2024 年全国大学生信息安全竞赛

作品报告

作品名称：

Time-Restricted, Verifiable, and Efficient Query Processing over Encrypted

电子邮箱：**xxx@qq.com**

提交日期：**2025.0x.11**

填写说明

1. 所有参赛项目必须为一个基本完整的设计。作品报告书旨在能够清晰准确地阐述（或图示）该参赛队的参赛项目（或方案）。

2. 作品报告采用 A4 纸撰写。除标题外，所有内容必需为宋体、小四号字、1.5 倍行距。

3. 作品报告中各项目说明文字部分仅供参考，作品报告书撰写完毕后，请删除所有说明文字。(本页不删除)

4. 作品报告模板里已经列的内容仅供参考，作者可以在此基础上增加内容或对文档结构进行微调。

5. 为保证网评的公平、公正，作品报告中应避免出现作者所在学校、院系和指导教师等泄露身份的信息。

目录

图 目录

表 目录

摘要

随着云计算的发展，数据存储在云上的比例不断增加，但数据安全和隐私问题也日益突出。传统的加密方法虽然能保护数据安全，但在查询效率和验证结果的正确性方面存在不足。为此，本项目旨在解决云环境中加密数据的安全高效查询问题，提出了一种时间限制的、可验证的、有效的加密数据查询处理方案（TiveQP），其主要功能和特性如下：

1. 时间限制访问：通过将空间属性和时间属性进行编码，支持用户在特定时间范围内访问数据。
2. 查询结果验证：采用补集验证机制，通过返回比特段和 HMAC 来验证查询结果的正确性和完整性，而不泄露未查询数据的隐私。
3. 高效查询处理：设计了剪枝策略，通过分类构建子树来减少不必要的搜索路径，显著提高查询处理速度和结果验证效率。

本项目将空间编码技术与时间编码结合，实现了时间限制的加密数据访问控制；提出了补集的概念，通过验证数据项是否在查询范围的补集中，避免了未查询数据的隐私泄露问题；设计了优化的 TiveTree 索引结构和剪枝策略，显著提升了查询处理和结果验证的效率。

该项目作品具有广泛的应用场景，适用于需要安全查询处理的云存储应用，如医疗数据、金融数据等敏感信息的管理。通过本方案，数据用户可以方便地进行安全查询，并验证查询结果的正确性和完整性，确保数据访问的可靠性。同时，在保证安全性的前提下，显著提高了查询处理速度和验证效率，适应大数据环境下的实际需求，具有很高的实用价值和应用前景。

关键字： 云计算　隐私保护　时空编码　补集验证　查询优化　TiveTree 索引

Abstract

With the development of cloud computing, the proportion of data stored in the cloud is continuously increasing, but data security and privacy issues are also becoming more prominent. Traditional encryption methods can protect data security but have deficiencies in query efficiency and result verification accuracy. To address this, this project aims to solve the problem of secure and efficient query processing for encrypted data in a cloud environment. We propose a time - restricted, verifiable, and efficient encrypted data query processing scheme (TiveQP), which has the following main features:

1. Time - Restricted Access: By encoding spatial and temporal attributes, it supports user access to data within specific time frames.
2. Query Result Verification: Using a complementary set verification mechanism, it verifies the correctness and completeness of query results by returning bit segments and HMAC, without leaking the privacy of unqueried data.
3. Efficient Query Processing: A pruning strategy is designed, which reduces unnecessary search paths by classifying and constructing subtrees, significantly improving query processing speed and result verification efficiency.

The innovations of this project include combining spatial encoding techniques with temporal encoding to achieve time - restricted access control for encrypted data; introducing the concept of a complementary set, which avoids the privacy leakage of unqueried data by verifying whether data items are in the complementary set of the query range; and designing an optimized TiveTree index structure and pruning strategy, significantly enhancing query processing and result verification efficiency.

This project has a wide range of applications, suitable for cloud storage applications that require secure query processing, such as the management of sensitive information like medical data and financial data. Through this scheme, data users can conveniently perform secure queries and verify the correctness and completeness of the query results, ensuring the reliability of data access. At the same time, while ensuring security, it significantly improves query processing speed and verification efficiency, meeting the practical needs of big data environments, and has high practical value and application prospects.

Keywords: Cloud Computing; Privacy Protection; Spatio - temporal Encoding; Complementary Set Verification; Query Optimization; TiveTree Index;

第一章 作品概述

本章分为背景分析、相关工作、研究问题、特色描述与应用前景分析五个部分，旨在对本时间受限可验证查询方案系统进行整体的介绍。

1.1 背景分析

在数字化浪潮席卷全球的当下，云计算与位置服务（LBS）的深度融合正重塑人类社会的运行逻辑。截至 2025 年，全球云计算市场规模突破 7,000 亿美元，LBS 日活用户超 50 亿，支撑着从实时导航到元宇宙空间定位的多元化场景。然而，技术赋能的另一面是愈发尖锐的隐私博弈——用户的位置轨迹、行为偏好等数据成为商业竞争与国家安全的新战场。苹果公司 2024 年推出的“私密云计算”（PCC）技术，通过无持久存储设计、硬件加密密钥随机生成及算后即焚机制，为云端 AI 处理构建了封闭的隐私保护屏障，其服务器操作系统经第三方审计验证，确保数据无法被追溯至具体用户。这一创新既是对传统云服务架构的颠覆，也映射出行业对隐私保护的技术焦虑。

全球数据主权意识的觉醒正催生严苛的监管矩阵。欧盟《人工智能法案》（2024 生效）首次将 LBS 算法偏见纳入审计范围，要求公开数据处理逻辑的可解释性；中国《个人信息保护法》则通过芯片级可信计算强化全链条安全要求。跨国公司深陷合规泥潭：Meta 因违规迁移欧盟用户数据被罚 12 亿欧元，暴露全球化服务与本地化立法的根本冲突；滴滴出行因未审计行程数据共享触发监管下架，凸显企业内部控制的系统性风险。这种监管与技术之间的张力，在跨境数据流动中尤为显著——美国 CLOUD 法案与欧盟 GDPR 的管辖权冲突，迫使企业采用“数据本地化孤岛”策略，但由此产生的运营成本与技术碎片化正成为数字经济的新桎梏。

技术防御与攻击的“军备竞赛”已进入量子级对抗阶段。传统加密体系在 IBM 量子计算机的威胁下岌岌可危，后量子加密算法成为云服务商升级焦点。微软 Azure SQL Database 通过同态加密实现毫秒级加密态检索，阿里云则构建三级认证体系，从基础知识到高级应用覆盖全链条安全能力。值得警惕的是，攻击手段正从数据窃取转向时空关联分析——2024 年英国 NHS 新冠追踪 APP 因哈希算法缺陷导致用户身份反推，揭示匿名化技术在行为模式识别中的脆弱性。这种风险在元宇宙场景中呈指数级放大，IDC 预测 2030 年 XR 设备位置数据达 80ZB，虚拟空间的行为轨迹可能成为现

实身份破解的密钥。

未来十年，隐私保护范式将经历从“数据控制”到“算法治理”的范式迁移。苹果 PCC 技术中“透明日志验证”机制，华为云的三维点云定位专利，均指向算法透明度的新战场。当瑞士 SkyMedical 急救无人机通过保形加密技术兼顾 3 秒响应与位置隐匿，当马士基物流区块链用零知识证明验证货物真实性，我们看到的不仅是技术突破，更是隐私、效率与成本“不可能三角”的突围尝试。这场关乎数字文明基石的博弈，终将在量子计算与元宇宙的叠加效应中，书写新的技术伦理篇章。

图 1.1　关于防范和打击电信网络诈骗犯罪的通告

1.2 相关工作

本节将回顾安全 k 近邻（SkNN）查询处理和隐私保护范围查询处理的相关研究。

(1)SkNN 查询处理

雷等人 [?] 提出了一种名为 SecEQP 的安全高效查询协议。该协议通过一组基础投影函数将位置转换为多个可行区域，再将这些区域整合成不规则多边形。云服务器通过对比两个位置的复合投影函数输出代码是否相等来判断其接近程度，并将代码存入不可区分布隆过滤器（IBF）构建 IBF 树。数据用户生成投影函数代码作为查询令牌，云服务器通过搜索 IBF 树完成查询。但该方案无法实现结果验证。

崔等人 [?] 设计了可验证的安全 k 近邻协议 SVkNN。该方案采用 Voronoi 图划分区域，通过可验证安全索引（VSI）支持快速查询验证。虽然提出了紧凑验证方法，但需依赖双服务器架构，导致额外的计算和通信开销。我们认为，大多数用户仅需在常规区域进行查询，无需处理复杂形状区域（如不规则多边形和 Voronoi 单元），且现有方法需用户预处理整个区域，效率较低。因此，本文设计了一种轻量级空间编码技术，以减少用户预处理时间和服务器索引搜索时间。

(2) 隐私保护范围查询处理

李等人 [?] 首次提出支持选择关键字攻击下索引不可区分的范围查询协议。该方案通过前缀编码转换数据项，将哈希随机化后的前缀存入布隆过滤器构建 PB 树。用户将查询范围转换为最小前缀集生成陷门，服务器通过搜索 PB 树匹配查询。在此基础上，李等人 [?] 进一步提出支持联合查询的隐私保护协议，采用孪生布隆过滤器（IBF）实现自适应安全，并构建 IBF 树替代传统 PB 树。

吴等人 [?] 提出的 ServeDB 协议支持加密数据多维范围查询，通过树索引验证机制确保结果正确性。但该方案返回的验证证明包含叶节点布隆过滤器，可能导致用户通过布隆过滤器查询其他范围，造成隐私泄露问题。

1.3 研究问题

在 SNN 框架下处理云位置数据时，面临着一系列挑战：

(1) 首先，在处理云位置数据时，一个迫切需要解决的问题是如何有效地应对用户对时间限制访问的需求。传统的位置查询服务往往只考虑了空间距离，但随着用户需求的不断演变，例如需要在特定时间范围内获取信息，这种情况变得越来越普遍。因此，我们需要探索如何在位置查询中集成时间限制，以便向用户提供更精确和个性化的服务。这需要在云计算数据处理中引入更多复杂性，并且需要设计和实施新的算法和技术来满足这种需求。

(2) 其次，在处理云上位置数据时，保护数据所有者和用户的隐私是一个至关重要的问题。位置数据涉及用户的日常活动、行为习惯甚至健康状况等敏感信息。因此，确保这些数据不被未授权的访问或滥用至关重要。为了解决这一问题，需要采取一系列的隐私保护措施，包括数据加密、访问控制和隐私保护协议的设计。

(3) 尽管现有的位置查询方案可能具有一定的结果验证性，但在保护隐私方面仍然存在着不足之处。用户可能通过分析查询结果来获取未查询数据的隐私信息，从而导致数据泄露和隐私侵犯的风险。因此，我们需要设计更加严格和可靠的隐私保护机制，确保即使在验证结果的过程中也不会泄露用户的隐私信息。这涉及到使用更加安全的加密算法、隐私保护技术的创新以及强化隐私保护协议的实施等方面。

(4) 最后，当前的位置查询方案往往忽视了数据项的空间属性，导致了查询效率的下降。然而，通过充分利用空间属性，可以设计出更加高效的查询处理方法，从而提高系统的整体性能和用户体验。这涉及到设计新的数据结构和算法来处理空间属性、优化查询处理过程以减少查询时间，以及引入更加智能化的查询优化策略等方面。通过这些措施，我们可以更好地应对处理云上位置数据时面临的挑战，并提高系统的整

体性能和用户满意度。

为解决上述问题，提出的方案是 TiveQP 方案。TiveQP 方案旨在解决位置数据隐私问题，特别是针对基于位置的服务中的隐私保护时间限制访问，与现有的 SNN 隐私和隐私保护范围查询方案相比，TimevQ 方案具有三个主要优势：（1）支持隐私保护时间限制访问；（2）采用更强的威胁模型实现隐私保护结果验证；（3）通过利用属性提高查询效率。通过这些优势，TimevQ 方案能够更好地满足数据所有者和用户的需求，提供更安全和高效的位置服务。为了达成这些目标，我们的项目方案面临着以下三个挑战：

如何实现具有安全的时间限制访问的 SNN 查询处理？空间转换和编码技术通常用于解决位置查询问题。安全的时间限制访问本质上是一个隐私保护的范围查询问题。一种直接的方法是分别处理空间属性和时间属性。然而，这样做会导致效率低下。我们必须将这两个问题整合起来，并统一执行查询。为了解决这个问题，我们使用空间编码技术来处理位置，并利用前缀编码来处理位置和时间，将所有生成的数据项的码都投影到一个不可区分的布隆过滤器（IBF）中。即，一个整合索引。每个查询都被编码成一组密钥门。通过这种方式，我们将这两个问题整合起来。

如何在现有的 SNN 框架下实现结果验证而不违反结果隐私？这个领域的开创性论文是 ServrDB[7]，作者提出了一种有效的方法，可以验证其结果的正确性和完整性，而不需要引入新的结构。对于完整性，云服务生成与数据用户发现每个步骤相关的搜索路径的证明信息。不幸的是，一些关键点，即叶节点，包含在返回的证明中。这允许数据用户在节点上自由查询任何集合，从而违反了未查询数据项的隐私。为了解决这个问题，我们提出了零知识集的概念，我们不证明值位置不在区域 A 中，而是证明它位于 A 的补区域中。具体来说，数据所有者为每个数据项计算了位置、类型和开放时间的三个补集合。每个集合都转换为最小的前缀集合。每个前缀都被查询节点 IBF 中生成一系列比特段。此外，对比特段计算出一个带有哈希的哈希消元返回码（IMAC）作为签名。当证明与匹配时，我们将匹配的比特段及其 IMAC 返给数据用户，而不是返整个 IBF 段。

如何在概率性安全性的情况下进一步提高查询效率？在概率性安全性的前提下提高查询效率是一项艰巨的任务 [19], [5]。为了提高效率，在创建索引的操作 TimevQ 时，我们设计了一种修剪策略，首先根据它的空间属性（例如类型和城市）对数据项进行分类，然后从底部向上为每种类型的位置构建子树。通过这样做，我们消除了不必要的搜索路径，减少了查询处理时间和结果验证时间。查询复杂性降低到 $\mathbf{O}(k\log(\frac{n}{t\times c}))$，其中 t 和 c 分别是类型和城市的数量。

subsection 特色描述 我们提出的解决方案 TiveQP（时域受限可验证高效查询处理, Time-Restricted, Verifiable, and Efficient Query Processing over Encrypted Data on Cloud），能够在云环境中抵御恶意云服务器的攻击，使得数据用户可以在不泄露其查询内容和位置隐私的情况下进行高效的查询和数据处理。系统架构如图 1 所示，包含三个核心实体：

(1) 扩展 SKNN 的时间受限查询。TiveQP 系统的第一个特色是扩展了传统的安全 k 最近邻（Secure k Nearest Neighbor, SkNN）查询，允许用户在特定时间段内查询 k 个最近的位置。这种时间受限的访问控制对用户来说非常重要。我们假定由一位患者需要医院的主刀医生，那么这位患者在提供地点信息外还要加上时间限制，预约医生的空闲时间段。

(2) 安全抵抗强化威胁模型。TiveQP 采用了更强的威胁模型来验证自身的安全性。在这种假设的强化威胁模型中，云服务器 (CS) 是恶意的。即安全系统不仅需要抵抗来自外界的攻击还要防范来自云服务器的可能窃取。但在 TiveQP 模型下，数据用户仍然可以验证查询结果的正确性，而不会泄露未查询的数据项的信息。这种安全性是通过补集验证技术实现的，即证明某数据项“在”某集合中的方式被转化为证明其“在”补集中。TiveQP 模型的安全性也得以增强。

(3) 高效查询和验证。为了提高查询处理和结果验证的效率，TiveQP 设计了空间编码技术和剪枝策略。这些技术显著减少了查询处理时间和验证时间。例如，在测试 TiveQP 中，我们在处理 Yelp 数据集时，查询 10 万个数据项中的前 10 个最近邻仅需 10 毫秒，验证时间仅为 1.4 毫秒。

1.4 应用前景分析

TiveQP 模型在时间限制、可验证和高效的加密数据查询处理方面展示了其独特的优势，具有广阔的应用前景。通过在云环境中实现安全、高效的查询处理，TiveQP 不仅适用于多种实际应用场景，还显示了其在技术和市场上的巨大潜力。以下将从实用性、技术可行性和市场可行性三个角度详细分析 TiveQP 的应用前景。

(1) 实用性

TiveQP 在多个实际应用场景中具有重要的实用性：

- 智能交通系统：在智能交通系统中，司机可以通过 TiveQP 查询特定时间段内开放的加油站、修理厂或停车场，从而更好地规划出行路线，避免拥堵。
- 医疗预约系统：TiveQP 可以帮助患者在不泄露隐私的情况下，查询特定时间段内

可用的医生或医疗资源，提高医疗服务的效率和用户体验。

- 隐私保护的数据查询：TiveQP 可以应用于金融数据查询、敏感地理信息服务等需要高隐私保护的场景，确保用户数据的安全和隐私。

(2) 技术可行性

TiveQP 需要依赖物联网、云计算和大数据分析等技术。随着互联网技术的不断发展和进步，这些技术已经具备了商业化的可行性。例如，物联网技术已经实现了万物互联，云计算已经能够提供高效、安全和低成本的数据存储和处理服务，而大数据分析则能够从海量数据中提取有价值的信息。此外，随着 5G 技术的普及和应用，TiveQP 应用的实时性和可靠性将得到进一步提高。

(3) 市场可行性

随着大数据和云计算技术的普及，越来越多的行业需要在保证隐私和数据安全的前提下进行高效的数据查询。以智慧交通为例，根据国家统计局数据，2022 年中国政府主导的智慧城市 ICT 市场投资规模为 214 亿美元，智慧交通占智慧城市的投资比重约为 14%。2021 年中国智慧交通行业投资规模为 3629 亿元，未来预计还将进一步加大投资力度。TiveQP 提供了一个安全、高效的解决方案，满足了这些市场需求。

TiveQP 不仅适用于医疗、交通、旅游等行业，还可以扩展到金融、地理信息服务等需要高隐私保护的数据查询场景。其应用范围广泛，市场潜力巨大。

第二章 作品设计

本章分为系统设计、实现原理与系统架构三个部分，阐述了 TiveQP 时限系统的设计思路与核心算法，并确定了系统的功能与软件架构。

2.1 系统设计

2.1.1 系统模型

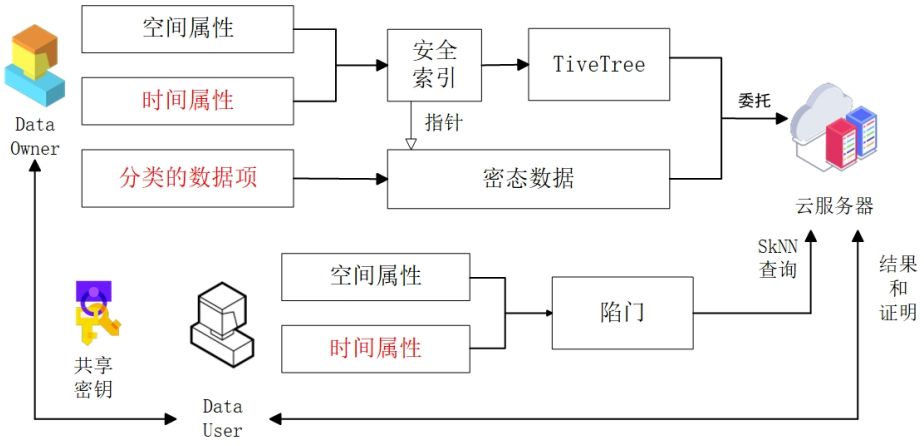


图 2.1　TiveQP 系统模型

数据拥有者 (Data Owner)：数据拥有者拥有一个包含 n 个数据项 $\{L_1, L_2, \ldots, L_n\}$ 的数据集 D。每个数据项 L_i 有两个空间属性（位置类型和坐标）和一个时间属性（允许访问的时间段）。数据拥有者使用 CPA 安全加密算法加密每个数据项，并使用密钥和唯一随机数 r 为每个数据项构建一个安全索引 B。这些索引被分类并组织成一个包含自底向上验证信息的 TiveTreeTR。然后，数据拥有者将 TiveTree 和加密的数据项外包给 CS，并与经过身份验证的数据用户共享密钥。

数据用户 (Combiner)：数据用户是期望获得时限请求的返回结果的用户，通过向云端发送位置和时限等信息进行最邻近（kNN）查询，并根据云端返回结果选择最佳的出行计划。数据用户使用查询位置的空间属性和时间属性构建一个陷门。这个陷门被发送给 CS。数据用户收到 CS 的搜索结果后，使用共享密钥和随机数验证结果的正确性和完整性，然后解密数据项。

云服务器 (Notary)：云服务器存储 TiveTree 和加密的数据项，为数据用户处理最邻近（kNN）查询，并生成结果验证的证明。它将查询结果和证明返回给数据用户。

2.1.2 补充定义

1. **数据模型：**数据集 D 由五个属性的记录组成：id（身份）、typ（位置类型）、lat（纬度）、lon（经度）和 per（允许访问时间）。数据集被安全地索引并存储在云上。
2. **索引树模型：**索引树 TR 是一个包含叶子节点和非叶子节点的二叉树。叶子节点包含安全索引和验证信息，非叶子节点在树中聚合这些信息。该树分两个阶段构建，以确保数据的完整性和安全索引。
3. **查询模型：**查询 Q 指定位置类型、坐标、访问时间和结果数量。系统支持对加密数据的安全查询，使用户能够验证结果的正确性和完整性。
4. **可验证的多维查询：**该方案包括设置、索引、陷门生成、查询和结果验证的算法。它确保数据保密性、查询隐私性和结果可验证性，支持高效和安全的数据检索。其包含五个多项式时间算法。

2.1.3 威胁模型

隐私和安全威胁主要来自于外部和内部的攻击者。在系统外部，攻击者可以对系统内各个实体之间传递的消息进行窃听、重放、篡改等，尝试获取用户的身份隐私和位置隐私，或攻击系统的可用性从而导致系统瘫痪。

威胁模型针对比以前的工作更强大的对手。除了对数据项、索引和查询感到好奇外，对手还可以篡改查询结果或访问数据集的部分内容。如果 CS 被对手或恶意员工攻破，这种攻击可能会发生。为了应对这些威胁，系统采用自适应选择关键词攻击下的不可区分性（IND-CKA）模型。该模型确保系统在对手可以基于先前知识选择关键词的自适应攻击下保持安全。

2.1.4 设计目标

安全性：为了防御半诚实体的攻击，我们利用安全索引和陷门来保护隐私。

- 数据隐私：对手无法从安全索引和加密数据项中学到任何有用的信息。
- 查询隐私：对手无法从陷门中推断出任何有用的位置信息。
- 结果隐私：数据用户无法知道未匹配的数据项的任何信息，除了它们不匹配的事实。

结果可验证性：为了防御恶意云服务器的攻击，我们设计了一种隐私保护且高效的方法，允许数据用户验证查询结果 R。需满足两个要求：

- 正确性: 每个返回的数据项 L 都没有被篡改, 并且是真实存在于原始数据集 $L \in D$ 中的数据项。
- 完整性：所有返回的数据项都是 SkNN 查询的答案，所有其他数据项都不是。

系统允许数据用户验证查询结果的正确性和完整性。正确性确保每个返回的数据项未被篡改，并且确实是原始数据集的一部分。完整性确保所有返回的数据项都是查询的正确答案，并且没有遗漏正确的答案。

2.2 实现原理

在这部分，我们回顾用于构建 TiveQP 的四种基础技术，即空间编码、前缀编码、IBF（不可区分布隆过滤器）和 Merkle 树。

2.2.1 空间编码

空间编码是一种用于处理空间信息的关键技术，在数据管理和检索中发挥着重要作用。它通过一系列函数和操作，将地理位置信息转化为便于查询和处理的编码形式，以提升空间数据的处理效率。下面从函数和定义等方面详细介绍空间编码：

1. 投影函数：投影函数是空间编码的首要步骤，它将地理位置映射到固定网格。设投影函数为 $project(lat, lon)$，其中 lat 代表位置的纬度，lon 代表经度。该函数的作用是根据预设的网格划分规则，把经纬度坐标 (lat, lon) 映射到一个具有唯一网格身份的固定网格 g 上。例如，在一个规则划分的网格系统中，对于某一特定位置 (lat_1, lon_1)，经过 $project(lat_1, lon_1)$ 的运算，能够确定其对应的网格为 g_1 。从数学角度来看，这个映射关系可以简洁地表示为：$g = project(lat, lon)$，这里的 g 就是目标网格，而 (lat, lon) 则是输入的地理位置坐标。这种映射为后续的处理提供了基础的空间定位。
2. 区域扩展函数：区域扩展函数是在投影得到单一网格的基础上，将其扩展为更宽的区域，以获取更多相关信息。定义扩展函数为 $expand(g)$，其中 g 是投影函数得到的输出，即单一网格。该函数的功能是根据特定的扩展规则，以 g 为中心，把它扩展为一组包含多个网格的区域 $G(lat, lon)$。通常情况下，会以 g 为中心，向周围扩展 8 个相邻网格，从而形成一个包含 9 个网格的区域 G 。用数学表达式表示为：$G(lat, lon) = expand(g)$，其中 G 表示扩展后的区域，g 是初始投影得到的网格。这种扩展操作能够涵盖更广泛的空间范围，为后续的查询提供更丰富的数据。
3. 转换为最小前缀集函数 **：最小前缀集函数负责将扩展后的区域转换为一种便于

查询处理的集合形式。定义函数 $toMinPrefixSet(G)$，其中 G 是区域扩展函数的输出，即扩展后的区域 $G(lat, lon)$。该函数会根据特定的编码规则，对 G 中的每个网格标识进行编码处理，生成一组具有特定前缀特征的集合，这个集合就是最小前缀集 MS。从数学角度表达为：$MS = toMinPrefixSet(G)$，这里的 MS 表示最小前缀集，G 表示扩展后的区域。最小前缀集的生成使得数据在后续的查询过程中能够更高效地进行匹配和筛选。

4. 数据用户端的前缀族计算函数：在数据用户端，需要通过特定函数计算与最小前缀集匹配的前缀族。定义函数 $computePrefixFamily(gridId)$，其中 $gridId$ 是数据用户根据自身当前位置提取的覆盖当前位置的网格标识。该函数会根据一定的算法，计算出与最小前缀集 MS 匹配的前缀族。例如，当数据用户确定当前位置对应的网格标识为 $gridId_1$ 时，通过 $computePrefixFamily(gridId_1)$ 函数的运算，就能得到相应的前缀族 PF，用于后续与 MS 进行匹配查询。数学表达式为：$PF = computePrefixFamily(gridId)$，其中 PF 表示前缀族，$gridId$ 表示网格标识。通过计算前缀族，数据用户能够在最小前缀集中快速定位和筛选出符合自身需求的数据，提高查询效率。

2.2.2 前缀编码

在数据处理与信息检索领域，前缀编码是一项关键技术。以下从函数定义、操作步骤等方面详细阐述前缀编码：

1. 前缀族定义函数：给定一个 w 位的数字 x，其以二进制形式表示为 $x_1x_2\cdots x_w$。我们定义前缀族生成函数 $F(x)$，它用于生成数字 x 的前缀族。数学上，$F(x)$ 是一个包含 $w+1$ 个前缀的集合，具体表示为 $F(x) = \{x_1x_2\cdots x_w, x_1x_2\cdots x_{w-1}*, \cdots, x_1*\cdots*, **\cdots*\}$。这里的通配符“$*$”表示该位置可以取任意值。例如，对于一个 4 位二进制数 $x = 1011$，$w = 4$，则 $F(1011) = \{1011, 101*, 10**, 1***, ****\}$。从形式化角度，对于任意 w 位二进制数 x，前缀族 $F(x)$ 可通过如下方式生成：从完整的 x 的二进制表示开始，依次将最右边的一位替换为“$*$”，直到所有位都变为“$*$”为止。
2. 最小前缀集生成函数：对于给定的范围 $[A, B]$（其中 A 和 B 均为二进制表示的数字），我们定义最小前缀集生成函数 $MF([A, B])$。该函数的作用是将范围 $[A, B]$ 转化为最小前缀集，使得这个最小前缀集的所有前缀的并集与 $[A, B]$ 等价。其生成过程较为复杂，通常基于二进制数的位运算和范围划分原理。例如，若 $A = 1000$，$B = 1011$，通过特定算法处理后，可能得到 $MF([1000, 1011]) = \{100*, 101*\}$。数

学上，设 $MF([A,B]) = \{p_1, p_2, \cdots, p_n\}$，那么需要满足 $\bigcup_{i=1}^{n} \text{range}(p_i) = [A,B]$，其中 $\text{range}(p_i)$ 表示前缀 p_i 所涵盖的数字范围。该函数确保生成的最小前缀集既能准确表示给定范围，又具有最小的规模，从而提高后续查询和匹配的效率。

3. 匹配判断函数：为了判断数字 x 是否在范围 $[A,B]$ 内，我们定义匹配判断函数 $match(x,[A,B])$。此函数基于前缀族 $F(x)$ 和最小前缀集 $MF([A,B])$ 的交集进行判断。当且仅当 $F(x) \cap MF([A,B]) \neq \varnothing$ 时，$match(x,[A,B]) = \text{true}$，即 x 在范围 $[A,B]$ 内；反之，若 $F(x) \cap MF([A,B]) = \varnothing$，则 $match(x,[A,B]) = \text{false}$，意味着 x 不在范围 $[A,B]$ 内。例如，若 $x = 1010$，$[A,B] = [1000, 1011]$，由于 $F(1010) = \{1010, 101*, 10**, 1***, 1****, *****\}$，$MF([1000,1011]) = \{100*, 101*\}$，二者交集包含 $101*$，不为空集，所以 $match(1010,[1000,1011]) = \text{true}$。从数学逻辑层面，匹配判断函数可形式化表示为：

$$match(x,[A,B]) = \begin{cases} \text{true}, & \text{if } F(x) \cap MF([A,B]) \neq \varnothing \\ \text{false}, & \text{if } F(x) \cap MF([A,B]) = \varnothing \end{cases}$$

这种基于前缀编码的匹配判断方式，避免了对每个数字进行直接的范围比较，大大提高了范围查询的效率，在数据库索引、数据筛选等众多领域有着广泛的应用。

2.2.3 IBF

双布隆过滤器（IBF）是一种从布隆过滤器扩展而来的数据结构。具体来说，IBF 是一个具有 N 对孪生单元的位数组 B，其中每对孪生单元包含两个存储相反位的单元格。根据这种结构，我们可以同时存储和屏蔽关键字信息。我们将 IBF 总结为以下四种算法：

1. 设置（Setup）：$\text{Setup}(1^n, k) \rightarrow \{H_{key}, H\}$。此算法生成 $k+1$ 个带密钥的哈希函数 $H_{key} = \{h_i : \{0,1\}^* \times \{0,1\}^\eta \rightarrow \{0,1\}^*\}$ 和一个哈希函数 $H : \{0,1\}^* \rightarrow [0,1]$。这里，$H_{key}$ 可以通过使用带密钥的哈希消息认证码 $HMAC$ 生成，即 $h_i(\cdot) = HMAC(K_i, \cdot)$，$K_i$ 是随机生成的长度为 η 的秘密密钥。H 可以使用随机预言机 $\mathbb{H} : \{0,1\}^* \rightarrow \{0,1\}^*$ 生成，即 $H = \mathbb{H}(\cdot)\%2$，用于确定选择哪个单元格。
2. 初始化（Initi）：$\text{Initi}(N, H_{key}, H, \gamma) \rightarrow B$。此算法生成 N 对孪生单元作为双布隆过滤器 B。对于每对孪生单元，选定的单元格和未选定的单元格分别初始化为 0

和 1，即

$$B[\iota][H(h_{k+1}(\iota) \oplus \gamma)] = 0,$$
$$B[\iota][1 - H(h_{k+1}(\iota) \oplus \gamma)] = 1, \iota \in [1, N],$$

其中 γ 是一个随机数，用于消除不同 TBF 之间的相关性。否则，B_i 和 B_j 中第 4 对孪生单元的选定单元格会相同，这会降低随机性并增加数据暴露的风险。

3. ** 插入（Insert）**：Insert$(B, w, H_{key}, H, \gamma) \rightarrow B$。对于集合 w 中的每个元素 w，此算法首先将其哈希到 k 对孪生单元，即 $B[h_1(w)], \cdots, B[h_k(w)]$，然后将这 k 对孪生单元中选定的单元格设置为 1，未选定的单元格设置为 0，即

$$B[h_i(w)][H(h_{k+1}(h_i(w)) \oplus \gamma)] = 1,$$
$$B[h_i(w)][1 - H(h_{k+1}(h_i(w)) \oplus \gamma)] = 0, i \in [1, k].$$

4. 检查（Check）：Check$(B, w, \gamma) \rightarrow 0/1$。为了测试元素 w' 是否存储在 TBF B 中，此算法计算并检查公式（4）。如果对于每个 $i \in [1, k]$ 该式都成立，则 w' 在 TBF B 中。

$$B[h_i(w')][H(h_{k+1}(h_i(w')) \oplus \gamma)] \stackrel{?}{=} 1$$

5. 正确性：如果对于所有安全参数 17、$k \in \mathbb{Z}_p$、$N \in \mathbb{Z}_p$、$\gamma \in \{0,1\}^*$ 以及所有元素 $w \in \{0,1\}^*$，都有 $\{H_{key}, H\} \leftarrow$ Setup$(1^n, k)$，$B \leftarrow$ Iniiti(N, H_{key}, H, γ)，$B \leftarrow$ Insert$(B, W, H_{key}, H, \gamma)$，并且当 $w' \in W$ 时，Check$(B, w, \gamma) = 1$，否则

$$Pr[\text{Check}(B, w', \gamma) = 0] = 1 - negl_1(|W|, N, k),$$

那么 TBF 是正确的。

TBF 的误报概率与传统布隆过滤器（BF）相同，计算为 $negl_1(|W|, N, k) = (1 - (1 - 1/N)^{|W|*k})^k \approx (1 - e^{-|W|*k/N})^k$。如果 $|W| = 56$，$k = 10$，$N = 800$，则该概率为 $(1 - e^{-56*10/800})^{10} \approx 0.1\%$。图 1 比较了 TBF 和 BF 在初始化和插入阶段的情况，其中 TBF 由 10 对孪生单元组成。我们观察到，TBF 可以屏蔽插入位置和插入元素的数量，而 BF 则会完全暴露这些信息。

2.2.4 Merkle 树

Merkle 树是一种在数据验证和完整性保护方面应用广泛的数据结构，常用于区块链、分布式系统等领域。以下从其定义、节点存在性验证以及批量验证等方面进行详细阐述：

1. Merkle 树的定义与结构：Merkle 树是一棵完全二叉树，它依赖于一个哈希函数 H 和一个任意函数 a。对于树中的任意非叶节点 n_p，设其两个子节点分别为 n_l 和 n_r，则 n_p 的函数 a 值由如下公式确定：

$$a(n_p) = H(a(n_l)\|a(n_r))$$

其中，“$\|$”代表连接操作，即把两个子节点的函数值连接起来后，再经过哈希函数 H 计算。例如，若 n_l 的函数值 $a(n_l) = x$，n_r 的函数值 $a(n_r) = y$，那么 n_p 的函数值 $a(n_p) = H(x\|y)$。这种自底向上的计算方式，使得根节点的函数值能够综合反映整棵树的信息，可看作是整棵树数据的“摘要”。

2. 节点存在性验证：在 Merkle 树中，验证某个节点 n 的存在性可通过计算从 n 到根节点路径上的哈希值来实现。假设从节点 n 到根节点路径上的节点依次为 $n_0 = n, n_1, \cdots, n_k$（$n_k$ 为根节点）。对于每个非根节点 n_i（$0 \leq i < k$），根据其在父节点中的位置计算哈希值：- 若 n_i 是其父节点 n_{i+1} 的左子节点，则计算：

$$H(a(n_i)\|a(\text{sibling}(n_i)))$$

其中，sibling(n_i) 表示 n_i 的兄弟节点。例如，若 $a(n_i) = m$，$a(\text{sibling}(n_i)) = n$，则计算结果为 $H(m\|n)$。- 若 n_i 是其父节点 n_{i+1} 的右子节点，则计算：

$$H(a(\text{sibling}(n_i))\|a(n_i))$$

例如，当 $a(\text{sibling}(n_i)) = p$，$a(n_i) = q$ 时，计算结果为 $H(p\|q)$。当完成从节点 n 到根节点路径上所有非根节点的哈希值计算后，若最终得到的根节点哈希值与预先存储的根节点哈希值相等，就表明节点 n 存在于 Merkle 树中。这是因为 Merkle 树的结构特点决定了每个节点的哈希值都基于其下一层子节点的哈希值计算，根节点哈希值是整棵树所有节点信息的综合体现。若计算得到的根节点哈希值与存储的根节点哈希值一致，就意味着从节点 n 到根节点路径上的所有节点信息都与 Merkle 树中的信息匹配，从而证明了节点 n 的存在性。

3. 批量验证：Merkle 树还支持批量验证，这在处理大量数据验证时能显著提高效率。假设有一组节点 $n'_1, n'_2, \cdots, n'_s$ 需要验证，我们可以同时计算这些节点到根节点路径上的哈希值。由于不同节点到根节点的路径可能存在部分重叠，在计算过程中可以共享这些重叠部分的中间计算结果。例如，节点 n'_1 和 n'_2 到根节点的路径在某一中间节点处重合，那么在计算这两个节点到根节点路径上的哈希值时，对于重合部分的中间计算结果无需重复计算，直接使用已有的结果即可。通过这种方

式，能大幅减少计算量，提升验证效率。这种批量验证方式在实际应用中，如区块链中多个交易的验证、分布式系统中多个数据块的完整性验证等场景，发挥着重要作用，有助于提升系统的整体性能。

2.3 系统方案

我们在 2.3.1 节中介绍索引树，在 2.3.2 节展示如何计算陷门，在 2.3.3 节展示如何回答多维位置查询并生成证明，在 2.3.4 节讨论结果验证过程。TiveQP 概述如图 2.2 所示，相关符号如表 2.1。

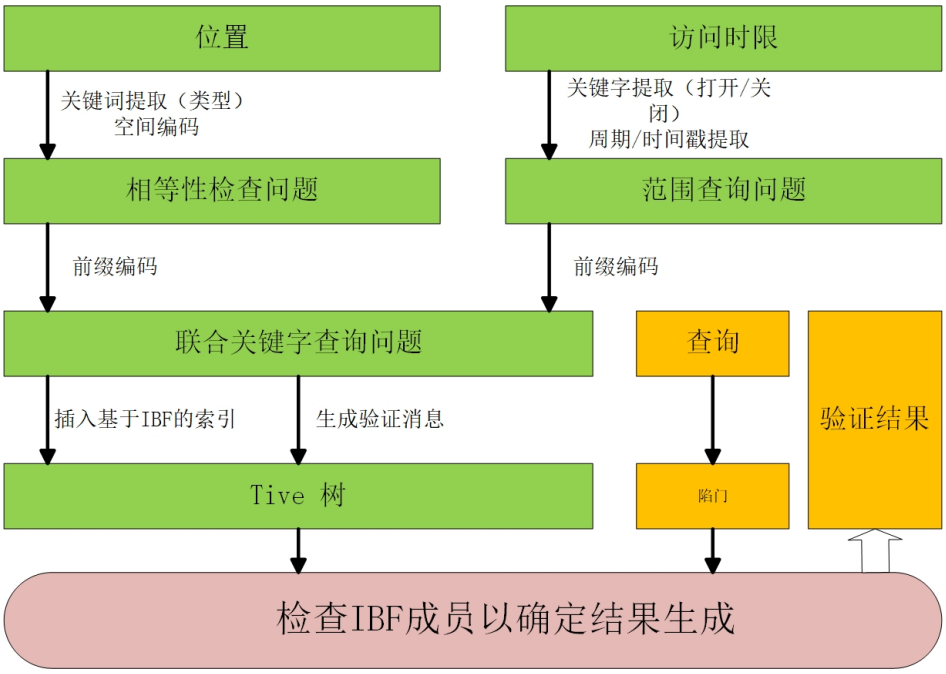


图 2.2　TiveQP 框架概述

2.3.1 构建索引

我们假设服务区域被均匀地划分为一组网格 $G = \{g_1, g_2, \cdots\}$。考虑从“00:00”到“24:00”的时间段，并将每半小时编码为一个单位，从而将时间段变为 $[0, 47]$。

假设数据拥有者和数据用户共享 $m+1$ 个秘密密钥 $SK = \{sk_1, sk_2, \cdots, sk_{m+1}\}$ 和一个随机数 r。构建 m 个伪随机哈希函数 $h_1, h_1, \cdots, h_m$,定义为 $h_i(.) = HMAC(.)\%N$ $(1 \leq i \leq m)$。另一个伪随机哈希函数定义为 $h_{m+1}(.) = HMAC_{m+1}(.)$。哈希函数定义为 $H(.) = SHA256(.)\%2$。检查 B_{kw} 用于检查关键字 kw 是否存在于 IBF B 中，这里关键字是陷门中的一个元素，例如 $T = \{110, 11*, 1**\}$ 有三个关键字。$QueLoc((B, kw))$ 通过查询 IBF B 中的关键字 kw 输出一组位置值。

符号	定义
L, D, E	数据项, 数据集, 加密数据项
B, r	IBF, 随机数
k, TR	查询项个数, TiveTree
di, id, typ	数据项, 身份, 位置类型
lat, lon, per	纬度, 经度, 访问时间段
g, $G(.)$, G	网格, 网格函数, 网格集
MS, $F(.)$, PF	最小前缀集, 前缀函数, 前缀族
w, $\{X_1, X_2, \cdots, X_2\}$, pr	字段长度, 前缀码, 前缀
N, m	位数组长度, hash 函数个数
$\{h_1, h_2, \cdots, h_m\}$	伪随机 hash 函数
SK	密钥
T, T_1	陷门, 类型陷门
T_2, T_3	位置陷门, 时间陷门
v, $root$	节点, 根节点
$result\_num$, R	结果集元素数, 结果集合
ϵ	密态数据项
z	陷门行数
π	证明集合
A	对手

表 2.1　TiveQP 中的关键符号

数据拥有者有一个位置数据集 $D = \{L_1, L_2, \cdots, L_n\}$，每个数据项 L_i 是一个位置，具有身份、位置类型、纬度、经度和开放时间，如图 2.3 所示。例如，“银行”被转换为类型“100”，然后是最小前缀集。类似的操作应用于位置和开放时间。然后数据拥有者计算、存储和上传加密的数据项 E 和 TiveTree TR。首先，数据拥有者将每个 L_i 加密为密文 $Enc(sk, L_i)$。其次，数据拥有者为每个 L_i 计算安全索引如下:

1. 提取 $typi$, lat, lon, $peri$。
2. 给定两个坐标, 定位到网格集 $G(i) = G(lat_i, lon_i)$，例如，一个汉堡店位于网格 g_4，其服务区域覆盖 $G = \{g_0, g_1, \cdots, g_8\}$。
3. 生成随机数 r，通过设置 $B_i[h_o(pr)][H(h_{m+1}(h_o(pr)) \oplus r)] = 1$ 和 $B_i[h_o(pr)][1 -$

$H(h_{m+1}(h_o(pr)) \oplus r)] = 0$，将 $G(i)$ 的最小前缀集中的每个前缀 pr 插入到 $IBF\ B$ 中 $(1 \leq o \leq m)$。

4. 将 $peri$ 编码为最小前缀集 MS_i，并将每个前缀插入 B_i，例如，“08:00 - 12:00”被编码为 $\{010**\}$。

接下来，数据拥有者构建 TiveTree TR 如下：

1. 根据类型和坐标对索引进行分类和排序；对于每种类型，构建一个从底部到顶部的子树，相应的索引作为叶节点，如 2.4 所示。
2. 在每个子树中，对于叶节点 v 上的每个数据项 d_i，计算位置补集（LCS）和时间补集（TCS）。
   - LCS 是覆盖 d_i 位置之外区域的网格标识集。例如，总共有 15 个网格，当前数据项位于网格 8，则其 LCS 为 $[1,7][9,15]$，其最小前缀集为 $\{0001, 001*, 01**, 1001, 101*, 11**\}$。如果查询的位置落在节点的补充区域内，我们可以证明查询不匹配节点的 IBF。
   - TCS 是数据项 d_i 的闭馆时间。例如，开放时间为“08:00 - 12:00”，则其 TCS 为 $[0000, 0800] \cup [1200, 2400]$，转换为 $[0, 15] \cup [24, 47]$，其最小前缀集为 $\{0****, 11*****, 10*1*\}$。

   对于 LCS 中的每个关键字 w_j,计算 $bits_{vj} = QueLoc(B_v, w_j)$ 和 $S_{vj} = HMAC_{sk0}(bits_{vj})$。定义 $bits_v$ 和 S_v 为包含所有计算出的 $\{bits_{vj}\}$ 和 $\{S_{vj}\}$ 的两个集合。对于 TCS，类似地计算 $bits_v$ 和 S_v。$\{bits, S\}$ 是完整性的证明。随后计算哈希值 $HV_v = hash(E_v)$ 作为正确性的证明。
3. 在每个非叶节点 v（除了子树根节点）中，合并两个子节点的补集，构建新的 LCS（TCS）以计算 $bits_v$ 和 S_v；计算 $B_v = B_{left} + B_{right}$ 和 $HV_v = hash(HV_{left} + HV_{right})$。
4. 对于所有子树根节点 v 及其父节点，计算 $B_v = B_{le} + B_{ri}$ 和 $HV_v = hash(HV_{le} + HV_{ri})$，将类型插入 B_i，插入类型补集（YCS）到 B_v 并计算 $\{bits_v, S_v\}$。例如，如果有 63 种位置类型，$typ_v = 20$，YCS 是编码的“$[1-19] \wedge [21-63]$”，即

$$\{000001, 00001*, 0001**, 001***, 0100**, 0101, 01011*, 011***, 1****\}$$

最终，数据拥有者将 (E, TR, r) 外包给云服务器，并与数据用户共享根哈希值 HV_{root}。

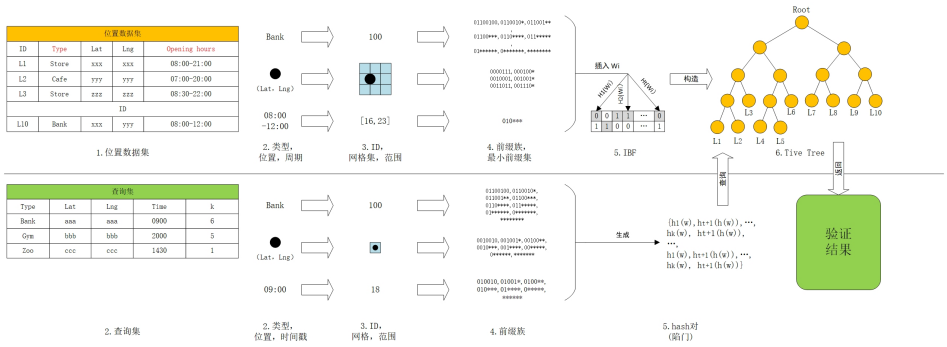


图 2.3　TiveTree 索引和查询处理

2.3.2 陷门计算

一个数据用户有一个位置查询 Q，并计算一个陷门 $T = T_1 \cup T_2 \cup T_3$。这里，这三个集合是针对类型、坐标和时间准备的。

1. 提取 Q 的 $(typ, lat, lon, time)$，计算三个前缀族 PF_1、PF_2 和 PF_3，用于 typ、$g(lat, lon)$ 和时间。例如，“09:00”编码为 $\{010010, 01001, 0100, 010, 01**, 0****, ******, *****\}$。
2. 将 $(h_o(pr), H(h_{m+1}(h_o(pr))))\ (1 \le o \le m, pr \in PF_1)$ 插入 T_1,将 $(h_o(pr), H(h_{m+1}(h_o(pr))))\ (1 \le o \le m, pr \in PF_2)$ 插入 T_2，将 $(h_o(pr), H(h_{m+1}(h_o(pr))))\ (1 \le o \le m, pr \in PF_3)$ 插入 T_3。
3. 将 T 发送给 CS 并等待查询结果和证明。

2.3.3 查询处理

我们通过查找 TiveTree 中节点的 IBF 中的代码来回答查询。对于每个节点 v，我们通过将陷门中的每个元素映射到 v 的 IBF 上来检查查询是否与 v 中的前缀有共同元素。在查询过程中，CS 还搜索三种类型的证据节点的证明。其思想是定位陷门中的一个前缀，该前缀可以查询匹配 $bits_v$ 中的字符串，即前缀与索引的补集 (s) 匹配。在深入研究搜索细节之前，我们介绍三种类型的证据节点。

- 匹配叶节点（MLN）：正确回答位置查询的叶节点。所有潜在的查询结果都存储在这些节点中。
- 不匹配节点（UMN）：不匹配的非叶节点和不满足查询条件的叶节点。搜索在这些节点上停止。
- 不必要节点（UNN）：在我们获得 k 个匹配数据项时不需要搜索的节点。

这三种证据节点被定义为对节点进行分类，并在搜索过程中帮助 CS 生成验证证明。它们的区别在于它们如何匹配搜索条件。CS 从上到下搜索 TR，具体如下：

1. 将当前搜索节点 v 设置为根节点 $TR.root$，并设置 $result\_num = 0$。
2. 在搜索进入子树之前，检查 $Check(B_v, T_1)$。

   (1) 如果 $Check(B_v, T_1) = 1$ 并且 v 是左节点，则将 v 的右子节点临时设置为 UNN，并沿着 v 的左子树和右子树搜索 T；如果 $Check(B_v, T_1) = 1$ 并且 v 是右节点，则移除之前设置的 UNN 标记并搜索 v 的子树。

   (2) 如果 $Check(B_v, T_1) = 0$，即在 $|T_1| = z * m$ 对哈希值 $\{(T_{ij}^1, T_{ij}^2)\}$ 中，对于所有 i 和一个 j，$1 \le i \le z$，$1 \le j \le m$，$B_v[T_{ij}^1][H(T_{ij}^2 \oplus r)] = 0$，则标记 v 为 UMN 并停止搜索。

   对于 UMN，我们生成证明如下：

   (1) 定位满足 $QueLoc(B, T) \in bits$ 的第一个前缀 $w \in T$。通过这种方式，我们获得一个证明，即通过证明 typ 落在类型补集中，证明 Q 不匹配 D。

   (2) 将 $\{w_i, bits_{vi}, S_{vj}, HV_v\}$ 插入 π。
3. 上述搜索递归地应用于子树，直到子根节点，最后一次检查 T_1 以决定是否在子树中搜索 T_1 和 T_2。
4. 在每个子树中，对于节点 v，检查 $Check(B_v, T_2)$ 和 $Check(B_v, T_3)$。有两种情况：

   (1) 如果两个查询都成功：v 不是叶节点，则设置/移除 UNN 并继续类似地搜索子树；v 是叶节点，则将 v 标记为 MLN，将 $\{HV_v\}$ 插入 π，将 $\{E_v\}$ 插入结果集 R 并将 $result\_num$ 加 1。如果找到 k 个数据项，则停止递归并返回将所有 UNN 的 HV 插入到 π 中。

   (2) 如果存在一个不成功的查询，则将 v 标记为 UMN 并停止搜索，在另一个陷阱集中搜索 w_i，并更新 π。
5. 返回 (R, π)。

2.3.4 结果验证

使用返回的证明 π，数据用户使用 π 来验证结果集 R 的正确性和完整性。

- 验证正确性：数据用户需要验证 R 是否正确以及 CS 是否创建了 R 本身。首先，数据用户解密 R 中的 ϵ 并检查她/他的查询是否与明文中的数据项匹配。其次，数据用户根据 Merkle 树从底部向上使用证据节点的哈希值 HV 重新计算根哈希值

$hash(root)$ 的值。如果 $hash(root)$ 等于数据所有者的哈希值 $hash(root)$，则数据用户确信 R 是真实的，并且证据节点是 TiveTree 的真实节点。

- 验证完整性：在查询处理中，陷阱门 T 从根向叶处理。查询处理过程直到陷门与叶节点匹配或者陷门不匹配 TiveTree 索引为止。每个匹配的数据项都有一个搜索路径，我们标记证据节点。因此，使用 UMNs、MLNs 和 UNNs，数据用户可以为每个路径从下到上重现查询过程。

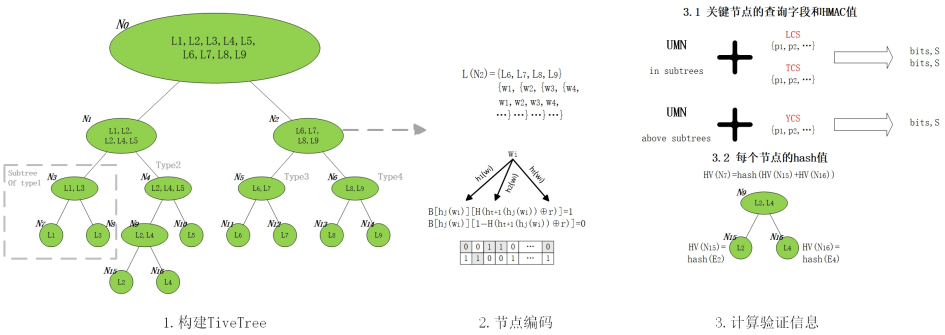


图 2.4　对 UMNs、MLNs 和 UNNs 的验证结果树

如图 2.5 所示，我们使用 Path 1 和 Path 2 作为两个示例。假设具有 $k = 2$ 的查询匹配 L_4 和 L_6。N_1 匹配具有 L_4 的查询，其左子节点标记为 UMN，表示不匹配查询。搜索在 N_1 的右子树中继续，直到节点 N16 匹配查询，该节点标记为 MLN。最后，Path 1 为 $N_0 \rightarrow N_1 \rightarrow N_4 \rightarrow N_9 \rightarrow N_{16}$。在 Path 2 上，我们注意到有两个 UNN，N_{12} 和 N_6。它们被标记是因为当搜索到达 N_{11} 时我们获得 $k = 2$ 个数据项。因此，我们不需要搜索 N_{12} 或 N_6。

对于每个搜索路径，UMNs、MLNs 和 UNNs 的并集必须忠实地重现搜索。如果 CS 故意忽略了一些节点，结果集将不完整。

第三章 安全与隐私性证明

我们证明了该算法在自适应 IND - CKA 模型下是安全的。我们分别使用 HMAC 和 SHA256 来实现哈希函数 $h_1, h_2, \cdots, h_{m+1}$ 和 H。当且仅当函数输出和真正随机函数输出不能被概率多项式时间（Probabilistic Polynomial Time，PPT）对手区分时，一个函数是伪随机函数 [12], [6]。在模拟中选择未来的查询之前，PPT 对手 A 可以查看其过去的查询和相应的陷阱门、结果和证明。

为了证明 TiveQP 在自适应 IND - CKA 模型下是安全的，我们首先构建了一个可以模拟未来查询的 PPT 模拟器 s。接下来，我们证明了与 s 相互作用的 A 面临着以不可忽略的概率区分真实安全指标和 s 中的指标的挑战。从形式上看，如果 PPT 攻击者 A 不能区分伪随机函数生成的真实索引和真正随机函数生成的模拟索引，那么查询处理方案是安全的，并且具有不可忽略的概率：

$$|Pr[Real_{A,C}(1^\lambda) = 1] - Pr[Ideal_{A,s}(1^\lambda) = 1]| \leq negl(\lambda)$$

其中，$negl(\lambda)$ 是一个可忽略的函数。我们定义两个泄漏函数如下：

- $L_1(D) = (n, N, TR, |E|)$：给定位置数据集 D，L_1 输出数据集大小 n、IBF 位长度 N、TiveTree TR 和密文位长度 $|E|$。
- $L_2(D, Q) = (\alpha(Q), \beta(Q), \gamma(Q))$：给定一个位置数据集 D 和一个位置查询 Q，L_2 输出由查询 $\alpha(Q)$、搜索模式 $\beta(Q)$ 和路径模式 $\gamma(Q)$ 返回的数据项 id。

定理 1（安全性）：在随机 oracle 模型中，针对自适应 A，TiveQP 是 $IND - CKA(L_1, L_2)$ - 安全的。

证明：我们首先构造 s 来模拟视图 $V^* = (TR^*, T^*, E^*)$，基于 $L_1(TR, D)$ 和 $L_2(TR, D, Q)$；接下来，我们证明 A 不能区分 V^* 和真正的对手观点 A。

- 为了模拟 TR^*，s 首先构建一个结构相同的 TiveTree。s 从 L_1 获取 N，并为 TR 中的每个节点 v 建立 IBF B_v。在 B_v 的第 i 个单元格中，s 设 $B_v[i][0] = 1$，$B_v[i][1] = 0$，反之亦然。双胞胎是通过抛硬币来决定的。接下来，s 将 B_v 与随机选择的数字 r 关联起来。对于验证信息，s 为每个叶节点随机选择一个网格，计算相应的 LCS 和 TCS，并生成 $bits_v$ 和 S_v。s 继续这个步骤，直到子树节点，并为每个节点计算 YCS，直到根节点。最后，s 将 TR^* 返回给 A。TR^* 中的每个 IBF B_v 与 TR 中的 $IBFB_v$ 大小相同。它们的 $'0's$ 和 $'1's$ 是均匀分布的。因此，A 不能区分模拟的 TR^* 和真实的 TR。
- 为了模拟 T^*，s 知道接收到的 Q 是否已经从 C_2 被处理。如果是，s 将之前的陷阱

门 T 返回给 A。否则，s 生成一个新的陷阱门 T^*，它是 m 对哈希的集合。给定来自 L_2 的访问模式，s 知道哪些数据项与 T 匹配。对于 MLF，s 通过使用 H 选择 e 对哈希来生成输出，同时满足所选哈希对与 MLF 匹配。对于不匹配的叶节点，s 通过使用随机 oracle 将 T 与叶节点不匹配来生成输出。哈希的 e 对是 T^*。因为陷门是由随机哈希函数产生的，所以 A 不能区分 T^* 和真实的陷门。

- 为了模拟 E^*，s 首先从 L_1 获取 n 和 $|E|$。接下来，s 用随机明文模拟密文集，已知的 CPA 安全加密算法 Enc。s 必须确保模拟密文的大小与真实密文的大小相同。

综上所述, 模拟视图和真实视图是 A 无法区分的, 因此, 在随机 oracle 模型中, TiveQP 是自适应的 $IND-CKA(L_1, L_2)$ 安全的，即实现了索引隐私和查询隐私。此外，CS 不像 [18] 那样将叶节点的 IBF 返回给数据用户，而只返回一个位串和一个 HMAC。数据所有者只能验证查询的不匹配，不能在 IBF 上插入其他查询的陷门。因此，实现了结果隐私。

第四章 系统实现

4.1 系统部署

根据实际情况添加相应部署内容。

4.2 后端搭建

可描述后端使用的编程语言、框架，数据库选型及配置等。

4.3 前端搭建

前端搭建涉及多种技术栈，以下是详细介绍：

- **Vue3**：Vue 3 是构建前端界面的核心框架，带来了诸多显著提升。其采用 Proxy 实现响应式系统，相比 Vue 2 的 Object.defineProperty，性能大幅飞跃。Proxy 能劫持对象操作，精准追踪数据变化，减少不必要更新。组合式 API 是一大亮点，它让代码结构更清晰，便于维护与扩展。开发时可将逻辑关注点封装成独立函数，按需引入使用，避免了 Vue 2 选项式 API 代码碎片化问题，尤其在处理复杂逻辑时优势明显。同时，Vue 3 对 TypeScript 支持更佳，结合 `<script setup lang="ts">` 语法，能充分发挥静态类型检查优势，为组件的 props、emits、data 等定义明确类型，在开发阶段就发现潜在类型错误，提高代码可靠性。本作品中使用 Vue 3 搭建网页前端界面。
- **Vite**：Vite 作为项目构建工具，赋予了我们快速的开发体验。它基于原生 ES 模块，开发阶段无需打包，启动速度极快。当修改代码时，能实现即时热重载，几乎瞬间更新页面，极大提高了开发效率。Vite 拥有丰富的插件生态，例如 `@vitejs/plugin-vue` 专门处理 Vue 3 组件编译，可无缝集成 Vue 3 应用到项目中。在生产环境，Vite 使用 Rollup 打包，支持 Tree Shaking 功能，能自动移除未使用代码，减小打包文件体积，提升应用加载速度。本作品中使用 Vite 构建和开发网页前端项目。
- **Element Plus**：Element Plus 是基于 Vue 3 的 UI 组件库，为我们提供了丰富多样的可复用组件。其组件库涵盖按钮、输入框、表格、菜单等常见 UI 组件，具有统一设计风格和良好交互性，直接在项目中使用这些组件，无需从头编写样式和交互逻辑，大大提高了开发效率。而且，Element Plus 提供完善文档和示例，方便我们对组件进行定制和扩展，可通过修改组件属性、插槽等实现个性化需求，如自定义按钮样式、修改表格列配置等。此外，它还支持国际化，引入相应语言包就能

轻松实现多语言切换，满足不同地区用户需求。本作品中部分组件使用了 Element Plus。

- **Vue Router**：Vue Router 是 Vue.js 官方的路由管理器，用于实现单页面应用（SPA）的路由功能。通过配置路由规则，可实现不同页面之间的导航和切换。在项目里，我们定义不同路由路径和对应组件，用户访问不同路径时，Vue Router 自动渲染相应组件。同时，它支持路由守卫，能在路由切换前进行验证和处理，比如检查用户是否登录，未登录则跳转到登录页面，提高应用安全性。另外，为提升应用性能，Vue Router 支持路由懒加载，将路由对应的组件进行懒加载，用户访问该路由时才加载相应组件，减少初始加载文件体积，加快应用启动速度。Vue Router 是本作品中用于实现单页面应用（SPA）路由功能的核心工具，为用户提供了流畅的页面导航体验。
- **TypeScript**：TypeScript 作为 JavaScript 的超集，为项目带来了静态类型检查的优势。在项目中，我们用它编写组件逻辑、接口定义和类型注解。静态类型检查能在开发阶段发现潜在类型错误，避免运行时出现类型相关问题，例如为函数参数和返回值定义明确类型，调用函数时若传入参数类型不符，TypeScript 会立即提示错误。而且，类型注解让代码更清晰易懂，其他开发者阅读代码时能快速了解变量和函数用途及类型，也有助于代码重构和维护，修改代码时能及时发现因类型不匹配导致的问题。此外，Vue 3 对 TypeScript 提供了良好支持，在 `tsconfig.json` 和 `tsconfig.app.json` 中配置编译选项，确保 TypeScript 代码能正确编译和运行。TypeScript 为本作品带来了静态类型检查，提升了代码的可靠性和可维护性。

4.4 开发工具及环境配置

本作品的开发过程使用了多种工具和技术，以确保系统的高效、可靠和可扩展性。这些技术和工具不仅提高了开发效率，也增强了系统的稳定性和安全性。开发过程中，我们使用的软件环境如下（假设此处后续会补充表格，这里仅占位说明）：

开发工具及环境配置涵盖前端和后端两部分技术栈，前端已在前端搭建部分详细介绍，后端技术栈如下：

- **Go** 语言：Go 语言是由 Google 开发的开源静态类型、编译型编程语言，它巧妙融合了传统编译型语言的高性能与脚本语言的高效开发特性。其具有显著优势：一是编译速度极快，运行时配备高效的垃圾回收机制，能有效降低内存开销，大幅提升程序整体性能；二是内置轻量级线程（goroutine）和通道（channel），极大简

化了并发编程复杂度，让开发者可轻松编写高并发程序，充分发挥多核处理器性能；三是语法简洁易懂，减少了冗余代码，提高了代码的可读性与可维护性。在本作品里，Go 语言承担着编写系统核心逻辑的重任，涵盖数据加载、树的构建、陷门生成、查询以及结果验证等功能。

- **Go** 标准库：Go 标准库为 Go 语言开发者提供了涵盖文件操作、网络编程、加密解密、数据结构等多领域的丰富功能与工具。在本作品中，主要运用了多个标准库包，其中 `crypto` 包提供各类加密和哈希算法，像 `crypto/sha256` 可计算 SHA - 256 哈希值、`crypto/hmac` 能计算 HMAC 哈希值，以此确保数据的完整性和安全性；`math/big` 包可处理大整数，在密码学计算和索引计算中，能准确应对超出普通整数范围的数值；`bytes` 包提供字节切片的操作方法，便于对字节数据进行处理和转换；`rand` 包用于生成随机数，在加密和初始化过程中发挥了重要作用。
- 并发编程模型（**goroutine** 和 **channel**）：Go 语言的并发编程模型以 goroutine 和 channel 为核心，为开发者提供了强大的并发编程能力，使得开发者能够轻松编写出高并发的程序。goroutine 是一种轻量级的线程，由 Go 运行时管理，相较于传统的线程，它的创建和销毁开销极小，并且可以在单个操作系统线程上运行多个 goroutine，从而充分利用多核处理器的性能。而 channel 则是一种用于在 goroutine 之间进行通信和同步的机制，它可以确保数据在不同的 goroutine 之间安全地传递，避免了共享内存带来的并发问题。在本作品中，充分利用了 Go 语言的并发编程特性，通过使用 goroutine 并行构建子树，显著提高了树的构建效率。具体来说，在构建树的过程中，将所有的 owner 数据分成多个小块，每个小块包含 1000 个元素。对于每个小块，启动一个独立的 goroutine 来构建子树。这样，多个子树可以同时构建，而不是按顺序依次构建，大大减少了总的构建时间。同时，为了确保并发操作的安全性和稳定性，本作品使用 channel 进行结果的传递和错误处理。定义了两个 channel：一个用于接收构建好的子树根节点，另一个用于接收构建过程中出现的错误。每个 goroutine 在完成子树构建后，将结果发送到对应的 channel 中。主 goroutine 通过监听这些 channel，收集所有子树的构建结果，并处理可能出现的错误。如果在构建过程中出现错误，主 goroutine 会立即停止后续的操作，并返回错误信息。
- 测试框架：使用 Go 的 `testing` 包进行单元测试，编写了多个测试用例来验证各个模块的功能。通过单元测试，可以确保系统的稳定性和正确性，及时发现和修复潜在的问题。在本作品中，`main_test.go` 和 `all_test.go` 文件中包含了对数据加载、树构建、加密解密等功能的测试。

第五章 作品测试与分析

5.1 测试目的

本系统的测试旨在验证各个组成部分的功能完整性和性能表现，确保整个系统的稳定运行和安全性。主要目标包括：

1. 功能测试：
   - 索引构建功能：确保 Index 算法能准确地对数据所有者的位置数据集进行加密处理，构建出安全的 TiveTree 索引。验证每个数据项的空间和时间属性都被正确编码并插入到相应的 IBF 中，同时检查各类互补集（如 LCS、TCS、YCS）的计算和插入是否准确无误。
   - 陷门计算功能：确认 Trapdoor 算法能根据用户的查询条件，正确生成用于查询的陷门。验证陷门中的前缀编码和哈希值计算是否正确，确保其能准确反映用户的查询需求。
   - 查询处理功能：验证 Query 算法能在 TiveTree 索引中高效准确地进行查询。检查查询过程中对各类节点（如 MLN、UMN、UNN）的判断和处理是否正确，确保返回的查询结果集 R 和证明集 π 准确无误，且符合查询条件。
   - 结果验证功能：通过 Verify 算法验证查询结果的正确性和完整性。确保数据用户能够利用证明集 π，准确验证返回的结果集 R 未被篡改，且包含所有符合查询条件的数据项，同时验证查询过程的真实性和完整性。
2. 计算开销测试：
   - 索引构建阶段：评估 Index 算法在构建 TiveTree 索引时的计算资源消耗，包括计算前缀编码、插入 IBF、计算互补集、生成哈希值等操作所占用的 CPU 时间和内存资源。分析不同规模数据集（如不同数量的数据项 n、不同的位置类型 t）和不同网格宽度 gw 对索引构建计算开销的影响。
   - 陷门计算阶段：测量 Trapdoor 算法生成陷门时的计算开销，主要包括前缀编码和哈希值计算所消耗的资源，探究查询参数（如查询类型、位置、时间）对陷门计算开销的影响。
   - 查询处理阶段：测试 Query 算法在查询过程中的计算资源消耗，重点关注遍历 TiveTree 索引、检查陷阱门与 IBF 的匹配、生成证明集等操作的计算开销。分析数据规模（数据项数量 n、位置类型 t）、网格宽度 gw 以及查询参数 k 对查询处理计算开销的影响，验证查询时间复杂度是否符合理论预期 $O(k \log(\frac{n}{tc}))$。

- 结果验证阶段：评估 Verify 算法验证查询结果时的计算开销，包括解密数据项、重新计算根哈希值以及检查证明集等操作所消耗的资源。分析结果集大小、证明集内容以及数据规模对结果验证计算开销的影响。

3. 通信代价测试：

- 数据上传阶段：测量数据所有者将加密数据项和 TiveTree 索引上传至云服务器时的数据传输量和传输时间，分析数据集规模（数据项数量 n、位置类型 t）对上传通信代价的影响。
- 查询请求与响应阶段：评估数据用户向云服务器发送查询请求以及接收查询结果和证明集时的通信开销，包括查询请求消息大小、响应消息（结果集 R 和证明集 π）大小以及传输时间。分析查询参数（如查询类型、位置、时间、查询数量 k）对通信代价的影响，研究不同网络环境下的传输效率。
- 多节点交互阶段：若涉及多个数据用户或云服务器之间的交互（如在分布式场景下），测试节点之间的数据传输量和传输延迟，评估系统在复杂网络环境中的通信性能。

这些测试将帮助我们识别和解决潜在问题，优化系统性能，从而提供可靠、高效且安全的服务。

5.2 测试环境

数据集：我们使用来自美国和加拿大 836 个城市的 Yelp 商家和用户数据集 [**?**, **?**]。我们将所有的源代码、处理后的数据集以及一份说明文件上传到了`https://github.com/UbiPLab/TiveQP`。由于原始数据集过大（4.9GB），我们仅提供一个链接。

参数：我们将 n 的值从 10,000 变化到 100,000，位置类型 t 从 20 变化到 100，位置宽度 gw 从 1 变化到 5 公里，k 从 1 变化到 50。根据 FPR 等式 [**?**, **?**]，一个 IBF 的结果大小范围从 24KB 到 120KB。（假设此处有详细参数设置表，需用 tabular 环境编写，示例如下）默认参数值 n 设置为 20000，我们在回顾相关文献时发现，这是当前最先进的实验设置，且是一个相对较高的值。

基线方法：为了评估安全的多维度查询框架的查询性能，我们将 TiveQP 与六种基线方法进行比较：（1）PBtree[**?**] 支持对单维数据进行范围查询。（2）IBtree[**?**] 通过使用布隆过滤器（IBF）实现合取查询处理。（3）SecEQP[**?**] 基于基于投影的函数和布隆过滤器（IBF）对位置进行编码。（4）ServeDB[**?**] 支持多维度且可验证的范围查询。（5）R*-树 [**?**] 是一种用于对空间信息进行索引的动态树结构。（6）无类型的

参数	最小值	最大值	默认值
n	10000	100000	20000
t	20	100	-
gw	1（公里）	5（公里）	-
k	1	50	-

表 5.1　详细参数设置

TiveQP 与 TiveQP 类似，但它的数据项中没有类型信息。

实验设置：我们在一台个人计算机服务器上进行了实验。该服务器运行的是 Windows Server 2021 R2 数据中心版操作系统，配备主频为 3.7 GHz 的英特尔（R）酷睿（TM）i7 - 8770K 处理器以及 32GB 随机存取存储器（RAM）。我们使用 HMAC - SHA256（哈希消息认证码 - 安全散列算法 256）作为伪随机函数来实现布隆过滤器（IBF）的哈希函数。我们采用高级加密标准（AES）作为对称加密算法。由于在对数据项进行加密时，高级加密标准（AES）可同时应用于 TiveQP 和其他方案，因此在对比中，我们将重点放在索引上，并排除了加密结果。

5.3 功能性测试

5.3.1 索引构建

此处内容待补充（可描述索引构建的测试方法、测试用例、预期结果等，示例：采用 [具体测试工具] 对不同规模的数据集进行索引构建测试，预期在 [时间范围] 内完成构建，且构建的索引能准确反映数据特征等）

5.3.2 检索测试

此处内容待补充（可介绍检索测试的场景、查询条件设置、评估检索准确性和效率的指标等，示例：设置不同的查询条件，如空间范围查询、时间范围查询等，通过对比实际返回结果与预期结果来评估检索准确性，记录查询响应时间来衡量检索效率）

5.3.3 正确性验证

数据用户需要验证 R 是否正确，以及云服务器（CS）是否伪造了 R。首先，数据用户解密 R 中的加密数据项 ε，并检查其查询是否与明文数据项匹配。其次，数据用户根据 Merkle 树 [**?**]，利用证明 π 中证据节点的哈希值 $\{HV\}$，自底向上重新计算

根哈希值 hash′(root)。如果 hash′(root) 与数据所有者的根哈希值 hash(root) 相等，数据用户就可以确信 R 是真实的，并且证据节点是 TiveTree 的真实节点。

5.3.4 完备性验证

在查询处理过程中，陷门 T 从根节点向叶节点进行处理。直到陷门与叶节点匹配，或者陷门与 TiveTree 索引不匹配时，查询过程才会终止。每个匹配的数据项都有一条搜索路径，我们会标记该路径上的证据节点。因此，利用未匹配节点（UMNs）、匹配叶节点（MLNs）和不必要节点（UNNs），数据用户可以自底向上重现每条路径的查询过程。

5.4 计算开销测试

5.4.1 构建耗时

随着 n 从 20 增长到 100K（千）且位置宽度 gw 从 1 千米增加到 5 千米，TiveQP 的构建时间分别从 3.58 分钟增长到 19 分钟，以及从 3.58 分钟减少到 1.54 分钟。在图 5.1(a) 和图 5.1(b) 中：

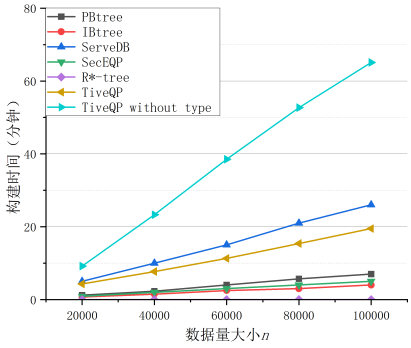
关于不同的 n 和 gw，树的大小分别从 2.89GB 增长到 14.37GB，以及从 2.89GB 减少到 2.34GB。

很明显，构建成本随着 n 的增加呈线性增长。而构建成本会随着位置宽度 gw 的增加而降低，这是因为当 gw 变大时，每个网格覆盖的空间更大，网格的数量会减少，从而减小了前缀最小集合（MS）的规模以及位置互补集合的规模。由于位（bits）和哈希值向量（Hv）的总长度比布隆过滤器（IBF）短得多，所以 gw 对树构建的影响较小。

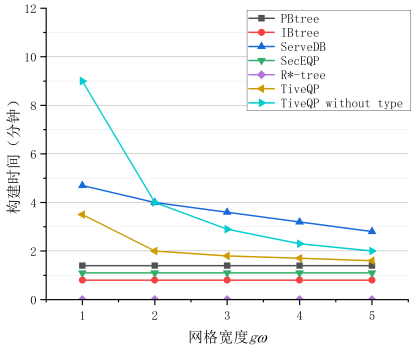
我们注意到，在图 5.1(b) 中最左边的绿色点迅速下降到下一个点。这是因为当 gw 从 1 增加到 2 时，两个相应互补集合的规模差异比之后的互补集合规模差异更大，这就导致了更多的哈希运算。

验证信息的生成在构建时间中占主导地位，因为我们需要计算每个节点的互补集合，然后再计算位（bits）和哈希值向量（Hv）。这些计算比在叶子节点上对布隆过滤器（IBF）进行的哈希运算消耗的时间更多。

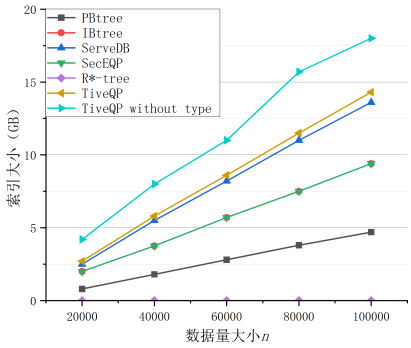
一个布隆过滤器（IBF）的大小对应于插入到该布隆过滤器中的前缀的总数。布隆过滤器（IBF）决定了树的大小。例如，当 $n = 20,000$ 且 $gw = 1$ 千米时，树的大小为 2.893GB，而布隆过滤器（IBF）的大小为 1.863GB。这是因为：



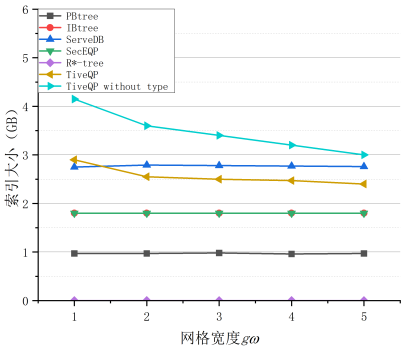
(a) Construction time varying n



(b) Construction time varying gw



(c) Index size varying n



(d) Index size varying gw

图 5.1　相关性能随 n 和 gw 的变化情况

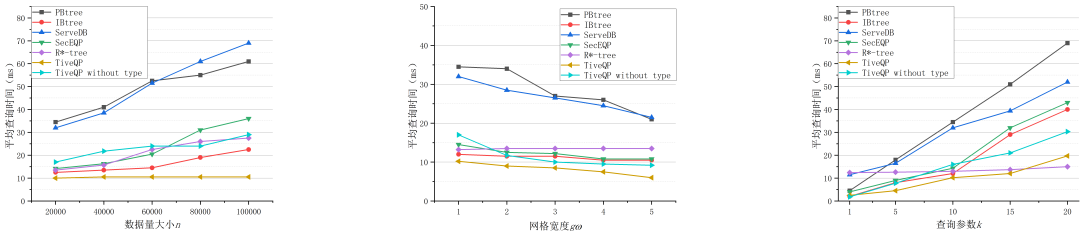
1. 在给定固定的 n 和 m 的情况下，为了维持误报率（FPR），一个布隆过滤器（IBF）的长度会比较长；
2. 对于一个叶子节点，其三个互补集合中的每一个都会生成一个较小的前缀族集合，这使得通信开销较小，并且随着节点向上合并，这三个互补集合会变得更小；
3. 位（bits）中的每个字符串都有 m 个位以及一个标识符，哈希消息认证码集合（Sv）中的元素和哈希值（HV）都是 256 位。

实验结果表明，TiveTree 的构建成本是可以接受的。

5.4.2 检索耗时

查询处理的时间复杂度为 $O(k\log(\frac{n}{k}))$。平均时间：图 5.2(a) 到 5.2(c) 分别展示了关于不同的 n、位置宽度 gw 和 k 的查询延迟（假设插入相关图片，示例同构建耗时部分图片插入方式）。实验结果表明，查询处理时间在毫秒量级。

改变 gw：当 $n = 20,000$ 且 $k = 10$ 时，TiveQP 的平均查询处理时间约为 10 毫秒。



(a) Average query time by varying n (b) Average query time varying gw (c) Average query time varying k

图 5.2 平均查询时间随不同参数的变化情况

它不会随 n 增加，原因有两点：

1. 我们在五个集合中使用所有数据项进行查询，因此它们的总数保持不变；
2. 在按照类型和城市组织树之后，所需的数据项被排列在叶子节点附近，即它们被放置在索引树的一个小的子树中。

当 $n = 20,000$ 且 $k = 10$ 时，随着网格宽度 gw 从 1 千米增加到 5 千米，平均查询处理时间从 10.1 毫秒减少到 6.2 毫秒。出现这种情况是因为随着 gw 增加，位置前缀数量中的陷阱门变得更少，这导致更少的哈希操作，进而减少处理时间。

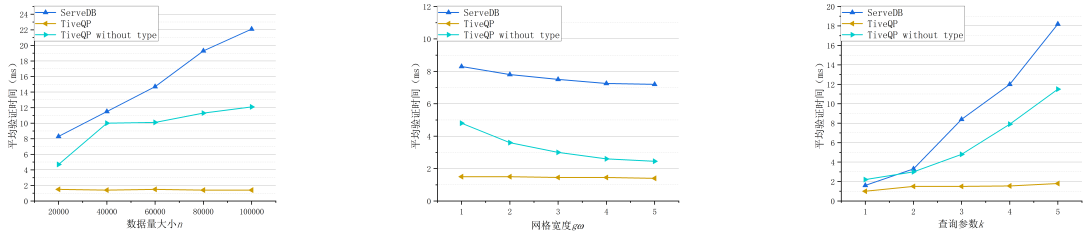
改变 k：当 $n = 20,000$ 且 $gw = 1$ 时，随着 k 从 1 增加到 20，平均查询处理时间从 2.5 毫秒增长到 19.8 毫秒。这是显而易见的，因为更多的查询会导致更多的搜索路径和时间。

5.4.3 验证耗时

数据用户会验证结果的正确性和完整性。我们记录了关于 n、位置宽度 gw 和 k 的平均验证时间。图 5.3(a)、图 5.3(b) 和图 5.3(c) 表明，该时间在 n 和 gw 变化时几乎保持不变（假设插入相关图片）。我们将此优势归因于，在精心进行结构组织后，TiveQP 才构建索引树。随着 k 的增加，时间会略有增长，因为用户必须检查更多搜索路径的更多证明。

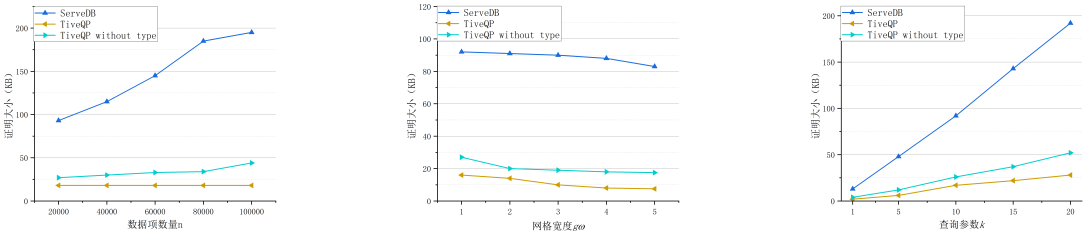
5.5 通信代价测试

图 5.4(a)、5.4(b) 和 5.4(c) 分别展示了关于不同的 n、gw 和 k 的通信开销（假设插入相关图片）。通信开销不会随 n 变化，原因与查询处理中的相同。当 gw 增加时，网格数量减少，网格编码的大小也随之减小，从而降低了通信开销。



(a) Average verification time varying n (b) Average verification time varying gw (c) Average verification time varying k

图 5.3　平均验证时间随不同参数的变化情况



(a) Proof size varying n (b) Proof size varying gw (c) Proof size varying k

图 5.4　证明大小随不同参数的变化情况

第六章 创新性说明

TiveQP 系统是一款极具创新性的限时、可验证且高效的 SkNN 查询处理系统。它深度融合了空间编码技术、前缀编码技术、不可区分的布隆过滤器（IBF）、互补集（CS）概念以及剪枝策略等前沿技术，成功打造出安全、高效且能充分保护隐私的查询处理与结果验证机制。该系统的核心创新优势显著，其独特的编码与查询整合方式极大提升了查询效率，同时，借助隐私保护验证方法和精妙的剪枝策略，TiveQP 系统不仅确保了查询结果的可验证性，更显著增强了查询过程中数据的隐私保护力度。此外，系统通过严格的理论分析和丰富的实验验证，充分展现了在理论层面的严谨性以及实际应用中的高效性，为云计算环境下的加密数据查询处理提供了极具价值的实践方案。

6.1 扩展 SkNN 以支持隐私保护的时间受限访问

TiveQP 系统开创性地对 SkNN 进行扩展，实现了对隐私保护的时间受限访问。系统运用空间编码技术精准处理位置信息，同时利用前缀编码技术巧妙处理位置和时间

信息，并将编码后的结果巧妙插入不可区分的布隆过滤器（IBF）。通过把每个查询精心编码为一个陷门，系统将原本分别针对空间属性和时间属性的处理查询，完美整合为对联合关键字的查询。这种创新方式在有效解决时间限制访问难题的同时，还充分保障了查询效率，使用户能够在特定时间范围内高效获取所需数据，并且全方位保护了数据的隐私性。

6.2 提出保护隐私的结果验证方案

TiveQP 系统创造性地提出了互补集（CS）概念，并设计出一套先进的保护隐私的结果验证方案。传统验证方式常常存在数据隐私泄露风险，而该系统另辟蹊径，不直接证明一个位置不在 A 区域，而是证明其位于 A 的互补区域。具体来说，数据拥有者针对每份数据精准计算位置、类型和打开时间三个互补集，每个集合被转化为前缀的最小集合，由前缀在 IBF 节点中生成一个比特段。系统进一步计算段的键控哈希消息认证码（HMAC）作为签名。当证明不匹配时，系统仅将匹配到的比特段与 HMAC 返回给用户，而非整个 IBF 或比特段，从而最大程度避免了未查询数据隐私的泄露。

6.3 设计剪枝策略提高查询处理和结果验证效率

为大幅提升查询处理和结果验证效率，TiveQP 系统精心设计了一种剪枝策略。系统首先依据数据项的空间属性（如类型和城市）对其进行科学分类，然后自上而下为每种位置类型构建子树。通过这一策略，系统有效消除了不必要的搜索路径，显著缩短了查询处理时间和结果验证时间。查询复杂度成功降低到 $O(k\log(\frac{n}{t*c}))$，其中 t 和 c 分别代表类型和城市的数量。该策略让 TiveQP 系统在面对大规模数据集时，能够迅速响应用户的查询需求，极大提升了系统的整体性能。

6.4 支持动态数据更新与查询优化

TiveQP 系统具备支持动态数据更新的能力，这是其区别于传统查询系统的重要创新点。在实际应用场景中，数据往往是动态变化的，例如实时的位置数据、时间敏感的数据等。TiveQP 系统能够高效地处理这些动态更新，而无需重新构建整个查询结构。系统通过设计一种增量式的编码和更新机制，当有新的数据加入或已有数据发生变化时，只需对相关的编码和布隆过滤器进行局部更新。同时，系统还会根据数据的动态变化，自动调整剪枝策略和查询计划，进一步优化查询性能。这种动态适应性使得 TiveQP 系统在不断变化的数据环境中始终保持高效稳定的查询处理能力。

6.5 实现跨平台兼容性与分布式查询处理

TiveQP 系统实现了出色的跨平台兼容性，能够在多种操作系统和硬件环境下稳定运行。无论是在传统的服务器集群上，还是在新兴的云计算平台、边缘计算设备上，TiveQP 系统都能无缝部署和高效工作。此外，系统还支持分布式查询处理，通过将查询任务合理分配到多个节点上并行执行，充分利用分布式系统的计算资源和存储能力。在分布式环境中，各个节点之间通过高效的通信协议进行数据交互和结果整合，大大提高了查询处理的速度和可扩展性。同时，系统在分布式处理过程中依然能够严格保证数据的隐私性和查询结果的准确性，为大规模、复杂的数据查询场景提供了强大的支持。

第七章 总结

本作品提出了一种限时、可验证、高效的 SkNN 查询处理方案 TiveQP。该方案在设计和实现过程中具备两大重要创新点。其一，通过整合空间属性与时间属性，成功把时间限制这一访问特征引入 SkNN 查询处理，达成了特定时间范围内的数据高效访问。其二，借助互补集中的成员检查，设计出一种独特的隐私保护验证方法。此方法通过验证数据是否处于互补集中，有效避免了未查询数据隐私的泄露，进而增强了系统的安全性与隐私保护能力。

在提升查询效率方面，本作品引入了新颖的空间编码技术和修剪策略。空间编码技术对数据进行精细编码和优化组织，让查询处理更为高效；修剪策略则智能地减少不必要的搜索路径，显著提高了查询速度和结果验证的效率。这些技术创新使 TiveQP 在面对大规模数据集时，能够迅速响应用户的查询需求。

为确保 TiveQP 的安全性，本作品开展了严格的理论分析与证明。通过形式化的安全声明，证实了 TiveQP 能够有效抵御各类潜在的安全威胁，保障用户数据在云环境中的安全性与完整性。实验结果进一步验证了 TiveQP 的性能优势。与传统方法相比，TiveQP 在查询处理速度和结果验证效率上均有显著提升。实验数据表明，在大规模数据集和高并发查询环境下，TiveQP 能够维持稳定的高性能表现。

总体而言，TiveQP 不仅在理论层面具有创新性和前瞻性，在实际应用中也展现出强大的实用性和高效性。它将时间限制、空间编码和隐私验证有机结合，实现了高效、安全的 SkNN 查询处理。这一方案拥有广泛的应用前景，尤其适用于需要高效安全数据访问的场景，例如医疗数据管理、金融数据分析等。未来，TiveQP 有望为更多领域的数据安全与查询效率提供可靠的解决方案，推动云计算环境下的数据处理技术迈向新高度。