}_i(q)\cdot\mathbf{A}_i(\Gamma)\right)\cdot\left(\sum_j \mathbf{Q}_j(q)\cdot\mathbf{A}_j(\Gamma)\right) \\ &= \sum_{i,j}\mathbf{Q}_{ij}(q)\cdot\mathbf{A}_{ij}(\Gamma) \end{aligned}$$

其中 $\mathbf{Q}_{ij}(q)=\mathbf{Q}_i(q)\cdot\mathbf{Q}_j(q)$ and $\mathbf{A}_{ij}(\Gamma)=\mathbf{A}_i(\Gamma)\cdot\mathbf{A}_j(\Gamma)$。最后的查询向量和聚合向量的大小 $|\mathbf{A}_{ij}|$ 最大为 $|\mathbf{A}_i|\cdot|\mathbf{A}_j|$，即两个维度上的相关变量的两两组合，在实际计算中可以视作常数。

3.6 实验设计与结果分析

在这一章中，我们评估了本论文提出的算法在不同参数设置和多种数据集下的表现情况。所有算法均使用 C++17 实现，并用 g++ 编译器以-O3 优化级别进行编译。所有的代码均运行在 Intel Xeon Gold 6258R @ 2.70GHz 的 CPU 上，内存为 128GB。在所有算法中，默认使用三角核函数作为默认核函数，并采用 Dijkstra 算法计算最短路径距离。

3.6.1 数据集介绍

我们使用了四个具有不同规模和类别的数据集，相关参数列在表 3.2中，$|V|$ 和 $|E|$ 分别表示路网图的点数和边数，$|N|$ 表示数据点数量，$N/|E|$ 平均每条边上的数据点数量。所有路网数据均从 OSMnx 包从 OpenStreetMap 上下载获得。每条边都被假设为一条直线，并且每个数据点会被匹配到距离最近的边上。数据点的来源包括包括报警记录、付费停车记录和出租车行程记录。

表 3.2　数据集参数

数据集	$\|V\|$	$\|E\|$	N	$N/\|E\|$
Berkeley	1576	4378	735366	168
Johns Creek	3074	3471	979072	282
San Francisco	9700	16008	5379023	336
New York	55765	92229	38400730	416

3.6.2　RFS 算法和已有工作的对比

首先，我们评估了我们提出的带有线段点共享优化的区间森林法（RFS）在生成精确核密度值时的效率，并与基线算法——最短路径共享算法（SPS）和最优算法——聚合距离增强算法（ADA）进行对比。SPS 算法只采用最基本的最短路径共享框架，而不会建立任何索引；ADA 算法会每次根据时间窗口过滤所有的数据点，并建立线性索引。

每组测试包含多轮不同的时间窗口查询，并且以在线的形式给出，即必须实时对于每个查询立刻返回结果。每个时间窗口默认包含 70% 的数据点。

以范围带宽为自变量。首先，我们在四个数据集中使用单组查询（而不是多组查询）评估不同带宽（50 米、1000 米、3000 米、5000 米）下的处理时间。线段点的长度设置为 10 米，这是一个合适的分辨率。

结果以对数形式展示在图 3.11 中。没有索引的 SPS 算法比其他基于索引的方法慢 1 到 2 个数量级，尤其是在较大带宽的情况下。RFS 算法相比 ADA 算法最多可实现 6 倍的加速，但在较小网络和较低带宽的情况下表现不佳，这是因为在索引构建上花费了过多的时间。

以多轮查询次数为自变量。前面的实验是对单个查询设计的。然而对于多轮查询，ADA 算法必须针对不同的时间窗口重建索引。因此，我们测试了不同查询次数对结果的影响，查询轮数分别为 5、10、15、20、25。空间带宽设置为 50 米，线段点长度为 50 米。在这种情况下，重新加载和索引所有事件将耗费较多的时间。

图 3.12 显示了所有方法的时间都呈现出线性增长的趋势，其中截距代表索引构建时间，而斜率则代表每次查询的处理时间。RFS 算法在小数据集（例如 Berkeley）

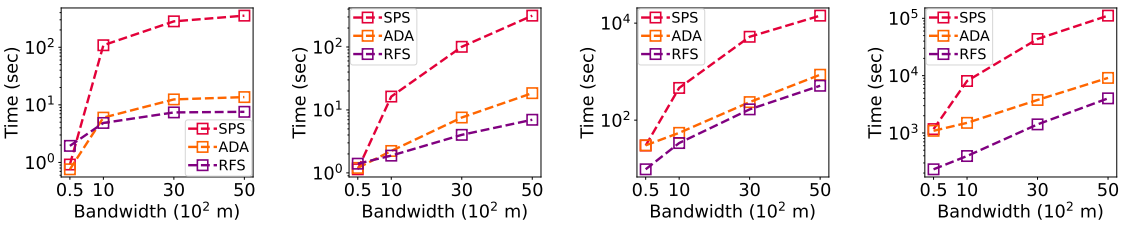


图 3.11　不同空间范围带宽（50m，1000m，3000m，5000m）下的处理时间。

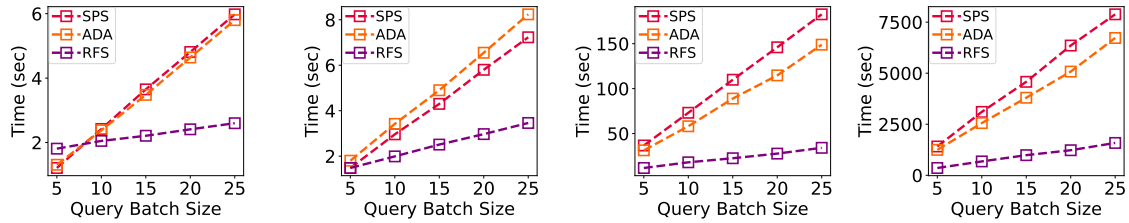


图 3.12　不同查询数量（5，10，15，20，25）下的处理时间。

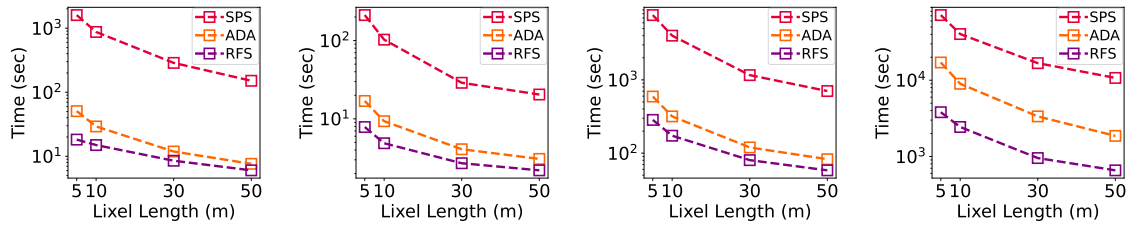


图 3.13　不同线段点长度（5m，10m，30m，50m）下的处理时间。

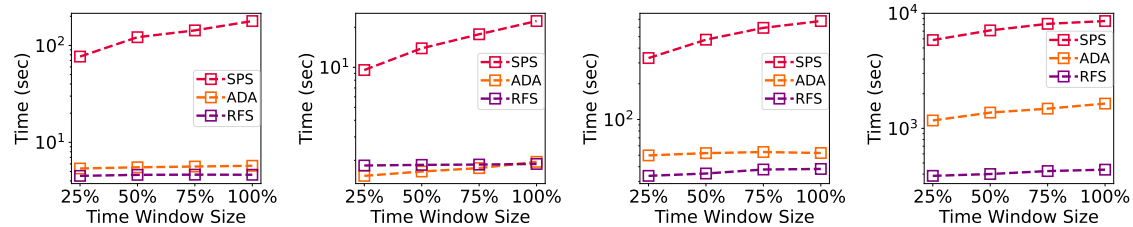


图 3.14　不同时间窗口大小（25%，50%，75%，100%）下的处理时间。

上的预处理时间相对于查询的时间较高，但其增长比率远低于 SPS 和 ADA，导致总的处理时间依然较少。对于其他三个数据集，RFS 算法在所有情况下表现得更好。

以线段点长度为自变量。线段点的长度直接决定了分辨率。对于一个粗略的快速视图，50 米的线段点已经足够，因为平均边长只有 100 米到 200 米。然而，用户可能为了提高分辨率而设置更小的线段点长度。在这部分实验中，线段点长度分别为 5 米、10 米、30 米和 50 米。每个查询批次包含 5 个时间窗口。

图 3.13 以对数形式展示了不同的线段点长度（5 米、10 米、30 米、50 米）下的处理时间。所有结果显示了大致的反比关系，因为较小的线段点长度会导致更多的线段点被创建，从而增加计算开销。与其它方法相比，RFS 算法表现出显著的优势，特别是在较低的线段点长度下，相比于 ADA 算法的速度提升高达 4.5 倍。

以时间窗口大小为自变量。我们还希望了解数据点规模如何影响查询效率，因此我们设置了不同大小的时间窗口，分别为总事件数的 25%、50%、75% 和 100%。

如图 3.14 所示，SPS 算法的处理时间与数据点数量呈线性关系，因此随着时间窗口大小的增加，处理时间也随之增加。相反，RFS 算法的处理效率与时间窗口大小无关。ADA 算法则略有不同，因为时间窗口大小仅影响预处理步骤，但不影响后续查询步骤。因此，即使事件数量增加，RFS 算法的效率也不会受到影响。

内存开销。图 3.15 展示了三种算法的内存使用情况。由于 SPS 算法不需要任何额外的空间，其内存使用量可以视为数据集本身的大小。其他两种算法（RFS 算法和 ADA 算法）的内存使用仅与数据点数量相关。RFS 算法在构建森林索引时仅需消耗相当于 ADA 算法 3 倍和 SPS 算法 8 倍的内存。考虑到复杂的索引结构，这样的内存开销是可以接受的。

3.6.3 DRFS 算法的高效性和有效性

这一节评估了动态区间森林算法（DRFS）在不同的区间森林高度 H 下的高效性和有效性。为了进行比较，我们使用了没有线段点共享优化的 RFS 算法，它具有静态结构。实验选择了两个带宽范围：1000 米和 20000 米，线段点长度固定为 50 米，时间窗口包含了所有事件。算法初始设置 $H = 1$，并逐步增加深度以展示

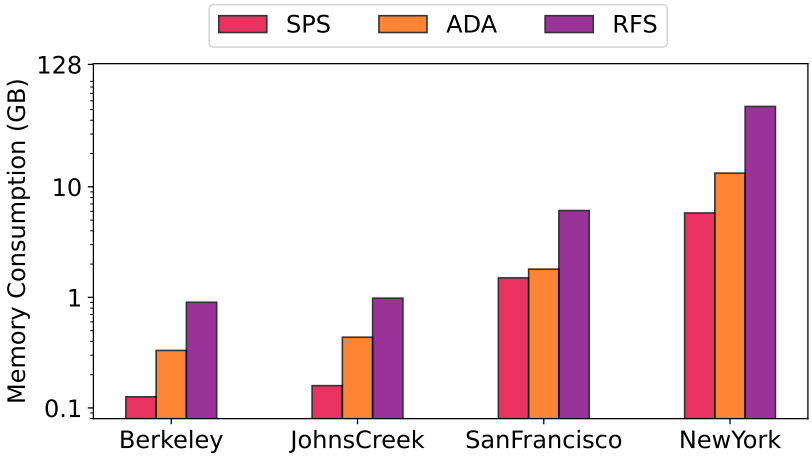


图 3.15　不同算法消耗的内存对比。

其动态过程。

高效性。为了说明索引和处理过程中的时间消耗，图 3.16 和图 3.17 分别展示了索引时间和处理时间。即使当 $H = 10$ 时，DRFS 算法构建索引所需的时间也很少，并且在大数据集（例如 NewYork）上甚至比 RFS 算法更快。大约来说，RFS 算法的效果等同于 $H \approx 8$ 的 DRFS 算法。在处理时间方面，不同的高度 H 之间没有显著差异。由于在一个查询批次中索引部分只执行一次，即使 H 较大，DRFS 算法在高效性上依然表现良好。

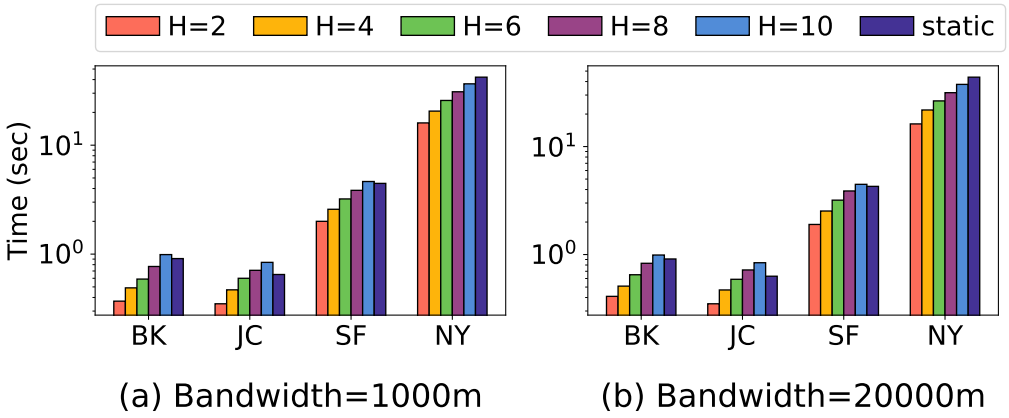


图 3.16　不同高度 H 所需要的索引时间对比。

有效性。由于 DRFS 算法是一种近似算法，我们也需要考虑其有效性，即计算出的核密度值是否准确。所有实验报告的准确率结果如图 3.18所示。即使在 $H = 2$

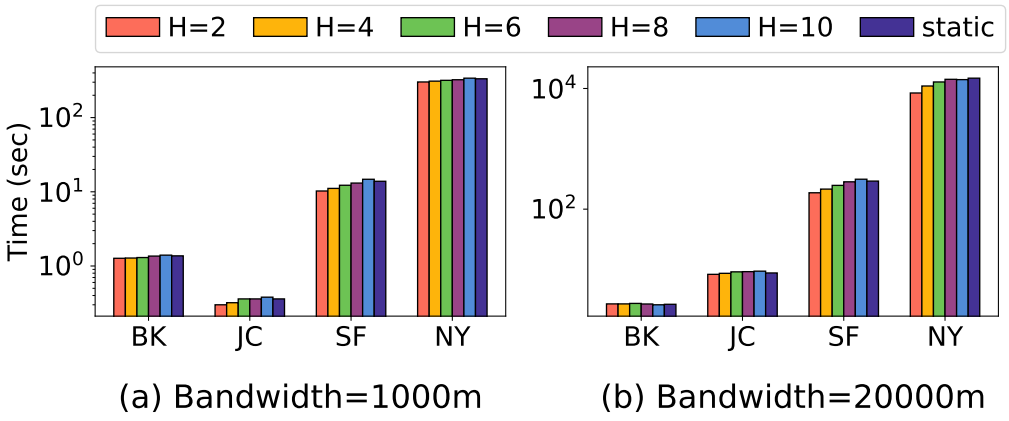


图 3.17　不同高度 H 下的查询时间对比。

的情况下，准确率仍然超过 94%。此外，随着 H 的增加，准确性迅速提升，当 $H = 10$ 时，所有四个数据集上的准确率均超过了 99.9%。

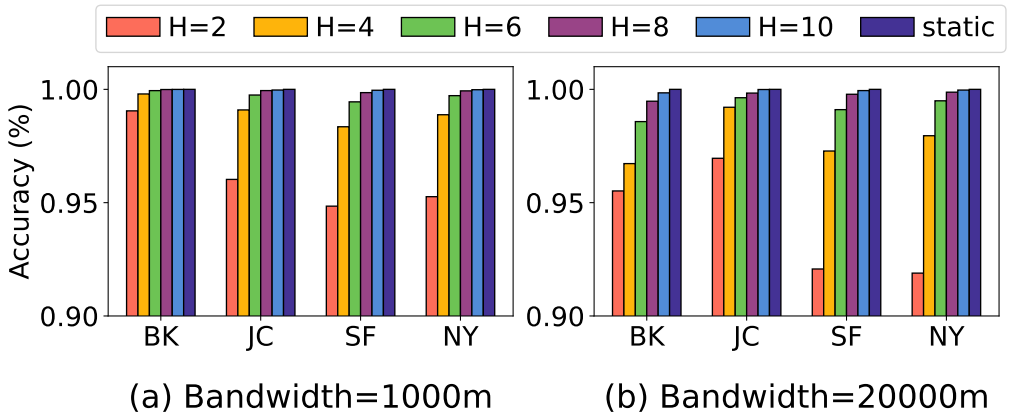


图 3.18　不同高度 H 下的答案准确度对比。

内存开销。图 3.19 显示了两种算法的内存开销，这反映了索引的大小。其趋势与图 3.16 类似，等效边界同样是 $H \approx 8$。当 $H = 2$ 时，内存消耗接近 ADA 算法，表明我们量化策略的可行性。此外，内存使用的增长比率几乎是线性的且可以预测，这可以帮助用户选择合适的参数 H。

综合考虑所有这些方面，实验表明 DRFS 算法具有很高的实用性。较小的 H 值可以显著节省时间，并且仍然能够产生相对准确的结果；而较大的 H 值仅需稍微多一点的时间即可生成几乎相同的结果。因此，用户可以根据需要选择合适的

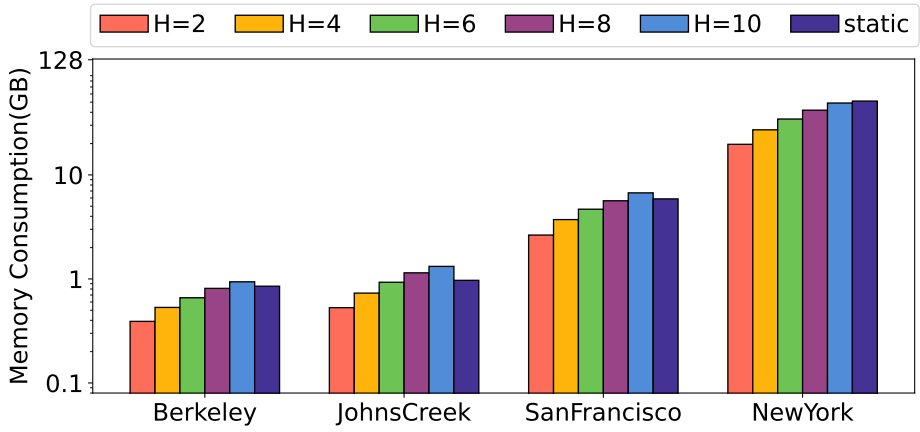


图 3.19　不同高度 H 所需要的内存开销对比。

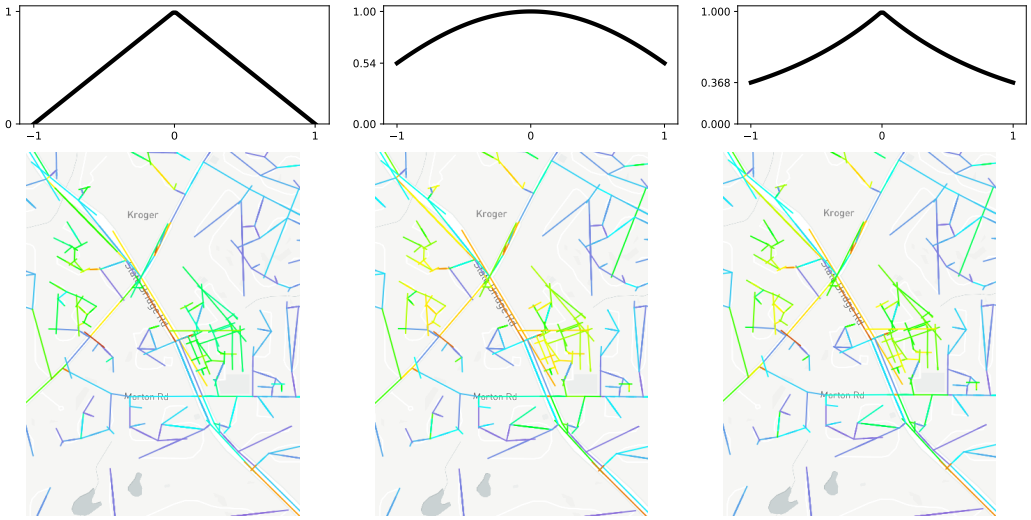
H 值，并动态调整它。

3.6.4　不同的核函数

该算法框架的另一个特点是可替换的核函数。由于使用不同核函数的所有计算过程都具有 $O(1)$ 的时间复杂度，因此它们可以在相同的时间内生成热力图。

图 3.20 展示了使用三角核函数、余弦核函数和指数核函数生成的三个不同的热力图。每个热力图上方都有其对应的函数图，所有密度值都被归一化处理。具体来说，余弦核函数的值始终大于指数核函数的值，而指数核函数的值又大于三角核函数的值，且斜率从高到低的依次是三角核函数、指数核函数和余弦核函数，这与结果的平滑度有关。此外，三角形核函数没有截断，而余弦和指数核函数在 ± 1 处被截断，这导致结果更加集中且边界更加明显。

小结。我们提出的 RFS 算法在实现在线 TN-KDE 问题的查询时，相比 ADA 算法和 SPS 算法分别呈现了高达 6 倍和 88.9 倍的加速。随着数据集规模的增大以及分辨率要求的提高，这种改进将更加显著。RFS 需要更多的空间来存储较大的索引，但其内存使用仅增加了 8 倍，并且在处理更多事件时保持稳定。DRFS 算法在达到与 RFS 算法相似的时间和空间复杂度的同时，在大多数情况下保持了超过 99.9% 的准确率。当索引使用 $H = 2$ 进行量化后，DRFS 算法最多可节省 40% 的时间成本和 60% 的内存成本。我们还测试了其他核函数，并对其结果进行了可视化，发



(a) 三角核函数　　(b) 余弦核函数　　(c) 指数核函数

图 3.20　应用不同核函数的热力图结果。

现它们在高密度区域的结果近似，但在边界区域有所不同。

第四章　最短路径距离的分布式加速计算

本论文提出的 RFS 算法的时间复杂度为 $O(|E| \cdot T_{sp} + L \cdot |E| \cdot \log(\frac{N}{|E|}))$，即使采用了较优的 Dijkstra 算法，计算最短路径距离依然会花费相当一部分的时间。作为一个传统算法，在分布式环境下最短路径计算可以加速这一流程。

本章首先会介绍分布式图计算的相关背景，以及分布式图计算的技术综述，然后会给出分布式最短路径的相关算法。

4.1　分布式图计算的背景

图计算的一个瓶颈时当图的规模很大时，无法像传统算法一样快速并行计算。近年来，大量的分布式图处理框架和算法被提出，用于快速地将传统算法移植到分布式环境中。

将图计算任务应用于分布式环境中并不直接，稀疏的图结构在分布式计算中，通常会面临诸多挑战，包括：

- 并行性。提高分布式算法的并发程度，增加在单个时间周期内的计算量，减少总计算轮数。
- 负载均衡。平衡在每台机器和每个线程上的计算量，尽量避免由于计算量不均等导致的空闲等待。
- 通讯开销。减少在分布式环境中由于多个线程和机器之间同步信息所需要的额外消息传递，通常远远大于本地访问的时间开销。
- 带宽限制。分布式环境的通讯时间受到带宽的限制，并且许多框架为了效率还会限制单次通讯的数据量

4.2　分布式图计算的划分算法

图算法的并行较为复杂。一方面，图算法往往有较强的顺序关系，例如 Dijkstra 算法通过特定的计算顺序才能保证最优性；另一方面，图结构的局部性并不直观，

如果按照传统的编号划分，一个节点的邻域往往会分布在不同机器中。因此，许多研究也提出了相关的计算框架，将这些复杂的底层问题抽象成固定的计算模式。

图划分算法的任务是将一个完整的图划分为多个子图存放在不同的机器中，一个好的划分算法能够尽量减少子图之间的通讯数据量，尽量保证本地计算的高效性。常见的图划分算法包含以下三种：

- 点划分。点划分是最基础的划分算法，即以点作为划分单元，分配到不同的机器上。如果一条边的两个点都在同一台机器上，那么这条边只会被本地访问；否则，这条边可能会产生跨机器的通讯，称之为割边。如图 4.1所示，边 (u_1, v_1) 就是一条割边。点划分的目标是尽量减少割边的数量。

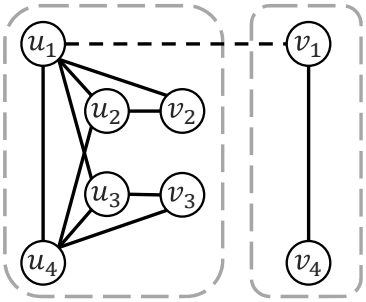


图 4.1　点划分示意图。

- 边划分。和点划分的定义类似，边划分以边作为划分单元，分配到不同的机器上。不同的是，一个点的边可能出现在多台机器上，称之为割点，因此无法采用传统的点对点同步方式，而是将其中一个点设为实际存储点，而将其他点作为镜像点，每次同步时镜像点和实际存储点之间进行同步操作。如图 4.2所示，点 v_1 就是一个割点。边划分的目标是尽量减少割点的数量。

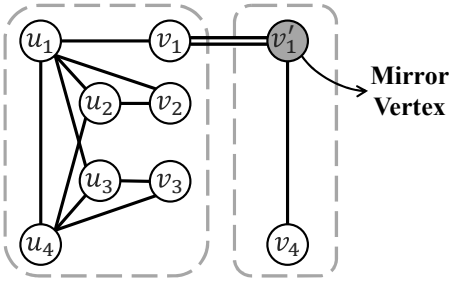


图 4.2　边划分示意图。

- 混合划分。注意到点划分和边划分都很容易引起负载均衡问题。一个割边或割点较少的划分并不意味子图的大小时平衡的，例如图 4.1和图 4.2中的子图大小相差过大。为了解决这一问题，混合划分集成了点划分的思路，但不再以割边为衡量目标，而是综合考虑各种参数，例如节点的度数、节点的连通度、节点的计算量等，直接量化估计整体的划分质量。图 4.3是一个优化后的划分，虽然存在三条割边，但是八个节点均匀的分配到两个子图中。

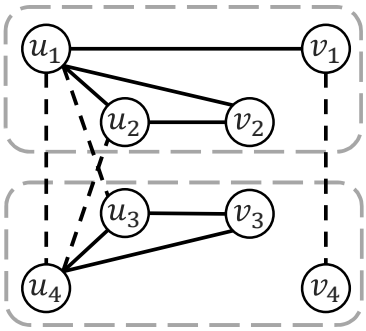


图 4.3　混合划分示意图。

4.3　分布式图计算的计算模式

目前，主流的图计算模式都依赖于块同步并行模型（Bulk Synchronous Parallelism，BSP）。BSP 模型的思想是将计算任务分为多个超步，每个超步包含以下三个阶段：

- 本地计算。每个线程独立地处理自己的计算任务，并且只能用本地存储的数据，无法和其他线程交互。
- 通讯。完成本地计算后，线程之间允许通过特定的通讯方式交换数据，并保存为本地数据，用于下一轮本地计算。
- 同步。同步操作用于确保所有的线程在完成了当前超步本地计算和通讯操作之后才能进入下一个超步，确保数据一致性。如果没有同步操作，就会出现部分线程读取到了旧数据，导致答案收敛变慢甚至出错。

BSP 模型将复杂的并行计算拆解成了三个步骤，并且各个阶段所需的计算资源不同：本地计算依赖于计算效率；通讯依赖于通讯的速度和带宽；同步则依赖于负载的均衡。从这三个维度入手，就可以清晰地看出某个特定的计算任务的瓶颈。

在图计算系统中，这三个步骤一般会被抽象为相关的接口函数，用户只需重载本地计算和通讯的代码，图计算系统会自动完成其他剩余的任务，例如图划分、通讯协议、同步信号等等，给用户带来了便利的编程体验。目前，主流的图计算模式分为以下几种：以点为中心、以边为中心、以块为中心、以子图为中心。

4.3.1 以点为中心的计算模式

以点为中心（Vertex-Centric）的计算模式是最基本也是最直观的计算模式。这种模型强调将图中的每个节点视为独立的计算单元，每个节点不仅能够在其内部执行特定的本地计算任务，还可以与其他节点通讯交换信息。这一设计理念极大地简化了复杂图算法的设计过程，允许用户专注于单个节点的行为逻辑，而不需要管理整个图的并发控制细节，尤其是本身就定义在节点上的算法，能够非常自然的迁移到以点为中心的计算模式上。

在以点为中心的计算模式中，用户的职责被简化为定义一个基于点的计算接口，该接口描述了每个点如何处理其状态更新以及如何向相邻节点发送消息。随后，底层平台会负责在整个图的所有节点上并行执行这个接口，可以设定固定的迭代次数或者直到系统达到稳定状态（即收敛）。这种方法不仅提高了开发效率，还增强了代码的可读性和维护性。例如，下述伪代码展示了如何实现最短路径的 Bellman-Ford 算法：

```
compute(vertex u):
    for m in messages:          # 更新当前的最短路径
        u.dist = min(u.dist, m)
    for [u, v, w] in edges[u]:  # 发送当前的最短路径
        send_message(v, u.dist + w)
```

其中 dist 数组和 edges 数组存储在本地，用于记录当前的最短路径距离和出边；messages 是一个保留字，用于存储上一轮通讯中收到的数据；send_message 也是一个保留字，用于向特定的节点发送数据，并保存在对应的 messages 中。该代码分为两部分，首先每个节点根据上一轮收到的最短路径更新自身的最短路径；然后遍历所有的边，构成新的可能的最短路径并发送给对应的节点。执行完该代码后，

视作完成了一个超步，由计算平台负责同步。

在这个例子中可以看到，这些计算模式可以大大简化用户层面的编程难度。此外，也有一些计算平台进一步扩展更高级的接口，例如将计算和通讯拆分成两个不同的函数 Scatter 和 Gather，能够更好地对两个阶段单独优化。

```
Scatter(vertex u):
    for [u, v, w] in edges[u]:  # 发送当前的最短路径
        send_message(v, u.dist + w)

Gather(vertex u):
    for m in messages:          # 更新当前的最短路径
        u.dist = min(u.dist, m)
```

此外，在多机多线线程的环境下，为了减少机器之间的通讯量，可以提前对数据进行聚合操作，即如果出现了多个发往同一节点的最短路径，可以直接筛选出最优的那一个。

```
Combine(target v, message m1, message m2):
    return min(m1, m2)
```

目前，70% 以上的工作都是基于以点为中心的计算模式完成的，这是最广泛的计算模式。然而，以点为中心的计算模式最大的缺点是负载均衡不佳，每一次同步时都需要等待所有线程完成后才能进入下一个超步。且在一些呈现幂律分布的图中，少量度数极大的节点会产生大量的通讯。

4.3.2 以边为中心的计算模式

注意到虽然在每个节点上的计算量往往会受到度数的影响而导致负载不均衡，但是在单条边上的计算量往往是恒定的，因此以边为中心的计算模式就是以边为调度单位。同样以最短路径举例，每条边只需要用起点的距离更新终点即可。

```
compute(edge [u, v, w]):        # 更新当前的最短路径
    update(v', u.dist + w)
```

由于以边为中心的计算模式存储的是完整的边，即采用了边划分的划分算法，因此可以每台机器各自的 v' 上实时更新，并在同步阶段和其他机器同步即可。此时用户只需要关注于本地计算阶段。

此外，同样有一些平台给出了细粒度更高的接口，允许用户分开定义计算操作，并且允许用户自定义是否激活节点，例如 GAS 模式：

```
Gather(edge [u, v, w]):          # 所有边计算出边的最短路径
    return u.dist + w

Sum(distance dist1, distance dist2):    # 合并相同终点的最短路径
    return min(dist1, dist2)

Apply(vertex u, distance dist): # 所有点更新最短路径
    u.dist = min(u.dist, dist)

Scatter(edge [u, v, w]):         # 如果最短路径发生变化，激活邻居
    if u.dist changed:
        activate(v)
```

Gather 函数统一负责计算需要更新的值，并且通过 Sum 函数聚合，减少通讯；Apply 函数更新值；Scatter 函数用于判断结果是否收敛。在这里，Gather 函数和 Scatter 函数都是定义在边上的，并且边划分可以非常方便地在本地设置镜像点，向用户隐藏了同步细节，而像是在一张完整的图上进行计算。然而，以边为中心的计算模式最大的缺点在于只能在邻居之间传递消息，无法和其他点通讯。

4.3.3 以块为中心的计算模式

以块为中心的计算模式则是更进一步，直接将图划分为多个互不相交的子图并进行计算。在计算阶段，块内部可以采用任意算法计算，只需要在同步阶段同步一轮数据即可。

```
compute(block b):                # 计算块内部的最短路径
    compute shortest path in b
```

以块为中心的计算模式特别适用于解决最短路径这种有顺序要求的问题，因

为在以点和以边为中心的计算模式中，最短路径算法的同步次数与图的直径相关，即至多需要直径轮才能将最短路径传播到所有节点。然而在以块为中心的计算模式中，由于块内部可以采用 Dijkstra 算法或 SPFA 算法等串行算法快速计算，减少了直径对执行时间的影响。

4.3.4 以子图为中心的计算模式

上述所有计算模式均无法较好的解决模式匹配问题，例如计算图中有多少个四阶完全图，一方面计算子图需要大量的通讯以同步邻居信息，另一方面模式的数量无法良好估计，也会产生严重的负载不均衡。

以子图为中心的计算模式则是尝试直接构造子图，并以子图为单位进行调度。它和以点为中心的计算模式类似，都是从每个节点触发尝试构造子图，而后续则是直接以子图作为任务调度的单位，尝试在原图中拉取邻居生成子图，并统计模式数量。

```
pre_compute(vertex u):          # 生成初始子图，仅包含一个点
    sg = [[u], []]
    subgraph.push_back([u])

compute(subgraph sg):
    u = sg.vertex[0]            # 获取初始子图对应的节点
    sg.add(pull(u))             # 获取邻居，并加入子图
    count(sg)                   # 在子图上统计具体的模式数量
```

4.4 常见图计算平台上的最短路径算法实现

由于各个图计算平台的接口各不相同，各自的实现方式也有很大差异，因此下面会给出一些常见的计算平台的基本介绍，以及这些计算平台上的最短路径算法实现。

4.4.1 GraphX 平台

GraphX 是 Apache Spark 上的一个分布式图处理框架，它使得用户能够在大规模图数据集上执行高效的图计算。GraphX 结合了 Spark 的强大功能，采用了和 Spark 一样的 RDD 存储结构，能够和已有的数据分析模块相结合，且自带了容错机制和故障处理机制，为用户提供了一个灵活且高性能的图处理解决方案。此外，GraphX 还提供了类似于 Pregel 的接口以提供高效的图计算。

然而，由于 Spark 本身依赖于 JVM，导致 GraphX 的计算效率较低，并且 Spark 中的基本数据单元 RDD 无法直接修改，导致图计算中每次迭代都需要拷贝数据才能实现更新，并且 Spark 采用 hadoop 存储数据，直接导致 IO 瓶颈。

GraphX 中的最短路径算法可以通过调用 Pregel 接口来实现，包含三个用户函数：VertexProgram，SendMessage，MergeMessage。在 VertexProgram 中，每个节点只需保存最优的路径；在 SendMessage 中，对于每一条出边，尝试构造一条路径并发送给目标节点。在 MergeMessage 中，发往同一顶点的最短路径会被合并，以减少通讯量。

注意，虽然这里的 SendMessage 参数是边，但 GraphX 依然是以点为中心的计算模式，只是在遍历每个点的出边时调用该函数。

```
VertexProgram(vertex u, distance dist, distance new_dist):
    return min(dist, new_dist)

SendMessage(edge [u, v, w]):
    if u.dist + w < v.dist:
        send_message(v, u.dist + w)

MergeMessage(distance dist1, distance dist2):
    return min(dist1, dist2)
```

4.4.2 PowerGraph 平台

PowerGraph 是一个高效的分布式图处理系统，设计用于高效地处理大规模图数据。它采用以边为中心的计算模式，特别擅长应对具有幂律分布特性的图（即少

数节点拥有大量的连接，而大多数节点只有少量连接），这类图在实际应用中非常常见，例如社交网络、网页链接结构等。

PowerGraph 的执行引擎经过优化，能够在大规模集群上高效地执行图算法。它利用了内存计算的优势，避免使用 hadoop 导致的 IO 开销，并通过减少不必要的数据传输来提高性能。

PowerGraph 中的最短路径算法可以通过 GAS 模型实现，包含四个用户函数：Init，Gather，Apply，Scatter。Init 函数负责记录上一轮收到的最短路径，且该消息已经经过聚合，也就无需使用 Gather 函数；Apply 函数将该最短路径更新；Scatter 函数尝试构造新的最短路径，并激活目标节点更新。

```
Init(vertex u, message m):
    new_dist = m

Gather(edge [u, v, w]):
    return null

Apply(vertex u):
    u.dist = new_dist

Scatter(edge [u, v, w]):
    new_dist = u.dist + w
    if new_dist < v.dist:
        signal(v, new_dist)
```

4.4.3 Flash 平台

Flash 是一个高效的图计算平台，能够同时支持对集合的操作，跨领域的通讯，以及灵活的控制流。Flash 平台是 Ligra 平台的扩展，Ligra 仅支持单机多线程，而 Flash 将其扩展到了多机上。Flash 设计了中间层 FlashWare 来实现高效的通讯、负载均衡、同步、调度等功能，并且只会更新实际需要的属性，避免整个节点的拷贝。整个中间层都是向用户隐藏的，可以自动地在任务中执行而无需用户配置。

Flash 相比于 Pregel 最大的优势是支持针对子集的操作，即可以自由地定义点集和边集并在此基础上执行图算法，而已有的大部分平台仅支持在全图上计算。除

了子集，Flash 甚至支持扩展集合的操作：例如为了统计矩形（四个节点组成的环）的个数时，可以直接对边集执行 join 操作，此时 E join E 就代表了两跳邻居，而每一对起终点相同的两跳邻居都能够组成一个矩形。

Flash 中的最短路径算法可以通过 EdgeMap 函数实现，和 Pregel 类似，对于每个顶点的每条出边，都会调用 EdgeMap 中的 Update 函数，用节点 u 的距离更新 v 的距离；不同的是，这里返回的是点 v 的一个拷贝，在 Reduce 函数中才会真正对比最短路径，并保留最优的一个。这种来自 C++ 语法的自由度给 Flash 带来了更方便的表达，能够支持更加复杂的语法。而在主函数中，通过 while 循环不断对激活点集 V 进行扩展，直至收敛。

```
Update(edge [u, v, w]):
    dis[v] = min(dis[v], dis[u] + w)
    return v

Reduce(vertex new_vertex, vertex old_vertex):
    if new_vertex.dist < old_vertex.dist:
        return new_vertex
    else:
        return old_vertex

main():
    while size(V) != 0:
        V = EdgeMap(V, E, Update, Reduce)
```

4.4.4 Pregel+ 平台

Pregel+ 平台是一个基于 Pregel 语法的优化版本，旨在优化 Pregel 的消息传递中高度数节点频繁发送消息所带来的通讯开销。Pregel+ 主要采用了和边划分类似的镜像法，对于度数较高的顶点，在多台机器中都存放镜像节点，这样所有的消息传递都将在本地进行，只需要将镜像节点同步即可。这种镜像法将边之间可能的大量通讯转移到了机器之间，由于机器的数量一般不会很高，所以镜像节点的同步速度是非常快的。镜像法除了可以减少通讯量，还可以减少由于高度数节点所带来的负载不均衡。

由于 Pregel+ 平台仅在底层优化，而没有改变顶层的接口设计，因此大量基于 Pregel 的算法可以非常方便地移植到 Pregel+ 平台。Pregel+ 的核心代码仅包含一个 compute 函数，这里给出我们的实现：

```
compute(vertex u):
    new_dist = u.dist
    for dist in message:
        if dist < new_dist:
            new_dist = dist
    if new_dist < u.dist:
        u.dist = new_dist
        for [u, v, w] in edges[u]:
            send_message(v, u.dist + w)
```

4.4.5 Grape 平台

Grape 平台是一个以块为中心的计算平台，专门用于优化最短路径这一类和图的直径直接有关的问题。Grape 包含两个用户函数，PartialEvaluation（PEval）和 IncrementalComputation（IncEval），分别用于块内的初始化计算，以及经过同步后的增量计算。

以最短路径为例，PEval 函数负责在块内部调用 Dijkstra 算法计算当前的最短路径，即仅用当前块内的边能组合出的最短路径：

```
PEval(vertex_set V, edge_set E):
    dist[source] = 0
    initialize PQ as a priority queue
    PQ.push(source, 0)
    while PQ is not empty:
        u = PQ.pop()
        for [u, v, w] in E:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                PQ.push(v, dist[v])
    initialize M as a set
    for v in V:
        M.push([v, dist[v]])
    return M
```

接下来，Grape 会不断调用 IncEval 直至收敛。具体来说，每个块会收到来自其他块的最短路径，然后尝试用这些最短路径进行松弛操作：

```
IncEval(vertex_set V, edge_set E, message M):
    initialize PQ as a priority queue
    for [v, dist[v]] in M:
        PQ.push(v, dist[v])
    while PQ is not empty:
        u = PQ.pop()
        for [u, v, w] in E:
            if dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                PQ.push(v, dist[v])
    initialize M as a set
    for v in V:
        M.push([v, dist[v]])
    return M
```

Grape 这种以块为中心的计算模式介于 Dijkstra 算法和 Bellman-Ford 算法之间：前者通过最优性保证每个节点只会被计算一次，而后者则由于自由的计算顺序导致最坏情况下会产生大量重复计算。Grape 算法在块内部保证最优，但是块之间的同步会产生一定的重复计算。具体来说，如果一条最短路径穿越了 k 次块边界，那么这条最短路径就需要 k 个超步才能会被计算到。

当然，以块为中心的计算模式最大的意义是给了一些难以并行的顺序算法一种通用的并行可能，即只需要满足增量更新的特点就可以实现并行。

4.5 分布式环境下的最短路径算法效率

在这一章中，我们会将所有平台中的最短路径算法进行测试，以观察不同模式和不同平台下的计算效率差异。

4.5.1 测试环境

所有的实验均在一个高性能计算集群上执行，该集群由 16 台独立的物理机组成，提供了强大的并行计算能力。每台物理机含有四颗 Intel® Xeon® Platinum 8163

@ 2.50GHz 处理器，每颗处理器包含 24 个物理核心和 48 个线程，确保并行计算能力。

此外，考虑到不同的计算平台有不同的存储架构（基于内存或基于磁盘），每台物理器还配备了 512GB 的 DDR4 内存和 3TB 的硬盘空间（通过 Hadoop 组成存储集群），避免由于内存或硬盘空间不足导致的性能瓶颈。

为了保证各物理机之间的高效通信，这 16 台机器通过一个带宽为 15Gbps 的高速局域网相互连接。这种高带宽的网络环境对于分布式计算任务至关重要，尤其是那些需要大量数据通讯的平台。另一方面，带宽不可能无限增大，15Gbps 是一个较为合理的数值。

综上所述，这样一个配置完整的集群环境，能够同时满足大规模图数据分析所需的计算资源、存储资源和通讯资源，尽可能减少由于硬件问题所带来的差异，充分发挥出各个平台的设计性能。

4.5.2 测试数据集

为了尽量减少数据集对测试结果的影响，并确保实验结果的可靠性和可重复性，我们采取了一种系统化的方法来生成多样化且可控的人工合成数据集。这些数据集不仅在规模上有所区别，还涵盖了不同的图特征，以便全面评估算法在各种情况下的性能表现。

数据集的生成基于 Facebook 的社交网络分布，通过学习该网络的节点特征，我们生成了一系列数据集，这些数据集满足社交网络的幂律分布，即少量节点有着较高的度数，且满足小世界定理，即任意两个人之间的最短路径跳数不超过六。

数据集的命名包含规模和特征两部分。数据集的规模是通过其点数和边数来定义的，具体来说，假设图中由 n 个点和 m 条边，该数据集的规模参数定义为点数和边数之和以 10 为底的对数，即：

$$Scale = \log_{10}(n + m)$$

考虑到社交网络图的边数远大于点数，可以近似认为数据集的规模参数是边数的对数。

除此之外，我们还为数据集在两个参数上进行增强，包括：

- 稠密数据集。稠密数据集在边数不变的基础上减少了点数，使得密度增大了十倍。这模拟了现实世界中的社交网络或紧密合作的专业社区等场景，在这些环境中，个体之间存在着大量的直接联系。
- 大直径数据集。大直径数据集在图规模不变的基础上大大增加了直径，这意味着从起点出发需要更多的超步才能到终点。这对最短路径算法是一个巨大的挑战。

表 4.1　生成数据集及其参数

数据集	点数	边数	密度	直径
S8-Std	3.6M	153M	2.4×10^{-5}	6
S8-Dense	1.2M	159M	2.2×10^{-4}	5
S8-Diam	3.6M	155M	2.4×10^{-5}	101
S9-Std	27.2M	1.42B	3.8×10^{-6}	6
S9-Dense	9.1M	1.47B	3.6×10^{-5}	5
S9-Diam	27.2M	1.48B	4.0×10^{-6}	102

4.5.3　运行时间的对比

首先，我们在单机 32 线程下对比各个平台执行最短路径的时间，减少由于通讯所带来的影响。这种设置允许我们专注于算法本身的效率，而不必担心分布式环境中常见的网络延迟和其他通信开销。具体结果如图 4.4所示。

从三个数据集对比来看，大直径数据集所花费的时间远远多于其他两个数据集，这也和之前的理论分析相符合，即最短路径算法的执行时间受到图的直径影响，图的直径越多，计算平台就需要更多的超步执行，而每个超级步都需要同步操作，这些同步点会引入额外的等待时间，从而导致总体执行时间增加。另一方面，高密度的数据集显示出更短的执行时间。尽管稠密图中的点数较少，且每个节点有更多的邻居，但相关的计算可以更加集中和高效地进行，这种集中性减少了冗余计算，提高了处理速度。

从六个平台对比来看，Grape 和 Ligra 展现出了最快的执行速度，因为 Grape 就是为了解决最短路径问题之类顺序算法而设计的，使其在解决这类问题时表现出色，而 Ligra 的轻量级设计也让它在单机环境下能够快速响应并高效处理任务，因此它的执行速度也非常快；PowerGraph、Flash 和 Pregel+ 运行速度其次，但在大多数情况下仍能提供合理的执行时间；GraphX 是最慢的平台，这一结果主要归因于 Spark 平台自身的限制，包括较高的启动延迟、JVM 堆栈管理开销以及序列化/反序列化的成本等。

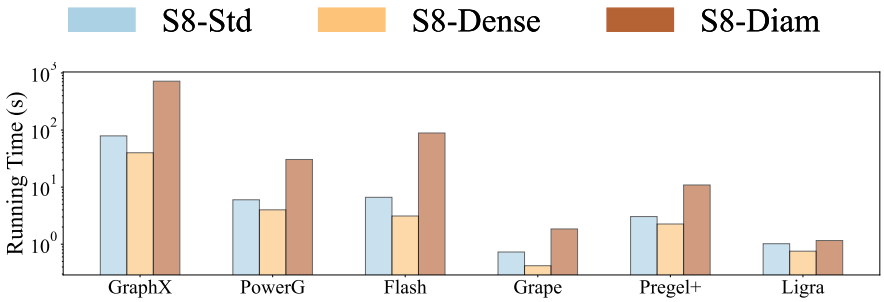


图 4.4　六个分布式计算平台下的三个不同数据集的最短路径算法执行时间。

4.5.4　扩展性的对比

为了全面评估图计算平台的扩展性，我们进行了详细的实验设计，分别测试了在不断增加线程数量（纵向扩展性）和不断增加机器数量（横向扩展性）情况下的运行时间变化。这些实验不仅帮助我们理解系统如何应对不同的计算资源配置，还揭示了影响其性能的关键因素。

纵向扩展性指的是通过增加单台机器的计算资源来提高其处理能力，在本实验中，即通过增加单机上的线程数量来进行测试。这种方法主要关注的是系统本身的计算性能以及算法的优化能力。一方面，纵向扩展性测试帮助我们了解平台如何有效地利用多核处理器的能力；另一方面，纵向扩展性测试体现了算法的并行程度以及在算法实现上的优化水平。这种设置允许我们专注于系统和算法本身的效率，而不必担心分布式环境中常见的网络延迟和其他通信开销。

与纵向扩展性相对应，横向扩展性是指通过增加机器数量来提高整个系统的计算能力。在本实验中，即通过逐步增加参与计算的机器数量来进行测试。这种方

法更加考验系统设计的能力，尤其是当增加机器数量时，有效地分配任务以确保各个节点之间的负载均衡，以及优化通信机制，选择合适的通信协议和拓扑结构，减少不必要的数据传输。

通过这种方式，我们可以更清晰地识别出哪些因素是限制系统性能的关键所在，并为后续的优化提供方向。

纵向扩展性的对比

纵向扩展性指的是在一台机器中增加线程所带来的加速。我们在单机上使用1、2、4、8、16、32 线程进行测试，其中 GraphX 至少需要 2 线程才能成功执行。

图 4.5 展示了不同线程数量下的最短路径算法执行时间，所有平台都展现出良好的纵向扩展性，即随着线程的增多，执行时间显著下降。详细的加速倍数，即最佳性能与单线程性能的比率，在表 4.2中列出。

表 4.2　纵向扩展性加速倍数

Dataset	GraphX	PowerG	Flash	Grape	Pregel+	Ligra
S8-Std	6.9	5.0	9.3	23.5	8.8	2.0
S8-Dense	7.8	5.8	8.5	19.7	11.3	2.4
S8-Diam	6.7	0.9	10.2	13.2	10.1	1.8

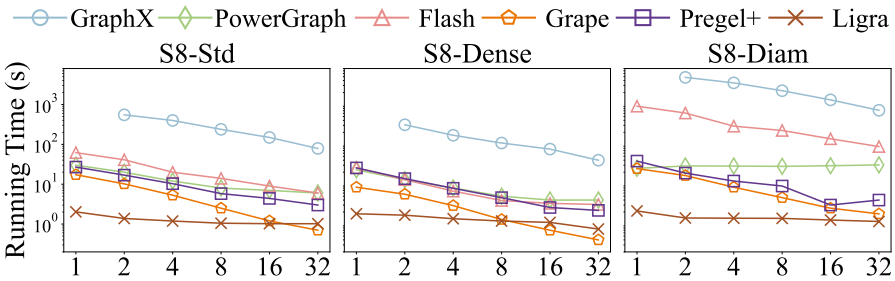


图 4.5　在单机不同线程数量下的最短路径算法执行时间。

整体来看，Grape 展现出较好的纵向扩展性，最高可达 23 倍，其次是 Pregel+、Flash、GraphX 和 PowerGraph，平均约为 5∼10 倍。特别地，Ligra 的纵向扩展性较差，这可能是因为 Ligra 平台本身的速度很快，而预处理时间由占据了较多的一部分，导致加速不明显。

从三个数据集来看，纵向扩展性在不同特性的数据集上也有很多差异。大多数平台在稠密数据集上有着更好的纵向扩展性，这是因为随着密度的增加，计算也会更加集中在更少的点上。然而，在大直径数据集上的扩展性一般较差，甚至 PowerGraph 平台呈现出负收益，这可能是由于超步数量快速增加而导致的。

横向扩展性的对比

横向扩展性指的是增加机器数量所带来的加速。我们在 1、2、4、8、16 台机器上进行测试，每台机器都使用了 32 线程。由于计算资源成倍增加，我们选用更大的数据集进行测试。其中 GraphX 无法在 S9-Diam 上完成计算，且至少需要 4 机才能在其他两个数据集上完成计算；Ligra 由于仅支持单机，所以没有被包含在实验内。

图 4.6 展示了不同机器数量下的最短路径算法执行时间，可以看到横向扩展性对比纵向扩展性较为一般，但随着机器的增多，执行时间整体呈现下降趋势。详细的加速倍数，即最佳性能与单机性能的比率，在表 4.3中列出。

表 4.3　横向扩展性加速倍数

Dataset	GraphX	PowerG	Flash	Grape	Pregel+
S9-Std	1.8	2.6	1.2	1.7	2.4
S9-Dense	2.2	2.9	1.3	3.3	3.1
S9-Diam	—	1.4	2.0	0.5	4.0

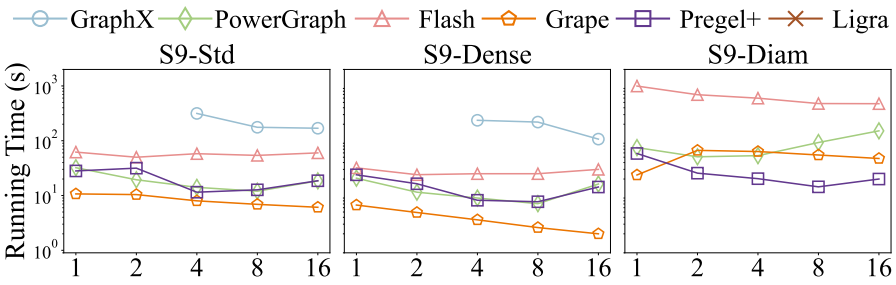


图 4.6　在不同机器数量下的最短路径算法执行时间。

所有平台的横向扩展性相比纵向扩展性都较差，这可能是由于机器间通信的巨大开销所致。此外，大部分平台在单机上已经已经实现了非常好的性能，扩展到

多台机器时，由于不可避免的通信开销，其性能增益趋于饱和。

而在不同数据集上的横向扩展性和纵向扩展性都较为类似，即在稠密数据集上表现更好，而在大直径数据集上表现较差，甚至同样会呈现出负收益。

4.5.5 平台性能对比与选取参考

从以上运行时间和扩展性对比的实验可以看出，不同的计算平台之间的差异较为明显，一方面受到平台本身语言和依赖的基础架构限制，另一方面也有本身计算模式之间的优劣，具体来说：

- 平台特性的影响。六个平台中除了 GraphX 使用 Scala 编写之外，其他平台均选择使用 C/C++ 语言，并使用 OpenMPI 或 MPICH 进行通讯，从实验结果也可以看出 C 系列语言高效的运行速度以及更加自由的编写模式。此外，基于内存存储的平台相比于基于硬盘存储的平台运行速度也有显著的提升，然而，代价是将数据导入内存需要消耗大量的时间，且内存的价格也是不可忽略的一部分。
- 计算模式的影响。在本实验中，我们选取的最短路径算法其实是一个比较复杂的问题，因为传统的 Dijkstra 无法很好的并行化，而经典的以点和边为中心的并行计算模式则只能采用最基础的 Bellman-Ford 算法，这就意味着虽然并行能够减少实际的运行时间，但是代价却是增加了大量的冗余计算。相比之下，以块为中心的计算模式则更加擅长解决这一类问题。

在实际应用中，则需要根据不同的落地需求综合考虑多个因素选取计算平台，包括：

- 应用场景。对于大规模的分布式计算，GraphX 这种带有全面容错机制的平台显然是最合适的，PowerGraph 和 Pregel+ 也在一定程度上提供了容错能力；而对于小规模的快速开发，单机的 Ligra 则更加合适。
- 性能需求。如果对性能有较高的要求，应优先考虑那些在运行时间测试和扩展性测试中表现出色的平台，如 Grape 和 PowerGraph，这些平台提供了极致的性能优化，能够最快地完成计算任务。
- 算法需求。由于分布式算法编写难度较高，如果用户需要其他更加复杂的图

算法，Flash 提供的大量图算法会带来帮助。并且 Flash 包含了大量在顶级会议上最新发表的高效算法，可以带来更高的效率。

- 兼容性需求。在一些情况下需要和已有数据和算法兼容，例如 GraphX 可以直接对接 Spark 处理后的数据，Pregel+ 也可以直接运行 Pregel 编写的代码，而无需过多的修改。

第五章　总结

在本文中，我们深度探讨了如何将核密度估计法应用于时空数据路网图上，并提出了一个搞笑的解决方案，即区间森林法（RFS）。RFS 算法通过采用基于内存共享的树形索引结构来同时维护数据点的时间信息和位置信息。这种设计使得它能够在高效处理多个时间窗口查询的同时，避免增加额外的内存开销。此外，通过树形结构组织和管理数据点，查询操作更加迅速，特别是在查询特定时间窗口内的核密度时，只需遍历相关的树节点即可获取所需的信息，大大提高了查询效率。

为了进一步提升 RFS 算法的灵活性和实用性，我们开发了动态区间森林法（DRFS）。DRFS 算法扩展了原始的 RFS 算法，通过引入动态结构以支持插入操作，这意味着系统可以在不影响现有查询性能的情况下动态添加新的数据点。更重要的是，DRFS 为用户提供了不同等级的量化参数，允许根据实际需求调整计算精度。这种灵活的设计允许用户根据应用场景的具体要求，在计算精度与计算时间之间找到最佳平衡点。例如，在需要快速响应的场景下，可以选择较低的精度以加快计算速度；而在对结果精确度有较高要求的场合，则可以适当提高量化级别。

除此之外，我们注意到相邻查询点之间的核密度值通常是平滑变化的，并且共享了大量的相似计算步骤。针对这一现象，我们设计了一种称为线段点共享（LS）的技术。LS 技术允许将相似的计算结果应用于多个查询点，从而减少不必要的重复计算，显著提升了整体计算效率。通过最大化利用已有的计算资源，LS 技术有效降低了整个系统的计算负担，使得大规模数据集的处理变得更加可行。

并且，我们的解决方案中的核函数是可以替换的，这意味着可以根据具体的分析目标选择最合适的核函数类型。除了传统的线性核函数外，我们的框架还支持更复杂的核函数，包括指数函数和余弦函数等。这为用户提供了一个更加灵活的选择空间，更好地适应不同的应用场景需求，确保结果的准确性和可靠性。

我们还设计了全面的实验流程来评测我们提出的算法的效率。性能实验包括在不同的带宽范围、查询次数、线段点精度、时间窗口大小参数下对比我们提出的算法和基线算法，有效性实验则是在不同的量化等级下对比 RFS 算法和 DRFS 算

法的差距。实验结果表明，RFS 算法相比于目前最优的算法和基线算法分别有 6 倍和 89 倍的性能提升，并且只增加了 30% 的额外内存开销；DRFS 算法则可以进一步减少 40% 的时间开销和 60% 的内存开销，且核密度误差不超过 5%。

考虑到传统的单机计算模式在处理大规模图数据时面临着显著的性能瓶颈，我们还将最短路径算法扩展到了分布式平台上，深入探索了这些算法的分布式计算潜力。分布式计算有两大优点：存储分布式和计算分布式。存储分布式可以将庞大的数据分散存储到不同机器上，避免由于数据过大而无法全部加载到内存的情况；计算分布式则可以将密集的计算任务分散到不同机器上，实现并行计算，提高计算效率。

由于目前还没有针对分布式图计算的全面总结和技术分类，我们先介绍了分布式图计算的相关背景，包括将图算法迁移到分布式环境中所需要的划分算法和计算模式。划分算法主要研究如何将完整的图切分成若干子图，存放到不同的机器上，并且减少机器之间的通讯所带来的开销；计算模式则是研究如何将计算任务分散到点、边、块或子图上，以实现更好的并行化。

针对这些不同的分布式部署特点，我们选取了六个最常用的且具有代表性的分布式计算平台，包括 GraphX、PowerGraph、Flash、Grape、Pregel+ 和 Ligra，这些平台涵盖了各种划分算法和计算模式，有着截然不同的计算特点。我们在这些平台上各自实现了最短路径算法，并在单机和多机环境下进行全面的多线程测试。所有分布式算法均展现了高效的计算流程，这给传统算法的并行化和提供了极大的便利，显示出巨大的应用潜力。

此外，考虑到分布式平台部署的时间成本较高，我们还根据实际的运行结果和部署经验，给出了分布式平台的选用指南。GraphX 作为 Spark 平台的扩展，能够较好的兼容已有的大数据，但基于 Hadoop 的存储方式和本身架构的局限性使其运行效率较低，相比之下 Pregel+ 作为 Pregel 的扩展，不仅代码可以兼容，且运行效率也较高；如果对性能较为敏感，则可以选择 Grape 和 PowerGraph，这两个平台都对计算效率做了大量的优化；如果对算法需求较高，则可以选择 Flash，Flash 提供了大量的算法，不仅包含传统的实现方式，还更新了许多最新论文中提出的优化算法。

最后，本论文依然存在一些不足之处。一方面，即使经过了大量的优化与加速，核密度的计算复杂度依然较高，距离实际应用在毫秒级别的导航类软件上还有很大的差距，并且额外的内存开销在边缘设备这类对资源极度敏感的场景下也是一个不可接受的负担；另一方面，我们发现分布式图计算的横向扩展性较差，多机环境下的表现较差，甚至可能会出现负提升。

针对这些问题，我们也提出了本工作在后续的改进路线。针对于核密度计算代价较为高昂的问题，我们将以 DRFS 算法为基础，进一步深入的研究近似算法，且以工业场景落地为目标导向，以用户实际体感为主要指标，淡化理论的近似比数值，同时在时间和内存两方面进行优化，针对分布式场景的横向扩展性较差的问题，需要进一步定位问题来源，可能是由于计算资源已经达到了算法并发度的瓶颈，那么就需要重新设计算法流程，甚至允许增加一部分计算开销来换取更高的并发度，也可能是多机环境本身通讯开销较大，可以选择换取更高效的通讯协议和数据传输格式，或是采用异步架构来减少通讯阻塞带来的损失。

参考文献

[1] SILVERMAN B W. Density estimation for statistics and data analysis[M]. [S.l.]: Routledge, 2018.

[2] GRAMACKI A. Nonparametric kernel density estimation and its computational aspects: Vol 37[M]. [S.l.]: Springer, 2018.

[3] DIEBOLD F X, HAHN J, TAY A S. Multivariate density forecast evaluation and calibration in financial risk management: high-frequency returns on foreign exchange[J]. Review of Economics and Statistics, 1999, 81(4): 661 – 673.

[4] DIEBOLD F X, GUNTHER T A, TAY A S. Evaluating Density Forecasts with Applications to Financial Risk Management[J/OL]. International Economic Review, 1998, 39(4): 863 – 883.

[5] HARVEY A, ORYSHCHENKO V. Kernel density estimation for time series data[J]. International journal of forecasting, 2012, 28(1): 3 – 14.

[6] BRUNSDON C, CORCORAN J, HIGGS G. Visualising space and time in crime patterns: A comparison of methods[J]. Computers, environment and urban systems, 2007, 31(1): 52 – 75.

[7] NAKAYA T, YANO K. Visualising crime clusters in a space-time cube: An exploratory data-analysis approach using space-time kernel density estimation and scan statistics[J]. Transactions in GIS, 2010, 14(3): 223 – 239.

[8] HART T, ZANDBERGEN P. Kernel density estimation and hotspot mapping: Examining the influence of interpolation method, grid cell size, and bandwidth on crime forecasting[J]. Policing: An International Journal of Police Strategies & Management, 2014, 37(2): 305 – 323.

[9] BLACK W R. Highway accidents: a spatial and temporal analysis[J]. Transportation Research Record, 1991, 1318: 75 – 82.

[10] XIE Z, YAN J. Kernel density estimation of traffic accidents in a network space[J].

Computers, environment and urban systems, 2008, 32(5) : 396 – 406.

[11] PLUG C, XIA J C, CAULFIELD C. Spatial and temporal visualisation techniques for crash analysis[J]. Accident Analysis & Prevention, 2011, 43(6) : 1937 – 1946.

[12] FLEURET F, SAHBI H, others. Scale-invariance of support vector machines based on the triangular kernel[C] // 3rd International Workshop on Statistical and Computational Theories of Vision. 2003 : 1 – 13.

[13] GONG W, YANG D, GUPTA H V, et al. Estimating information entropy for hydrological data: One-dimensional case[J]. Water Resources Research, 2014, 50(6) : 5003 – 5018.

[14] SAMIUDDIN M, EL-SAYYAD G. On nonparametric kernel density estimates[J]. Biometrika, 1990, 77(4) : 865 – 874.

[15] BíL M, ANDRášIK R, JANOšKA Z. Identification of hazardous road locations of traffic accidents by means of kernel density estimation and cluster significance evaluation[J]. Accident Analysis & Prevention, 2013, 55 : 265 – 273.

[16] SCHOLKOPF B, SUNG K-K, BURGES C J, et al. Comparing support vector machines with Gaussian kernels to radial basis function classifiers[J]. IEEE transactions on Signal Processing, 1997, 45(11) : 2758 – 2765.

[17] KRISTAN M, LEONARDIS A, SKOčAJ D. Multivariate online kernel density estimation with Gaussian kernels[J]. Pattern Recognition, 2011, 44(10-11) : 2630 – 2642.

[18] DE FELICE M, PETITTA M, RUTI P M. Short-term predictability of photovoltaic production over Italy[J]. Renewable Energy, 2015, 80 : 197 – 204.

[19] ArcGIS: https://pro.arcgis.com/[EB]. 2023.

[20] QGIS: https://docs.qgis.org/[EB]. 2023.

[21] CHAN T N, IP P L, U L H, et al. KDV-Explorer: A near real-time kernel density visualization system for spatial analysis[J]. Proceedings of the VLDB Endowment, 2021, 14(12) : 2655 – 2658.

[22] GAN E, BAILIS P. Scalable kernel density classification via threshold-based prun-

ing[C] // Proceedings of the 2017 ACM International Conference on Management of Data. 2017 : 945 – 959.

[23] CHAN T N, IP P L, U L H, et al. SAFE: a share-and-aggregate bandwidth exploration framework for kernel density visualization[J]. Proceedings of the VLDB Endowment, 2021, 15(3) : 513 – 526.

[24] CRISTIANINI N, CAMPBELL C, SHAWE-TAYLOR J. Dynamically adapting kernels in support vector machines[J]. Advances in neural information processing systems, 1998, 11.

[25] ROMANO B, JIANG Z. Visualizing traffic accident hotspots based on spatial-temporal network kernel density estimation[C] // Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2017 : 1 – 4.

[26] CHAN T N, IP P L, U L H, et al. SWS: a complexity-optimized solution for spatial-temporal kernel density visualization[J]. Proceedings of the VLDB Endowment, 2021, 15(4) : 814 – 827.

[27] BORRUSO G. Network density estimation: analysis of point patterns over a network[C] // International Conference on Computational Science and Its Applications. [S.l.] : Springer, 2005 : 126 – 132.

[28] CHAN T N, LI Z, U L H, et al. Fast augmentation algorithms for network kernel density visualization[J]. Proceedings of the VLDB Endowment, 2021, 14(9) : 1503 – 1516.

[29] CHAN T N, CHENG R, YIU M L, et al. Efficient algorithms for kernel aggregation queries[J]. IEEE Transactions on Knowledge and Data Engineering, 2020.

[30] CHAN T N, CHENG R, YIU M L. QUAD: Quadratic-bound-based kernel density visualization[C] // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020 : 35 – 50.

[31] CHAN T N, YIU M L, LEONG H U. KARL: Fast kernel aggregation queries[C] // 2019 IEEE 35th International Conference on Data Engineering (ICDE). [S.l.] :

IEEE, 2019 : 542 – 553.

[32] SILVERMAN B W. Algorithm AS 176: Kernel density estimation using the fast Fourier transform[J]. Journal of the Royal Statistical Society. Series C (Applied Statistics), 1982, 31(1) : 93 – 99.

[33] AUBER D, JOURDAN F. Interactive refinement of multi-scale network clusterings[C] // Ninth International Conference on Information Visualisation (IV'05). [S.l.] : IEEE, 2005 : 703 – 709.

[34] ABELLO J, VAN HAM F, KRISHNAN N. Ask-graphview: A large scale graph visualization system[J]. IEEE transactions on visualization and computer graphics, 2006, 12(5) : 669 – 676.

[35] HINNEBURG A, GABRIEL H-H. Denclue 2.0: Fast clustering based on kernel density estimation[C] // International symposium on intelligent data analysis. [S.l.] : Springer, 2007 : 70 – 80.

[36] LIU Z, JIANG B, HEER J. imMens: Real-time visual querying of big data[C] // Computer Graphics Forum : Vol 32. [S.l.] : Wiley Online Library, 2013 : 421 – 430.

[37] LI C, BACIU G, HAN Y. Interactive visualization of high density streaming points with heat-map[C] // 2014 International Conference on Smart Computing. [S.l.] : IEEE, 2014 : 145 – 149.

[38] GRAY A G, MOORE A W. Nonparametric density estimation: Toward computational tractability[C] // Proceedings of the 2003 SIAM International Conference on Data Mining. [S.l.] : SIAM, 2003 : 203 – 211.

[39] FAN J, MARRON J S. Fast implementations of nonparametric curve estimators[J]. Journal of computational and graphical statistics, 1994, 3(1) : 35 – 56.

[40] PATUELLI R, REGGIANI A, GORMAN S P, et al. Network analysis of commuting flows: A comparative static approach to German data[J]. Networks and Spatial Economics, 2007, 7 : 315 – 331.

[41] MOURATIDIS K, YIU M L, PAPADIAS D, et al. Continuous nearest neighbor mon-

itoring in road networks[J], 2006.

[42] SASIKALA I, GANESAN M, JOHN A. Uncertain data prediction on dynamic road network[C] // International Conference on Information Communication and Embedded Systems (ICICES2014). 2014 : 1 – 4.

[43] CHENG T, HAWORTH J, WANG J. Spatio-temporal autocorrelation of road network data[J]. Journal of Geographical Systems, 2012, 14 : 389 – 413.

[44] KOUDAS N, OOI B C, TAN K-L, et al. Approximate NN queries on streams with guaranteed error/performance bounds[C] // Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. 2004 : 804 – 815.

[45] FIGUEIRAS P, HERGA Z, GUERREIRO G, et al. Real-time monitoring of road traffic using data stream mining[C] // 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC). 2018 : 1 – 8.

[46] LI T, CHEN L, JENSEN C S, et al. TRACE: Real-time compression of streaming trajectories in road networks[J]. Proceedings of the VLDB Endowment, 2021, 14(7) : 1175 – 1187.

[47] VALIANT L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8) : 103 – 111.

[48] MCCUNE R R, WENINGER T, MADEY G. Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing[J]. ACM Computing Surveys (CSUR), 2015, 48(2) : 1 – 39.

[49] MALEWICZ G, AUSTERN M H, BIK A J, et al. Pregel: a system for large-scale graph processing[C] // Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010 : 135 – 146.

[50] YAN D, CHENG J, LU Y, et al. Effective techniques for message reduction and load balancing in distributed graph computation[C] // Proceedings of the 24th International Conference on World Wide Web. 2015 : 1307 – 1317.

[51] LI X, MENG K, QIN L, et al. Flash: A Framework for Programming Distributed Graph Processing Algorithms[C] // 2023 IEEE 39th International Conference on

Data Engineering (ICDE). 2023 : 232 – 244.

[52] SHUN J, BLELLOCH G E. Ligra: a lightweight graph processing framework for shared memory[C] // NICOLAU A, SHEN X, AMARASINGHE S P, et al. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013. [S.l.] : ACM, 2013 : 135 – 146.

[53] STUTZ P, BERNSTEIN A, COHEN W. Signal/collect: graph algorithms for the (semantic) web[C] // The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9. 2010 : 764 – 780.

[54] YAN D, CHENG J, XING K, et al. Pregel algorithms for graph connectivity problems with performance guarantees[J]. Proceedings of the VLDB Endowment, 2014, 7(14) : 1821 – 1832.

[55] GONZALEZ J E, XIN R S, DAVE A, et al. {GraphX}: Graph processing in a distributed dataflow framework[C] // 11th USENIX symposium on operating systems design and implementation (OSDI 14). 2014 : 599 – 613.

[56] GONZALEZ J E, LOW Y, GU H, et al. {PowerGraph}: Distributed {Graph-Parallel} computation on natural graphs[C] // 10th USENIX symposium on operating systems design and implementation (OSDI 12). 2012 : 17 – 30.

[57] ROY A, MIHAILOVIC I, ZWAENEPOEL W. X-stream: Edge-centric graph processing using streaming partitions[C] // Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013 : 472 – 488.

[58] KYROLA A, BLELLOCH G, GUESTRIN C. {GraphChi}:{Large-Scale} graph computation on just a {PC}[C] // 10th USENIX symposium on operating systems design and implementation (OSDI 12). 2012 : 31 – 46.

[59] ROY A, BINDSCHAEDLER L, MALICEVIC J, et al. Chaos: Scale-out graph processing from secondary storage[C] // Proceedings of the 25th Symposium on Operating Systems Principles. 2015 : 410 – 424.

[60] YAN D, CHENG J, LU Y, et al. Blogel: A block-centric framework for distributed

computation on real-world graphs[J]. Proceedings of the VLDB Endowment, 2014, 7(14): 1981–1992.

[61] FAN W, XU J, WU Y, et al. GRAPE: Parallelizing sequential graph computations[C] // 43rd International Conference on Very Large Data Bases. 2017: 1889–1892.

[62] YAN D, BU Y, TIAN Y, et al. Big graph analytics platforms[J]. Foundations and Trends® in Databases, 2017, 7(1-2): 1–195.

[63] YAN D, YUAN L, AHMAD A, et al. Systems for Scalable Graph Analytics and Machine Learning: Trends and Methods[C] // Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2024: 6627–6632.

[64] TEIXEIRA C H, FONSECA A J, SERAFINI M, et al. Arabesque: a system for distributed graph mining[C] // Proceedings of the 25th Symposium on Operating Systems Principles. 2015: 425–440.

[65] DIAS V, TEIXEIRA C H, GUEDES D, et al. Fractal: A general-purpose graph pattern mining system[C] // Proceedings of the 2019 International Conference on Management of Data. 2019: 1357–1374.

[66] MAWHIRTER D, WU B. Automine: harmonizing high-level abstraction and high performance for graph mining[C] // Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019: 509–523.

[67] JAMSHIDI K, MAHADASA R, VORA K. Peregrine: a pattern-aware graph mining system[C] // Proceedings of the Fifteenth European Conference on Computer Systems. 2020: 1–16.

[68] YAN D, GUO G, CHOWDHURY M M R, et al. G-thinker: A distributed framework for mining subgraphs in a big graph[C] // 2020 IEEE 36th International Conference on Data Engineering (ICDE). 2020: 1369–1380.

[69] RAKSHIT S, BADDELEY A, NAIR G. Efficient code for second order analysis of events on a linear network[J]. Journal of Statistical Software, 2019, 90: 1–37.

[70] DE BERG M. Computational geometry: algorithms and applications[M]. [S.l.]: Springer Science & Business Media, 2000.

致　　谢

攻读硕士学位期间发表论文和科研情况

■ 已公开发表的论文

- Yu Shao, Peng Cheng, Longbin Lai, Long Yuan, Wangze Ni, and Xuemin Lin. Most Probable Maximum Weighted Butterfly Search. The 41st IEEE International Conference on Data Engineering (ICDE 2025).
- Yu Shao, Lingkai Meng, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Jingren Zhou, and Xuemin Lin. Revisiting Graph Analytics Benchmarks. SIGMOD International Conference on Management of Data (SIGMOD 2025).
- Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Xuemin Lin, and Jingren Zhou. A Survey of Distributed Graph Algorithms on Massive Graphs. ACM Computing Surveys (CSUR 2024).
- Qinzhou Xiao, Yu Shao, Peng Cheng, Lei Chen, Wangze Ni, Wenjie Zhang, Hengtao Shen, Xuemin Lin, and Liping Wang. Time-Optimal Route Planning for Non-Linear Recharging Electric Vehicles on Road Networks. The 30th International Conference on Database Systems for Advanced Applications (DASFAA 2025).

■ 已发表的专利

- 一种基于基于核密度估计的路网图时空数据可视化系统及方法