QML：一种混合空间索引结构

崔栋，温巧燕，张华，王华伟
（北京邮电大学网络与交换技术国家重点实验室，北京 100876）

摘 要：为了丰富现有学习多维索引的功能并提高索引效率，提出了可以保留数据分布特征的动态数据分段算法 DDSA，并结合四叉树和 *Z* 顺序曲线构建了混合空间索引（QML），在此基础上分别设计范围查询算法和 KNN 查询算法。这种保留数据分布特征的索引可以灵活实现快速查询和更新。实验结果表明，QML 索引在实现丰富功能的前提下优化了检索效率，数据更新的时间复杂度为 *O*(1)。与 R*-tree 相比，QML 索引存储减少约 33%，更新效率提升 40%～80%。查询效率与最优树形索引相近。

关键词：数据库；空间索引；学习索引
中图分类号：TP392
文献标识码：A
DOI: 10.11959/j.issn.1000−436x.2021229

QML: a hybrid spatial index structure

CUI Dong, WEN Qiaoyan, ZHANG Hua, WANG Huawei
The State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract: In order to enrich the functionalities of existing learned multidimensional indexes and improve the efficiency, the dynamic data segmentation algorithm DDSA was proposed, which could preserve the data distribution characteristics. A hybrid spatial index was constructed by combining the QuadTree and Z-order curve (QML). The range query algorithm were designed and KNN query algorithm respectively. The proposed index allowed flexible fast queries and updates with preserving the characteristics of data distribution. Experimental results show that QML optimizes the query efficiency on the premise of achieving rich functionalities, and the time complexity of data update is *O*(1). Compared with R*-tree, the storage consumption of QML is reduced by about 33%, and the update efficiency is improved by 40%~80% . The query efficiency is similar to the optimal tree Index.

Keywords: database, spatial index, learned index

1 引言

物联网设备会生成大量的地理空间数据，为了有效地访问和处理此类数据，数据库管理员通常会采用基于树的索引结构来提高数据分析和事务性工作负载性能[1]。为了提升数据库的读写性能，整个索引通常会保存在内存中。但当数据量达到 PB 级别以上时，树形索引结构会急速变大，进而严重侵占系统资源。研究表明，在商用内存数据库中，索引占用了全部内存空间的 55%[2]。现有的索引结构在空间大数据环境下面临存储空间占用高和查询 I/O 次数多的问题。

收稿日期：2021−08−30；**修回日期**：2021−11−30
通信作者：张华，zhanghua_288@bupt.edu.cn
基金项目：国家自然科学基金资助项目（No.62072051, No.61976024, No.61972048）；中央高校基本科研业务费专项资金资助项目（No.2019XD-A01）；中华人民共和国教育部区块链重点项目计划基金资助项目（No.2020KJ010802）
Foundation Items: The National Natural Science Foundation of China (No.62072051, No.61976024, No.61972048), The Fundamental Research Funds for the Central Universities (No.2019XD-A01), The Key Project Plan of Blockchain in Ministry of Education of the People's Republic of China (No.2020KJ010802)

机器学习算法的引入从根本上改变索引结构。2018 年，Kraska 等[1]在 SIGMOD 会议上首次提出了学习索引的概念。他们将机器学习方法应用于索引结构中，在降低索引空间代价的同时提升了索引查询性能。紧随其后，学者从一维数据结构和多维数据结构中开展了研究工作。其中，一维索引的研究主要有 FITing-Tree[3]、AlEX[4]、PGM-index[5]、XIndex[6]、Dabble[7]等，多维索引的研究主要有 ZM 索引[8]、ML-Index 索引[9]、LISA 索引[10]和 Flood 索引[11]等。空间数据库主要使用的是多维索引。

2019 年，Wang 等[8]在多维空间数据上使用 Z 顺序曲线进行投影降维构建了支持范围查询的 ZM 索引。ZM 索引不支持 K 近邻（KNN, K-nearest neighbor）查询，也不支持插入和删除操作。2020 年，Davitkova 等[9]为了支持 KNN 查询，基于 iDistance 方法构建了 ML-Index 索引，但是没有解决插入和删除的问题。同年，Li 等[10]和 Nathan 等[11]在 SIGMOD 会议上分别提出了 LISA 索引和 Flood 索引。LISA 索引支持了索引的插入和删除操作，在一定程度上优化了效率，但是 LISA 索引并不适用于频繁大规模更新的空间数据集。Flood 索引相比于之前的索引大幅度优化了存储，提高了查询性能，但是只支持只读工作负载，不支持插入等操作。

本文使用四叉树索引结构对空间数据进行均匀快速的划分，使用 Z 顺序曲线进行数据降维。为了使生成的 QML 索引更符合数据的分布特征以支持快速的更新和查询操作，本文提出了动态数据分段算法 DDSA。该算法生成的数据段符合线性和二次曲线分布，这 2 种分布更接近现实数据情况。本文在此基础上使用分段线性函数和二阶多项式函数拟合多维空间数据构建 QML 索引，并实现了范围查询和 KNN 查询。实验结果表明 QML 索引存储比 R*-tree[12]等减少 33%，更新效率提升 40%～80%，查询效率与最优树形索引相近。QML 索引数据更新的时间复杂度仅为 $O(1)$，优于 LISA 索引。本文的主要贡献如下。

1) 提出了 QML，这是一种新的混合空间索引结构，在保证索引查询和更新性能的同时，降低存储空间，是利用数据分布规律来构建空间索引的一种新的尝试。

2) 提出了适用于学习索引的动态数据分段算法 DDSA，该算法能够根据局部数据分布特征选取不同的多项式函数进行拟合。

3) 设计并实现了基于 QML 的范围查询算法和 KNN 查询算法，并通过真实数据集的测试证明 QML 提供了与最优树形索引结构相近的（点查询性能更好）查询性能和更新性能，大幅降低了存储空间。

2 相关工作

空间索引是为多维数据访问设计的数据结构。大多数空间索引是空间驱动或数据驱动的结构[13]。空间驱动的结构（如固定网格索引[14]）将空间分解为单元格，并根据几何准则映射数据对象。数据驱动的结构（如 R-tree[15]）将数据对象划分为簇，并通过其最小外接矩形（MBR, minimum bounding rectangle）划分空间。此外，UB-tree[16]是 Z 顺序曲线[17]和 B^+tree 的组合，是一个平衡树，数据对象按 Z 阶存储。实际数据具有一定的分布特点，但是传统的数据索引结构没有利用这一特点，因此不能发挥数据最大的作用。

由文献[18-22]可知，学习索引可以充分利用数据的分布规律，在提升查询性能的同时降低空间消耗。ZM 索引[8]使用 Z 顺序曲线进行降维；ML-Index 索引[9]采用基于 iDistance[23]方法的改进策略对数据进行投影；LISA 索引[10]使用网格划分和基于勒贝格测度的投影进行数据有序化。Flood 索引[11]采用基于成本模型的网格划分策略；在模型选择上，ZM 索引和 ML-Index 索引都选用了 RMI 模型；为了使索引拥有更低的空间成本和执行成本，LISA 索引和 Flood 索引选择使用分段线性回归模型。在空间查询应用中，已有的多维索引都实现了对空间范围查询的支持，ML-Index、LISA 索引和 Flood 索引还可以支持 KNN 查询。Nathan 等[11]没有给出利用 Flood 索引进行 KNN 查询的具体实现。LISA 索引采用格子回归模型[24]，并结合范围查询实现了 KNN 查询。

支持频繁的数据更新是上述多维学习索引面临的一个难题。ZM 索引、ML-Index 索引和 Flood 索引目前仅支持只读的静态负载；LISA 索引虽然通过采用异地插入的方式支持了索引插入和删除，但是索引的更新效率随着数据更新规模的增加而下降，因此不适用于频繁更新的空间数据。为了应对可能频繁更新的空间数据，本文提出了 QML 混合空间索引结构。QML 索引在支持范围查询和

KNN 查询的基础上，通过动态数据分段算法 DDSA 快速构建本地模型，在大幅降低存储空间和提高动态更新效率的同时，获得近似最优树形空间索引的查询性能。

多维空间数据在降维和有序化后可以近似看作具有时序特征的数据。时序数据的分段模式表示是一种对时序数据进行抽象和概况的表示方法[25]。分段线性近似（PLA, piecewise linear approximation）算法将数据分割成若干段后使用线性函数来逼近每个段。Liu 等[26]提出了一种利用 PLA 实现紧凑分段的方法，称为可行空间窗（FSW, feasible space window）。FSW 算法通过可行空间的概念，在保证每个数据点有误差界的情况下，可以找到每个数据段的最远分割点，在学习索引模型 FITing-Tree[3]中应用的就是该方法。但 FSW 算法无法直接用于非线性函数。Xu 等[27]在此基础上提出一种可行系数空间（FCS, feasible coefficient space）算法用于解决该问题。该算法通过寻找特定函数系数边界确定一个可行系数空间，通过不断缩小该空间的范围来进行数据分段。FCS 算法适用于各种复杂多项式函数的拟合，但空间数据一般呈二次曲线分布，FCS 算法对于索引构建过于复杂。

本文针对上述的问题提出了适用于学习索引的动态数据分段算法 DDSA，该算法使用分段线性函数和二阶多项式函数进行分段，在保证分段效率的同时尽可能将相似数据划分到一起。

3　QML 索引

3.1　QML 核心思想

本文选取了公开数据集 OpenStreetMap[28]上的美国本土各类建筑、商店等标签的地理信息，通过 Z 顺序曲线变换后数据的分布规律如图 1 所示。从图 1 中可以看出，空间数据在经过降维和序列化后呈单调递增的分布规律。

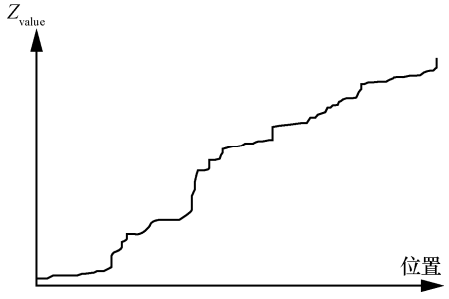


图 1　OSM 美国本土地理信息数据分布规律

为了更好地拟合数据分布并支持索引的动态更新，QML 索引采用四叉树进行数据划分，Z 顺序曲线进行数据降维和有序化，最后使用动态数据分段算法 DDSA 进行数据分段和模型构建。

四叉树的生成和维护比较简单，当数据分布比较均匀时，有较高的数据插入和查询速率。当数据分布不均且数据量较大时，为了缓解数据偏移带来的索引树不平衡的压力，QML 索引使用存储最大数据量阈值的单元格 cell 作为四叉树的叶节点，减少中间节点的数量和树深度。LISA 索引采用的网格划分策略因为在数据变动较大时需要重新排序并划分，所以并不适用于频繁变更的数据集，相比之下，QML 的四叉树结构则可以更灵活快速地进行索引插入和删除操作。

在存储优化方面，QML 索引的节点包括中间节点和叶节点，中间节点不存储数据，叶节点存储最大数据量阈值的数据和本地模型集合。因此 QML 可以显著降低中间节点的数量，减少存储消耗。

在索引动态更新方面，四叉树的数据分割策略将数据空间 V 按数据量划分成最小单元格 cell，保证在数据更新时只需对单独 cell 上的模型进行重构，避免相互间的影响。动态数据分段算法 DDSA 可以通过一次性的数据遍历构建索引模型。实验表明 QML 索引拥有比传统索引更好的更新性能。

在查询性能优化方面，QML 的混合索引结构通过减少中间节点减少了不必要的间接查询，其模型可以根据数据分布特征选取不同的多项式函数进行拟合，减少不必要的数据分段。这些结构设计可以优化 QML 索引查询算法，提高检索速度，同时也很好地支持了范围查询和 KNN 查询。

3.2　QML 构建

如图 2 所示，QML 索引的构建过程可以分为 3 个阶段：数据划分、数据降维和有序化、数据分段和模型生成。相关参数定义如表 1 所示。QML 索引的输入为原始数据集 Data: Data=$\{k_i \mid i=0,1,\cdots,t\}$。

阶段 1　数据划分

步骤 1　将原始数据集 Data 的数据点 k_i 映射到二维数据空间 $V=[0,X)[0,Y)\in \mathbf{R}^2$，计算数据空间 V 的数据量 V_{count}；

步骤 2　如果 $V_{\text{count}}>$ Cell_max_count，执行步骤 3；如果 $V_{\text{count}} \leqslant$ Cell_max_count 则构建单元格 cell，将 cell 加入根节点并执行阶段 2 的步骤 4；

表 1 **参数定义**

参数	含义
k_i	原始数据中的点 $k_i=(x_i,y_i)$
z_{value_i}	k_i 按照 Z 顺序曲线映射函数 M 计算得到，$z_{\text{value}_i}=M(x_i,y_i)$
k_{z_i}	k_i 增加 Z 值后的三元组 $k_{z_i}=(x_i,y_i,z_{\text{value}_i})$
cell	最小空间单位单元格 $\text{cell}=[l_x,u_x)[l_y,u_y)$
Cell_max_count	cell 最大数据量阈值
d	有序数据集: $d=\{k_{z_j}\mid j=0,1,\cdots,n\}$, $(z_{\text{value}_j}\leqslant z_{\text{value}_{j+1}}\leqslant z_{\text{value}_{j+2}})$
D	cell 的数据集合 $D=\{d_i\mid i=0,1,\cdots,m\}$
l_i	数据集合 D 的分段数据集 d_i 的本地模型
L	cell 的分段模型集合 $L=\{l_i\mid i=0,1,\cdots,m\}$
ε	模型集合 L 的置信区间，动态数据分段算法 DDSA 的最大允许误差，$\varepsilon\in N^+$

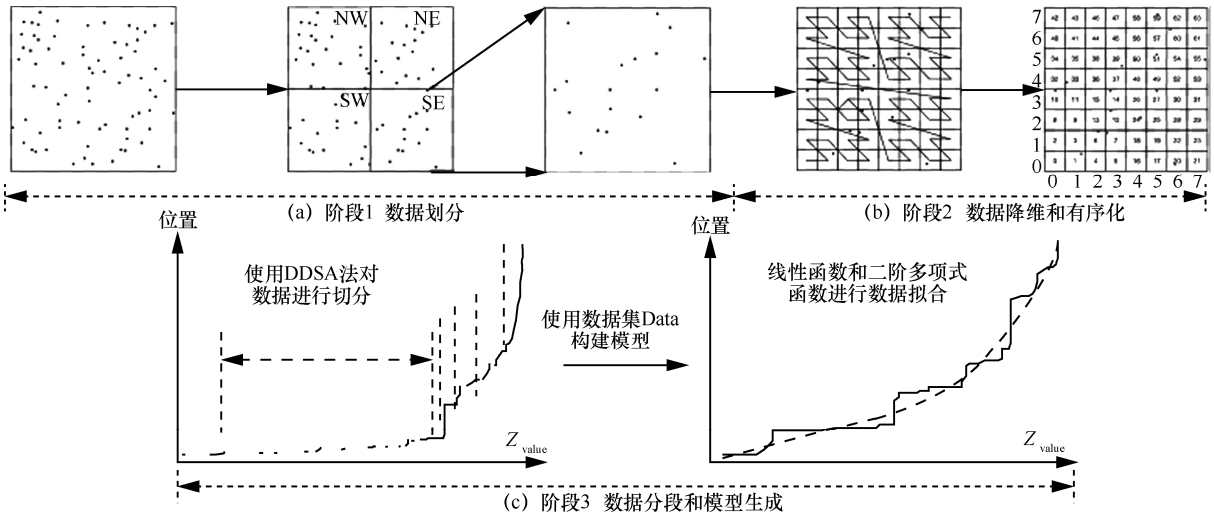


图 2 QML 构建示意

步骤 3 对数据空间 V 按照四叉树进行等距划分，生成 4 个子数据空间{NW, NE, SW, SE}并加入根节点中，对新生成的空间区域 $V_i\in$ {NW, NE, SW, SE}利用步骤 1 再次进行划分；

（见算法 1 的第 1)行～第 11)行）

阶段 2 数据降维和有序化

步骤 4 对于单元格 cell 内任意的 $k_i=(x_i,y_i)$ 按照 Z 顺序曲线映射函数 M 计算 $z_{\text{value}_i}=M(x_i,y_i)$，构建三元组 $k_{z_i}=(x_i,y_i,z_{\text{value}_i})$；

步骤 5 对单元格 cell 所有的 k_{z_i} 按照 z_{value_i} 由小到大排序，形成有序数据集 $d=\{k_{z_j}\mid j=0,1,\cdots,n\}$，$(z_{\text{value}_j}\leqslant z_{\text{value}_{j+1}}\leqslant z_{\text{value}_{j+2}})$；

（见算法 1 的第 12)行～第 21)行）

阶段 3 数据分段和模型生成

步骤 6 将有序数据集 d 中每个 k_{z_j} 的 z_{value_j} 和 j 映射到二维空间，然后使用 DDSA 进行数据分段，得到数据集合 $D=\{d_i\mid i=0,1,\cdots,m\}$；

步骤 7 根据每个子数据集 d_i 的数据分布特征分别使用线性函数和二阶多项式函数进行拟合，生成本地模型集合 $L=\{l_i\mid i=0,1,\cdots,m\}$；

（见算法 1 的第 22)行～第 24)行）

算法 1 QML 索引构建

输入 原始数据集 Data=$\{k_i\mid i=0,1,\cdots,t\}$，cell 最大数据量阈值 Cell_max_count

输出 QML 索引 root

1) function Partition(V, root)

2) 计算数据空间 V 的数据量 v_count

3) if v_count > Cell_max_count then

4) 根据四叉树分割策略构建 V 的子数据空间{nw,ne,sw,se}，将 V(nw,ne,sw,se) 加入 root

5) for v_i in {nw,ne,sw,se} do

6) Partition(v_i, V)

7) end for

8) else

9) DDimensionAndOrder(V, root)

10) end if

11) end function

12) function DDimensionAndOrder(V, root)

13) 初始化数据集 d

14) for k_i in V do

15) $z_{\text{value}_i} = M(x_i, y_i), k_{z_i} = (x_i, y_i, z_{\text{value}_i})$

16) 将 k_{z_i} 加入数据集 d

17) end for

18) 将数据集 d 按照 k_{z_i} 的 z_{value_i} 进行排序

19) $L \leftarrow$ SegmentationAndModel(d)

20) 将 V(L)加入 root

21) end function

22) function SegmentationAndModel (d)

23) 使用算法 2 进行分段和模型生成，得到 d 的本地模型集合 L

24) end function

25) 初始化 QML 索引 root

26) 将 Data 映射到空间 V: $V = [0, X)[0, Y) \in \boldsymbol{R}^2$

27) Partition(V, root)

28) return root

4 动态数据分段算法 DDSA

为了使索引的模型更加符合数据分布特征并提高构建效率，本文设计的动态数据分段算法 DDSA 采用 FSW 算法[27]和 SFCS 算法交叉验证的方式，分别使用线性函数和二阶多项式函数进行拟合。其中 SFCS 算法是本文针对数据的非线性特征专门设计的一个与之适配的简化可行系数空间算法。

本文 DDSA 的工作原理是将待分割有序数据集的第一个数据点初始化为分段的左端点，然后在每一步中尝试将下一个点也放入分段。新加入的数据点需要在当前分段的最大误差允许范围内，如果超过最大误差则开启新的分段。其关键是使当前分段在给定的最大误差下尽可能地延长。

对于每一个分段，DDSA 都会根据最大误差阈值生成一个可行区间 S。当有新的数据点加入时，算法需要结合初始点和最大误差阈值计算新数据点的可行区间 S'，当 2 个区间 S 和 S' 没有共同区域时则表示当前分段结束。

DDSA 先使用 FSW 算法[27]判断新加入的数据点是否符合线性分布，如果不符合再使用本文设计的 SFCS 算法判断是否为二次曲线分布。

DDSA 的详情如算法 2 所示。DDSA 输入为有序数据集 data、最大误差阈值 ε 和算法类型 type。算法类型 type 分别用 0 和 1 表示 FSW 算法和 SFCS 算法。输出结果为数据集 Data 的本地模型集合 L。

算法 2 DDSA 算法

输入 有序数据集 Data=$\{k_{z_i} \mid i = 0,1,\cdots,n\}$，最大误差阈值 ε，type={0,1 | 0: FSW, 1: SFCS }

输出 本地模型集合 L=$\{ l_i \mid i = 0,1,\cdots,m \}$

1) 初始化 L={}

2) 初始化 type=0, p_0=k_{z_0}, d=$\{k_{z_0}, k_{z_1}\}$, d.type=0

3) 初始化可行区间 S_{fsw} = FSW(k_{z_0}, k_{z_1}, ε)

4) for i=2 to n do

5) if type == 0 then

6) $S_{\text{fsw}} = S_{\text{fsw}} \cap$ FSW(p_0, k_{z_i}, ε)

7) if S_{fsw} == null then

8) type=1， $p_0 \leftarrow d$ 起始数据点， $p_m \leftarrow d$ 中间数据点， $p_n \leftarrow d$ 最后数据点

9) 初始化 S_{sfcs} = SFCS (p_0, p_m, p_n, ε)（算法 3）

10) $S_{\text{sfcs}} = S_{\text{sfcs}} \cap$ SFCS (p_0, p_m, k_{z_i}, ε)

11) end if

12) else

13) $S_{\text{sfcs}} = S_{\text{sfcs}} \cap$ SFCS (p_0, p_m, k_{z_i}, ε)

14) end if

15) if S_{fsw} == null and S_{sfcs} == null then

16) 根据 d.type 拟合数据集 d，生成模型 l

17) 将模型 l 加入模型集合 L 中

18) type = 0，$p_0 = k_{z_i}$，d = $\{k_{z_i}, k_{z_{i+1}}\}$，$i = i + 1$，

19) S_{fsw} = FSW(k_{z_i}, $k_{z_{i+1}}$, ε)，d.type = 0

20) else

21) 将 k_{z_i} 加入数据集 d 中，d.type = type

22) end if

23) end for

24) 根据 d.type 类型拟合数据集 d，生成模型 l

25) 将模型 l 加入模型集合 L 中

26) return L

首先进行初始化操作。设置 type=0，构建 FSW 算法可行区间 S_{fsw}，初始数据段 d 包含第一个和第二个数据点 $d=\{k_{z_0},k_{z_1}\}$，初始数据点 $p_0=k_{z_0}$，设置初始数据段 d 的算法类型 d.type=0。

算法从数据点 k_{z_2} 开始遍历。用 FSW 算法构建新的可行区间 S_{fsw}，当该可行区间为空时改用 SFCS 算法进行判断。每一次从 FSW 算法切换到 SFCS 算法时都需要进行初始化操作，选取当前数据段的起始点、中间点和最后数据点构建 SFCS 算法的可行区间 S_{sfcs}。然后根据新生成的可行区间 S_{sfcs} 来判断待检测点是否满足二次曲线分布。

如果在 k_{z_i} 处 S_{fsw} 和 S_{sfcs} 都为空，则在此处进行切分，k_{z_i} 作为下一个数据段的起始点。d 形成新的数据段。最后根据 d 的算法类型 d.type 确定使用线性函数或者二次多项式函数进行拟合生成本地模型 l，加入模型集合 L 中。

上述过程中，SFCS 算法通过计算二次多项式函数参数的可行区间进行判断。详情如下。

假设 XY 二维坐标系中有数据点 $p_0, p_1, \cdots, p_n$ 近似二次多项式 $y=ax^2+bx+c$ 分布，对于每个数据点 $p_i=(x_i, y_i)$ 如果满足 $x_i<x_{i+1}$，则 $y_i \leqslant y_{i+1}$。SFCS 算法的思想就是通过计算二次多项式函数参数 a、b 的可行区间逐个验证每个数据点是否满足最大误差范围的二次多项式函数分布。

坐标系中二次函数采用式(1)的形式，其中 a、b、c 是该函数的系数

$$y=ax^2+bx+c \tag{1}$$

为了定位该曲线，将坐标系的第一个数据点 $p_0(x_0, y_0)$ 和第二个数据点 $p_1(x_1, y_1)$放置在近似曲线上，可以得到

$$y_0=ax_0^2+bx_0+c \tag{2}$$

$$y_1=ax_1^2+bx_1+c \tag{3}$$

根据式(2)和式(3)可以得到参数 a、b 的映射关系

$$b=-(x_1+x_0)a+\frac{y_1-y_0}{x_1-x_0} \tag{4}$$

设置允许的误差范围为 $\varepsilon \in \mathbf{N}^+$，即拟合曲线与数据点的距离在 ε 内。为了简化算法，本文将该距离近似为两者 Y 坐标轴的差值，假如第三个数据点 $p_2(x_2, y_2)$ 满足最大误差的二次曲线分布，则有

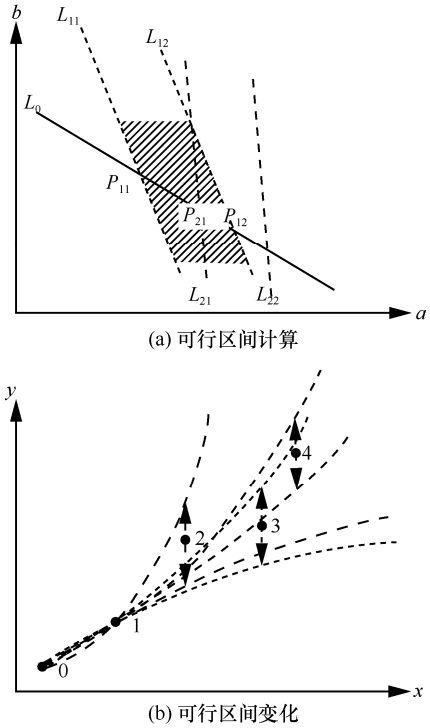
$$y_2-\varepsilon \leqslant ax_2^2+bx_2+c \leqslant y_2+\varepsilon \tag{5}$$

利用式(5)和式(2)，可以进一步得到

$$b \geqslant -(x_2+x_0)a+\frac{y_2-y_0-\varepsilon}{x_2-x_0} \tag{6}$$

$$b \leqslant -(x_2+x_0)a+\frac{y_2-y_0+\varepsilon}{x_2-x_0} \tag{7}$$

如图 3(a)所示，式(4)描述为直线 L_0，式(6)和式(7)描述为平行线 L_{11} 和 L_{12} 之间的区域，因此可以得到参数 a、b 的可行区间$[P_{11}, P_{12}]$。当验证第四个数据点时，可以得到类似式(6)和式(7)的关系（图 3(a)中平行线 L_{21} 和 L_{22} 之间的区域），该区间与直线 L_0 相交于点 P_{21} 和 P_{22}。因此区间$[P_{11}, P_{12}]$和$[P_{21}, P_{22}]$ 重叠区域$[P_{21}, P_{12}]$即新的可行区间。重复该过程，直到可行区间为空。该算法通过不断缩小范围来得到分布相似的数据点并进行划分，其过程如图 3(b)所示。



(a) 可行区间计算

(b) 可行区间变化

图 3 SFCS 算法示意

SFCS 算法如算法 3 所示。相比于算法复杂度为 $O(n)$ 的 FCS 算法，SFCS 算法将多边形的可行空间计算简化为线性的可行区间计算，算法复杂度为 $O(1)$，虽然牺牲了一定的精确度，但是降低了算法的复杂度，提升了算法的运行速度。

算法 3　SFCS 算法

输入　当前可行区间 $S = [P_l, P_u]$，起始数据点 $k_{z_s} = (x_s, y_s, z_{\text{value}_s})$，中间数据点 $k_{z_m} = (x_m, y_m, z_{\text{value}_m})$，待检测数据点 $k_{z_i} = (x_i, y_i, z_{\text{value}_i})$，最大误差阈值 ε

输出　新的可行区间 S'

1) 根据 $k_{z_s}, k_{z_m}, k_{z_i}$ 构建 $p_t=(z_{\text{value}_t}, t)$, $t = s, m, i$

2) 根据式(4)使用 p_s, p_m 构建直线 L_0：

$$L_0 : b = -(z_{\text{value}_m} + z_{\text{value}_s})a + \frac{m-s}{z_{\text{value}_m} - z_{\text{value}_s}}$$

3) 根据式(6)和式(7)使用 p_s, p_i 构建可行区域，上下边界为 L_{n_1} 和 L_{n_2}：

$$L_{n_1} : b \geqslant -(z_{\text{value}_i} + z_{\text{value}_s})a + \frac{i-s-\varepsilon}{z_{\text{value}_i} - z_{\text{value}_s}}$$

$$L_{n_2} : b \leqslant -(z_{\text{value}_i} + z_{\text{value}_s})a + \frac{i-s+\varepsilon}{z_{\text{value}_i} - z_{\text{value}_s}}$$

4) 计算 L_{n_1}，L_{n_2} 和 L_0 的相交点 P_{n_1} 和 P_{n_2}

5) if　S 为空　then

6)　　$S' \leftarrow [P_{n_1}, P_{n_2}]$

7) else

8)　　$S' \leftarrow S \cap [P_{n_1}, P_{n_2}]$

9) end if

10) return　S'

5　基于 QML 的查询处理

5.1　范围查询

对于 QML 索引的范围查询可以视为对矩形区域的查询。给定查询矩形 $\text{qr} = [x_l, x_u)[y_l, y_u)$，其中 x_l 和 y_l 分别表示 x 轴和 y 轴方向最小边界值，x_u 和 y_u 分别表示 x 轴和 y 轴方向最大边界值。范围查询示意如图 4 所示，先通过四叉树快速查找到与 qr 有交集的单元格 cell，然后将 qr 查询分解为多个小矩形的并集

$$\text{qr} = \bigcup_{i=0}^{Q'} \text{qr}_i', \text{qr}_i' = [x_{li}', x_{ui}')[y_{li}', y_{ui}') \in R^2 \tag{8}$$

qr_i' 可以分为两类，一类是 qr_i' 和单元格 cell 完全重合，定义为 cr_i；另一类是 qr_i' 和单元格 cell 不完全重合，定义为 lr_i。对 cr_i 可以将其下面所有的数据直接加入查询结果中；对 lr_i 需要根据本地模型进行转换查询，经过数据校验合格后再加入结果。查询结果可以表示为

$$R = \bigcup_{i=0}^{Q_c} \text{cr}_i + \bigcup_{j=0}^{Q_l} \text{lr}_j \tag{9}$$

当查询的类型为 lr_i（图 4(b)）时，索引会先计算其矩形区域的最小 Z 值和最大 Z 值。然后将 Z 值映射到模型曲线上（图 4(c)）获取可能的数据地址，对应的数据地址需要根据最大误差范围进行修正以保证覆盖率。因为模型曲线映射的数据范围大于查询的 lr_i 实际范围（图 4(b)深色区域），因此需要对查询的数据进行校验再放入查询结果中。

5.2　KNN 查询

给定一个数据点 $q_{\text{knn}} = (x, y)$ 和一个距离值 δ，以点 q_{knn} 为圆心划定半径为 δ 的圆，将该圆内部区域定义为 $B(q_{\text{knn}}, \delta)$。将 $B(q_{\text{knn}}, \delta)$ 包含的点按照与点 q_{knn} 的距离由小到大排列，最终选取的 top K 个点即 KNN 查询的结果。

QML 索引采用空间密度估算 KNN 查询距离 δ，将 KNN 查询转化为递归和范围查询。QML 的 KNN 查询过程如图 5 所示，通过构建矩形区域 $Q(q_{\text{knn}}, \delta)$，将目标区域 $B(q_{\text{knn}}, \delta)$变为 $Q(q_{\text{knn}}, \delta)$的内切圆，将查询 $B(q_{\text{knn}}, \delta)$的点转化为查询 $Q(q_{\text{knn}}, \delta)$ 范围内的点。通过设置初始距离 δ_0，逐步扩展查询区域，直到查询结果满足 K 个或者距离大于最大查询距离 δ。

初始距离 δ_0 设置和扩展策略对 KNN 查询算法至关重要。δ_0 过大或者或过小都会导致查询性能下降。为此本文提出 QML 的空间密度测量方法。QML 混合索引通过引入四叉树，尽可能地将数据进行等量划分，每个单元格 cell 包含的数据量不大于 Cell_max_count。因此本文可以做出合理假设：空间 cell 内数据点大致均匀分布，在此基础上计算 $\text{cell} = [l_x, u_x)[l_y, u_y)$ 包含数据的空间密度

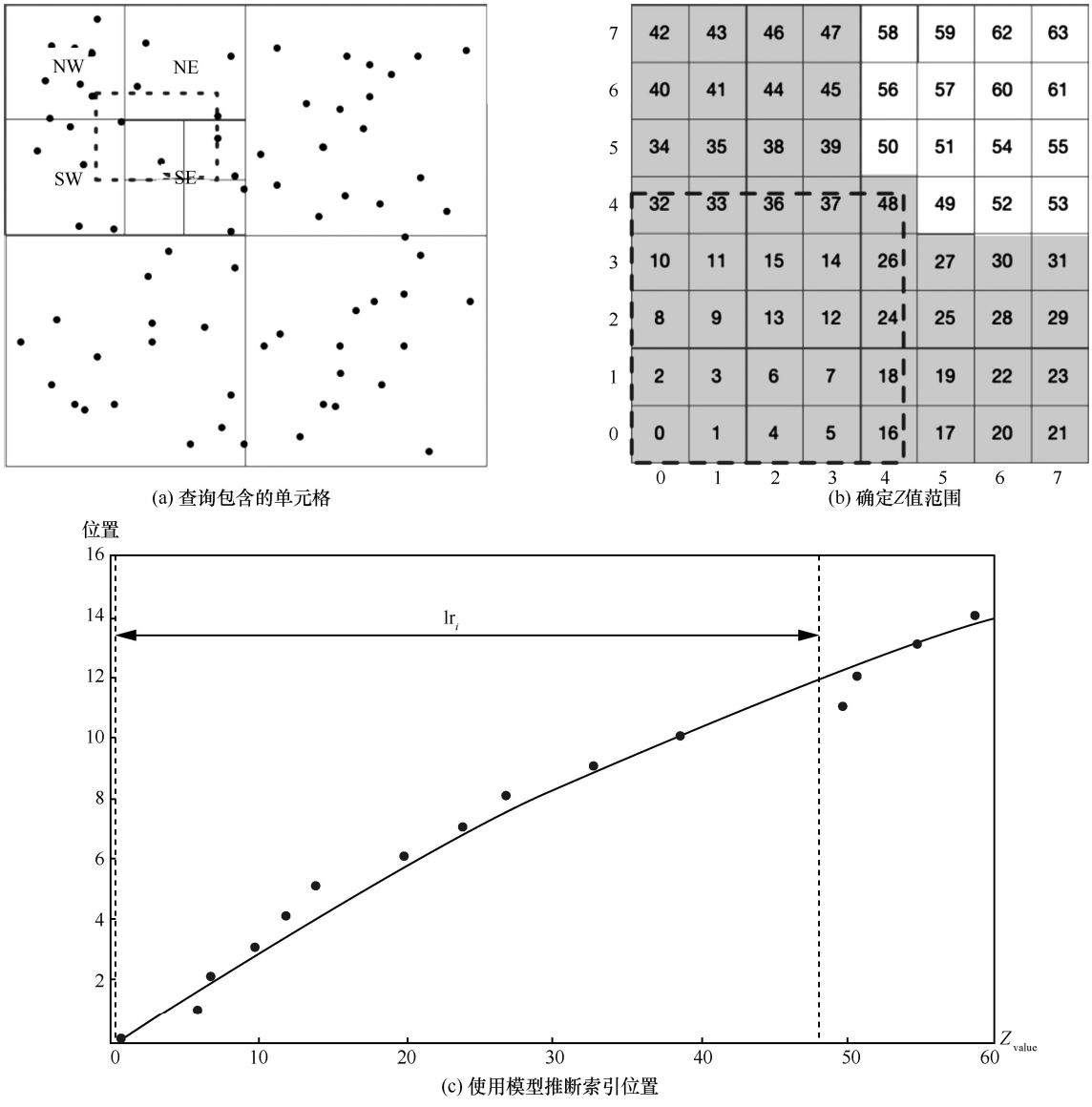
$$\rho = \frac{\text{cell\_count}}{(u_x - l_x)(u_y - l_y)} \tag{10}$$

当要选取区域 cell 内 K 个点时，可以根据空间密度估算距离 δ_0 为

$$\delta_0 = \sqrt{\frac{K}{\pi\rho}} \tag{11}$$

如图 5 所示，此时可以先获取区域 $B(q_{\text{knn}}, \delta_0)$ 内的所有数据点。假设区域 $B(q_{\text{knn}}, \delta_0)$ 内的数据点数量 K'小于要查询的 K，则需要对初始距离 δ_0 进行扩展。在空间密度保持不变的假设条件下，$B(q_{\text{knn}}, \delta_1)$ 的勒贝格测度是 $B(q_{\text{knn}}, \delta_0)$ 的 K/K'倍，因此可以推断出拓展后的距离为

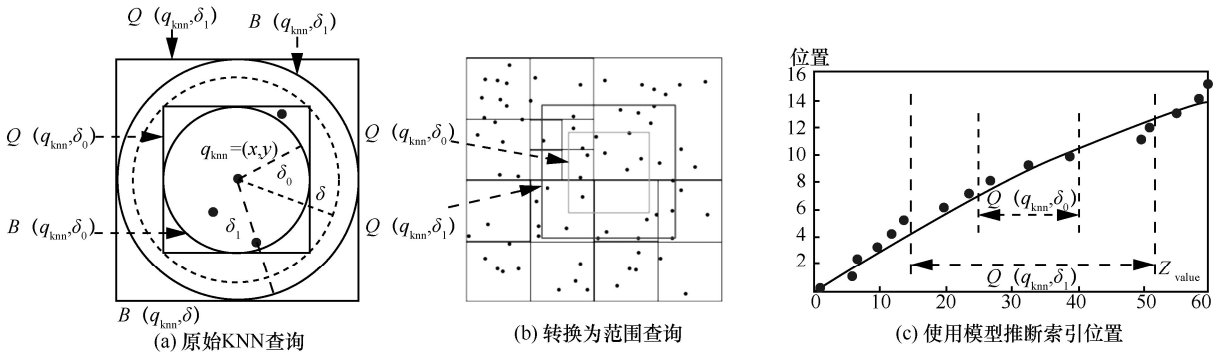
$$\delta_1 = \sqrt{\frac{K}{K'}}\delta_0 \tag{12}$$



(a) 查询包含的单元格　(b) 确定Z值范围

(c) 使用模型推断索引位置

图 4　范围查询示意



(a) 原始KNN查询　(b) 转换为范围查询　(c) 使用模型推断索引位置

图 5　KNN 查询示意

给定查询点 $q_{knn}=(x, y)$、最大查询距离 δ、返回结果数量 K，QML 索引的 KNN 查询过程如下。

步骤 1　初始化查询矩形 qr 为空，查找点 q_{knn} 所在的 cell，并根据 cell 的空间密度 ρ 和返回结果数量 K 计算半径距离 δ_0（式(11)）；

步骤 2　比较 δ_0 和最大查询距离 δ，选择较小

值为初始查询距离 δ'；

步骤 3　使用 q_{knn} 和 δ' 构建查询矩形 $\text{qr}'=[x-\delta',x+\delta'][y-\delta',y+\delta']$，然后计算 qr′和 qr∩qr′的差集得到需要查询的矩形区域 qr″；

步骤 4　查询矩形区域 qr″得到可能的结果队列 result′，遍历 result′判断数据点是否在圆形区域 $B(q_{\text{knn}},\delta')$范围内，如果在，则加入结果队列 L_r 中，否则加入等待队列 L_w 中；

步骤 5　对等待队列 L_w 中数据点做校验判断是否符合 $B(q_{\text{knn}},\delta')$ 的范围要求，符合则加入 L_r 中；

步骤 6　判断结果队列 L_r 中的数据量是否已经满足 K 个，满足则退出，不满足则使用式(12)重新计算 δ'，并将现在的 qr′赋值给 qr，再次执行查询操作。

6　QML 的数据更新

QML 的数据更新包括插入和删除操作，相应的更新算法同时支持单点更新和批量更新。

6.1　插入

QML 的本地模型是基于有序数据排列完成的，当有新的数据加入时需要对数据进行重新排序和本地模型训练。这样的操作会消耗一定的计算资源。为了避免频繁的排序和模型训练，QML 采用缓存插入的方式。当新的数据点要插入数据节点时，先加入等待插入的缓存队列，当缓存队列达到最大阈值时，再对节点进行重新排序和模型训练。

给定需要插入的数据点 $k=(x, y)$，QML 索引的插入过程如下。

步骤 1　通过 QML 索引根节点查找到 k 所在的单元格 cell，如果单元格 cell 为空则初始化该单元格；

步骤 2　使用 Z 顺序曲线映射函数 M 计算 k 的 Z 值 z_{value} 得到 k_z，使用单元格 cell 的本地模型集合 L 对 z_{value} 进行检索；

步骤 3　如果检索到 z_{value} 所在位置数值为空，则直接将 k_z 插入该点；如果检索到 z_{value} 不存在，则先将该数据点加入 cell 的缓存队列中，然后执行步骤 4 的分裂检查机制；

步骤 4　如果 cell 的数据量 size 大于 Cell_max_count 或者缓冲区数据量大于缓冲区阈值则合并数据集，并对当前单元格 cell 执行算法 1 的数据划分操作。

6.2　删除

考虑到更新效率，QML 索引在执行删除操作时不会直接删除索引，而是先将该索引处的数值置空，然后执行节点合并检查机制，当满足条件时再对单元格 cell 重新构建。

给定需要删除的数据点 $k=(x, y)$，QML 索引的删除过程如下。

步骤 1　通过根节点查找 k 所在的 cell，若 cell 不存在则直接返回 false，否则执行步骤 2；

步骤 2　通过 Z 顺序曲线映射函数 M 计算 k 的 Z 值得到 k_z，使用 cell 的本地模型集合 L 检索 k_z 的 z_{value}，若 z_{value} 存在且数值不为空，则将该索引处的数值置空，然后执行步骤 3 的合并检查机制；

步骤 3　获取 cell 的父节点 pCell，如果 pCell 的数据量小于 Cell_max_count，则将 pCell 赋值给 cell，并递归使用步骤 3 进行判断；当 pCell 数据量大于 Cell_max_count 时执行步骤 4 并发送当前的 cell；

步骤 4　将步骤 3 的输出结果命名为 newCell，若 newCell 与初始 cell 不同，即上层节点满足合并条件时，对 newCell 执行算法 1 的数据划分操作；如果 newCell 和 cell 是同一个，则比较 cell 的数据量和其索引长度大小，若两者差值大于最大阈值，则对当前 cell 执行算法 1 的数据分段和模型生成操作。

7　索引评估

本节从学习多维索引对比和实验验证 2 个方面对构建的 QML 索引进行评估。

7.1　学习多维索引对比

QML 索引实现了 ZM[8]、ML-Index[9]、LISA[10]、Flood[11]等学习多维空间索引所支持的各种功能，包括范围查询、KNN 查询、索引更新等，本文从数据布局、一维空间学习模型以及算法复杂度等角度对现有的学习多维空间索引进行了对比，如表 2 所示。

QML 索引的数据布局采用四叉树和 Z 顺序曲线结合的方式，根据设计的 DDSA 通过一次性数据遍历构建分段模型。QML 索引相比于其他学习多维索引的优势主要在功能实现和数据更新上。QML 索引的范围查询时间复杂度为 $O(\log n)+O(n)$，优于 LISA 索引的 $O(n^2)+O(n)$。QML 使用树状索引作为中间节点保证了数据更新（插入和删除）的时间复

表 2 学习多维索引比较

索引名称	投影策略/数据布局	一维空间学习模型	范围查询时间复杂度	KNN 查询时间复杂度	插入时间复杂度	删除时间复杂度	存储空间复杂度
ZM	Z 顺序曲线	RMI	$O(n)$	不支持	不支持	不支持	$O(n)$
ML-Index	基于 iDistance 的改进策略	RMI	$O(\log n)$	$O(\log n)$	不支持	不支持	$O(n)$
LISA	网格划分+基于勒贝格测度的投影	分段线性回归模型	$O(n^2)+O(n)$	$O(n \log n)$	$O(1)$	$O(n)$	$O(\log n)+O(n)$
Flood	基于成本模型的网格划分	分段线性回归模型	$O(n)$	支持（未讨论）	不支持	不支持	$O(n)$
QML	四叉树+Z 顺序曲线	基于 DDSA 构建的分段模型	$O(\log n)+O(n)$	$O(n \log n)$	$O(1)$	$O(1)$	$O(\log n)+O(n)$

杂度为 $O(1)$，相比其他索引具有更灵活的构建方式，能快速实现索引的动态更新。

7.2 实验验证

因为实验数据的限制，本文的实验验证部分主要是将 QML 索引和 R*-tree[13]、KD-tree、Quad-tree、UB-tree 进行比较。本节实验包括实验设置、数据分段算法对比实验、参数影响、数据更新表现、范围查询表现和 KNN 查询表现。

1) 实验设置

本文使用了如下的 3 个空间数据集作为测试数据集。

① Imis-3months 由专注于 AIS 技术和公共信息系统的公司 IMIS Hellas S.A. 收集。删除重复项后，保留 106 700 263 条记录。

② OpenStreetMap（OSM）[28] 由英国 Steve Coast 于 2004 年创建。本文从 OpenStreetMap 上获取了美国本土各类建筑、商店等标签的地理信息，去重后包含 2 490 799 条记录。

③ Uniform 是通过对一定区域内均匀分布的数据进行随机采样产生的数据集，去重后的数据为 20 000 000 条。

对比模型。为了验证本文提出的动态数据分段算法 DDSA 和 QML 混合空间索引的有效性，本文设计了如下的实验：将本文的 DDSA 与传统的 FSW 算法进行比较，将 QML 索引与 4 种现有的索引 R*-tree[13]、KD-tree、Quad-tree、UB-tree 进行对比实验。

评估指标。在数据分段算法实验中本文比较了 FSW 算法和 DDSA 在不同数据集下的表现。使用 3 个度量来评估数据切分算法的性能：平均误差 Average Error、分段数量 Number、算法耗时 Time。

在空间索引对比实验中本文使用 2 个度量来评估方法的性能：内存空间占用 Size、索引操作耗时 Time。对于每个数据集本文随机生成 5 000 个查询矩形 R 和 5 000 个 KNN 查询点 K，然后计算出每个索引类型进行查询操作的平均值并进行比较。

2) 数据分段算法对比实验

数据分段算法的效果与置信区间设置和数据集大小相关。为了综合评估 FSW 算法和 DDSA，本文在 3 个空间数据集 Imis-3months、OSM、Uniform 上分别选取记录数为{2 000, 4 000, 6 000, …, 50 000, 60 000}的测试集合 $M=\{m_0, m_1,\cdots, m_{11}\}$，在每个测试集 m_i 上设置置信区间{5, 10, 15, 20, 25, 30}进行多次实验，

最后取平均值作为测试集 m_i 的实验结果。在生成测试数据集时，本文按照四叉树划分算法进行等比例划分以更符合实际情况。通过比较 2 种算法在不同数据集上的平均误差、分段数量和算法耗时来评估算法的性能。

实验结果如图 6 所示。从图 6(a)中可以看出，DDSA 比 FSW 算法平均误差要小，经测算平均可以减少 12%～17%的误差。这是因为 DDSA 采用分段线性函数和二阶多项式函数对不同分布特征的数据分别进行拟合，更贴合数据实际分布规律。缩小平均误差可以减少 QML 索引中间查询结果队列的数量，优化查询速度。

从图 6(b)中可以看出，DDSA 相比于 FSW 算法可以有效减少分段函数的个数，测试数据集越大，DDSA 效果越好。经测算 DDSA 可以平均降低 7%～10%分段数。减少分段数量可以有效降低索引的存储空间和维护成本，同时也可以更快地查找到目标数据的本地模型，从而加快索引速度。

从图 6(c)中可以看出，DDSA 的时间消耗要比 FSW 算法多，平均会增加 13%～17%的耗时。这是因为 DDSA 需要先进行线性函数的验证，如果不符合线性分布会再进行二阶多项式函数的验证，所以

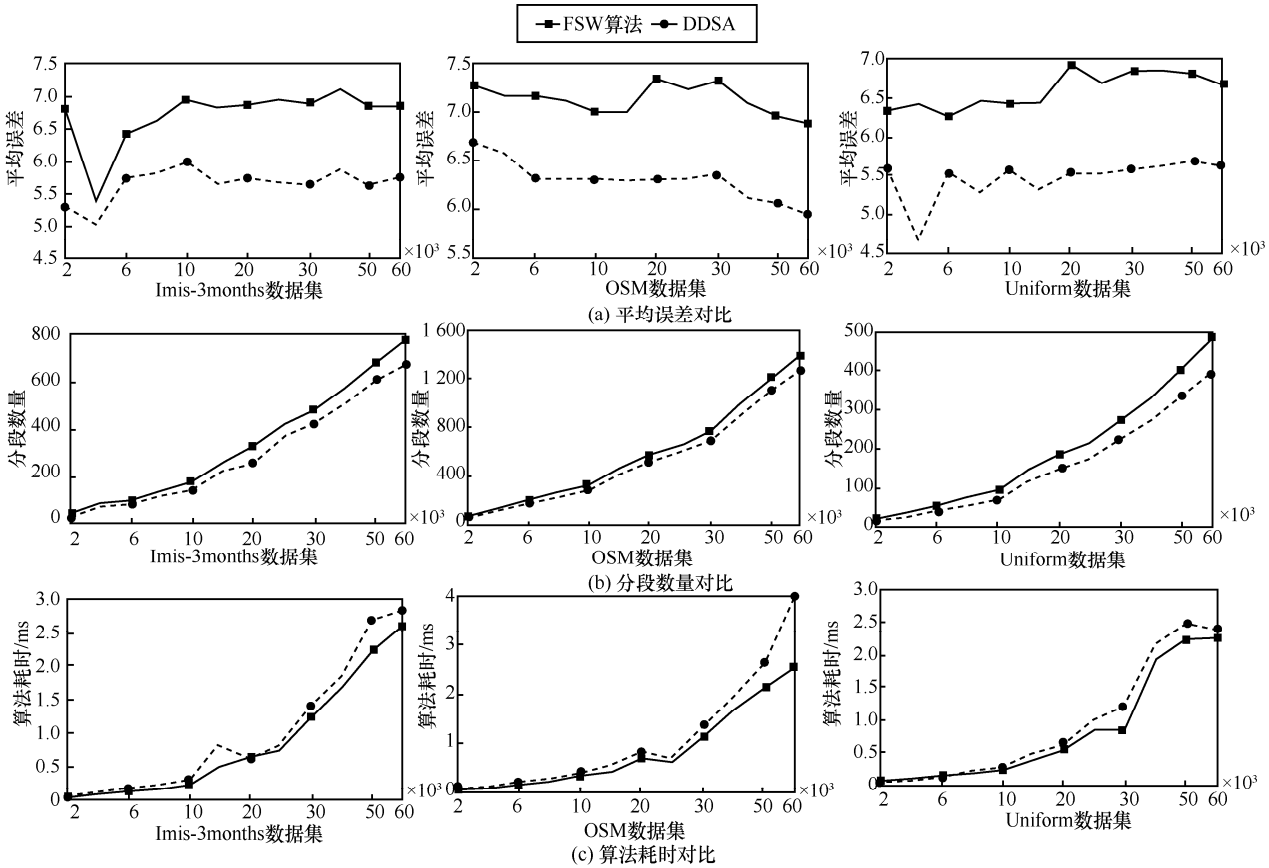


图 6　FSW 和 DDSA 平均误差、分段数量、算法耗时比较

会不可避免地带来时间的损耗。

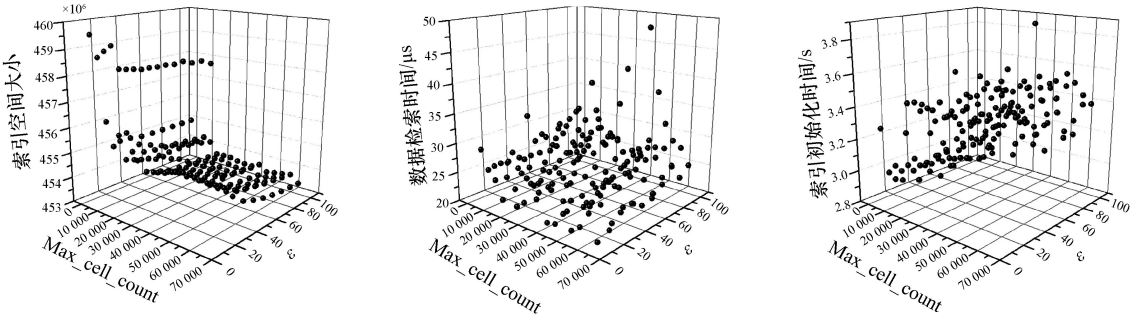
在 QML 检索应用中，对生成的数据分段先通过查找算法（以二分查找为例）找到对应的分段函数，然后通过该函数推测数据可能的位置，最后根据误差阈值进行校验（$\pm\varepsilon$）得到可能的结果队列。DDSA 相比于 FSW 算法对 QML 索引检索速度提高 10%～13%。为了保证 QML 索引性能，单元格最大阈值 Cell_max_count 应设置在[10 000, 30 000]范围内，过大或者过小都会降低索引性能。QML 索引的构建支持并发操作，当数据集比较大时，可以通过适当增加计算资源（1.13～1.17 倍）采用多线程的方式缓解 DDSA 的算法耗时。

3) 参数影响

影响 QML 索引性能的参数包括单元格最大阈值 Cell_max_count 和本地模型的置信区间 ε。为了研究这 2 个参数对 QML 索引空间大小、索引初始化时间和数据检索时间的影响，本文设计如下实验：从公开数据集 Imis-3months 中随机选取 40×10^6 条记录作为测试集，将 QML 的索引单元格最大阈值 Cell_max_count 设置在[2 048, 63 488] 范围内，将本地模型置信区间 ε 设置在[10, 100]范围内，进行多次实验后取平均值作为实验的最终结果。

实验结果如图 7 所示。从图 7 中可以看出，QML 单元格最大阈值 Cell_max_count 与 QML 索引空间大小成反比，与 QML 索引初始化时间成正比，对 QML 检索时间影响不大。这是因为随着 Cell_max_count 的增大，QML 索引的叶节点数量随之减少，所以需要的存储空间变小。但是每个单元格 cell 需要花费更多的时间构建本地模型，叶节点初始化和更新维护成本也会随之上升。当 Cell_max_count 增加时，QML 索引的本地模型集合 L 也会变大，但是 QML 的中间节点减少，所以整体的检索速度变化不大。

QML 检索时间和初始化时间随 ε 的增加逐渐波动上升。QML 索引空间大小随 ε 增加逐渐下降。当 ε 增加时，QML 索引本地模型的准确率会逐渐下降，这就使本地模型推算的结果队列增加，因此 QML 需要花费更多的时间对结果队列进行校验，导致检索速度成本上升。同时随着 ε 的增加，QML 的单位模型可以容纳更多的数据量，使本地模型集合的分段数量减少，因此可以加快分段函



(a) 索引空间大小 (b) 数据检索时间 (c) 索引初始化时间

图 7 QML 参数影响三维立体

数检索速度，减少模型集合空间大小。QML 索引准确率的影响要比分段数量带来的影响要大，这就使QML检索时间和初始化时间随着ε的增加呈现逐渐波动上升，QML 索引空间大小随着 ε 的增加逐渐下降。

综合来看，QML 的索引参数 Cell_max_count 和 ε 设置应充分考虑 QML 索引的具体应用场景。如果应用场景对索引空间占比要求较高，应适当调大 2 个参数，如果对索引更新性能要求较高，应适当调小 2 个参数。本文后续的实验中 QML 索引最大单元格阈值设置为 20 000，本地模型置信区间 ε 设置为 10。

4) 数据更新表现

为了比较 QML 索引与传统的数据索引 R*-tree、KD-tree、Quad-tree、UB-tree 在数据更新时的性能表现，本文设计如下实验：从数据集 Imis-3months、Uniform 中按照区域选取记录数为 4×10^6、8×10^6、12×10^6、16×10^6、20×10^6 的数据构成测试数据集 $D=\{d_0,d_1,d_2,d_3,d_4\}$，对于每个数据集 d 选取其中 50%构建初始数据集 I，另外的 50%作为额外的数据集 E 插入索引中。从 $I\cup E$ 中随机抽选 50%的点作为要删除的数据集 D。以上 3 种配置上进行实验。Configuration Init 为初始化构建实验，使用 I 来构建所有的 5 个索引。Configuration AI 为插入测试实验，使用 E 来进行插入操作。Configuration AD 为删除测试实验，使用 I 构建索引并插入 E 后，删除 D 中的数据。

实验结果如图 8 所示。在 Configuration Init 实验中，所有的索引结构耗时都随着数据量的增加呈线性增长。当数据量激增时，Quad-tree 的初始化性能会降低很多。除了 Quad-tree，QML 索引耗时始终少于其他索引结构。在 Configuration AI 和 Configuration AD 实验中，QML 索引呈现出近似 $O(1)$ 的时间复杂度变化趋势，除 Quad-tree 外的其他索引呈现 $O(n)$ 的线性变化。因为 QML 使用了四叉树作为中间节点，所以 QML 索引与 Quad-tree 的数据更新操作效率相近。与最典型的树形索引 R*-tree 对比，在 Configuration Init 中，QML 索引平均耗时减少 40%；在 Configuration AI 中，QML 索引平均耗时减少约 80%；在 Configuration AD 中，QML 索引平均耗时减少 40%～60%。

图 9 显示了 5 个索引结构在不同数据集下的内存消耗。从图 9 中可以看出，QML 索引相比于传统的索引结构，存储相同的数据量时可以占用更少的内存空间。而且随着数据量的增加效果越明显。相比于 R*-tree，平均可以减少约 33%的存储空间。

5) 范围查询表现

为了比较 QML 索引与 4 个传统索引范围查询的性能，本文设计如下实验：对于每个数据集本文随机生成 5 000 个数据点，并构建查询矩形 R，分别使用 5 个索引进行点查询和范围查询，记录耗时。

实验结果如图 10 所示。从图 10 中可以看出，QML 索引在点查询方面呈现出相当大的优势。在范围查询中与其他索引基本持平。这是因为 QML 索引 cell 的本地模型在推测目标数据位置时存在一定的误差（这也是需要设置置信区间的原因）。误差的存在导致 QML 索引必须对模型检索的结果进行遍历校验。当检索的矩形区域范围扩大时，包含的数据点也会增多，QML 索引本地模型检索出的数据量也会增多，因此需要花费更多的时间进行数据校验。因为点查询过程只需要检索一个有效数据，所以 QML 本地模型推算到的数据误差也就能保证在$[-\varepsilon,\varepsilon]$范围内。当 QML 索引本地模型检索的数据结果长度小于传统树形索引结构需要遍历的深度时，QML 索引就能表现出优于传统树形索引的性能。

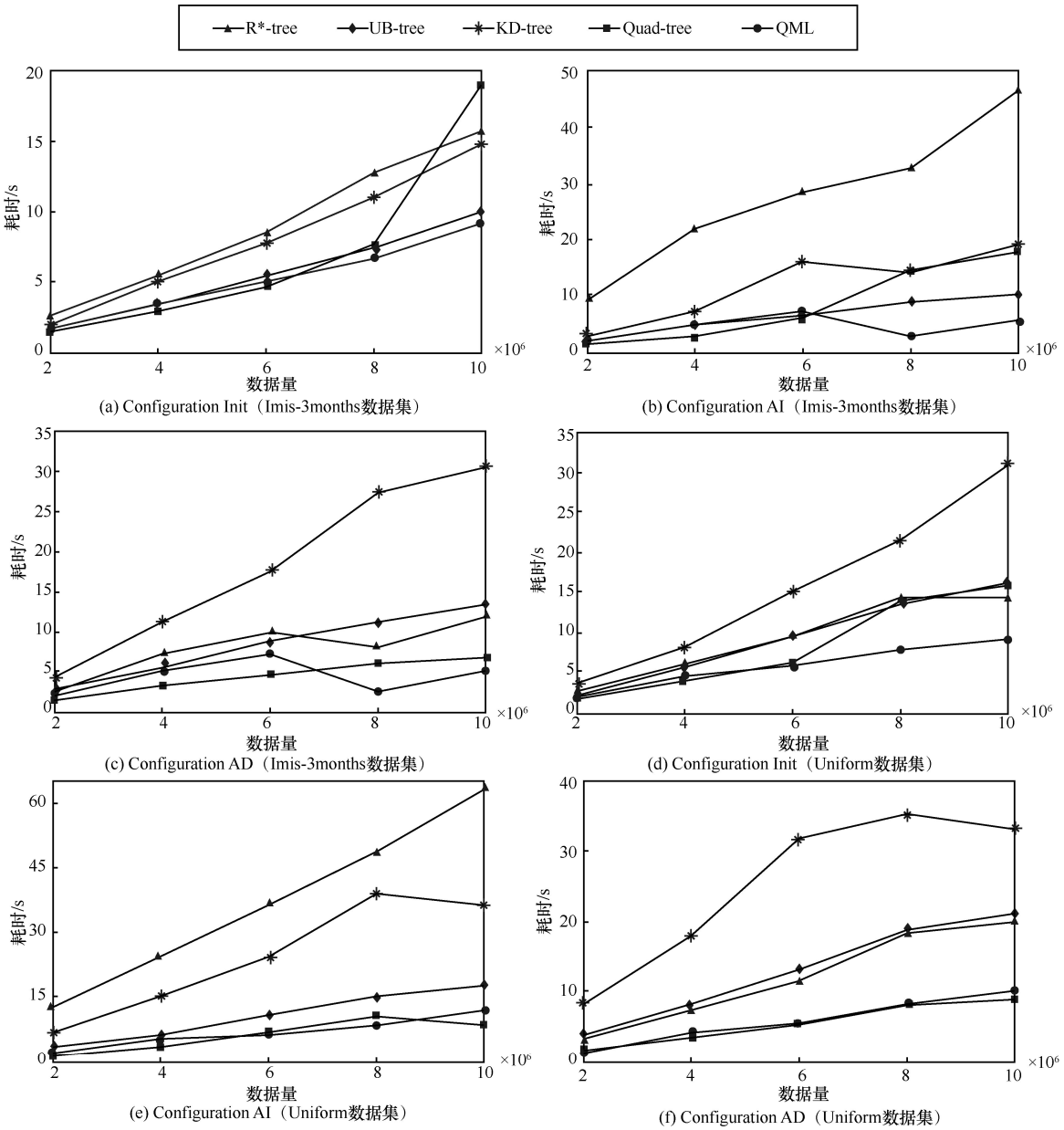


图 8　数据更新处理耗时比较

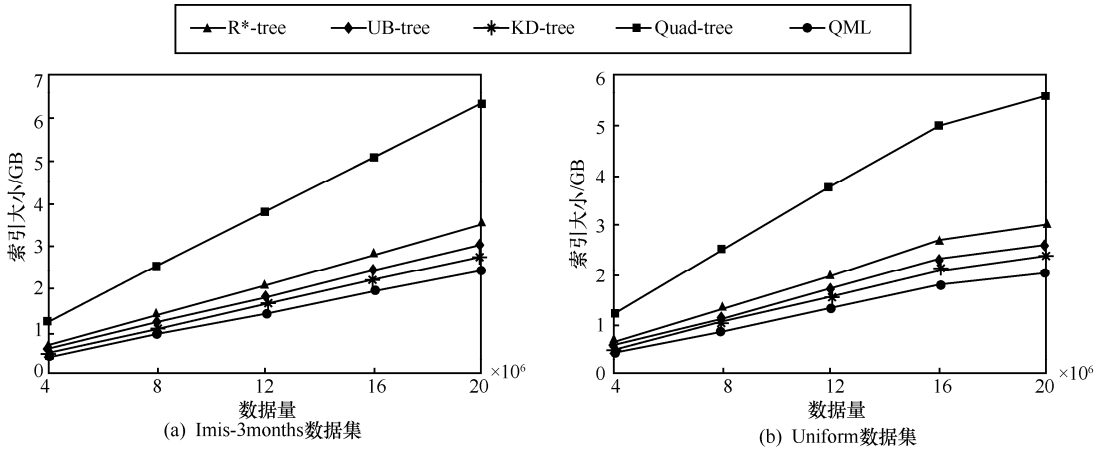


图 9　索引内存消耗比较

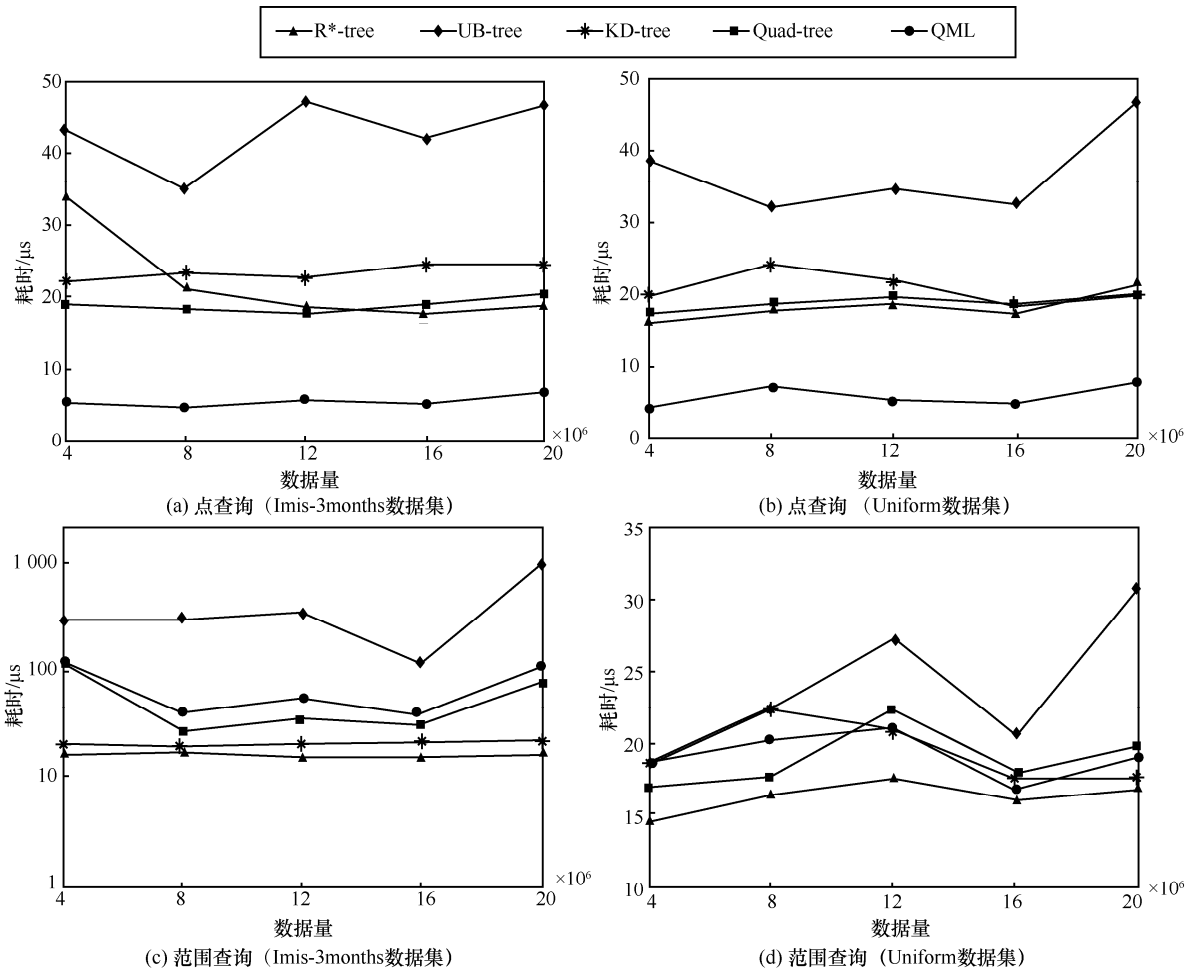


图 10　点查询和范围查询比较

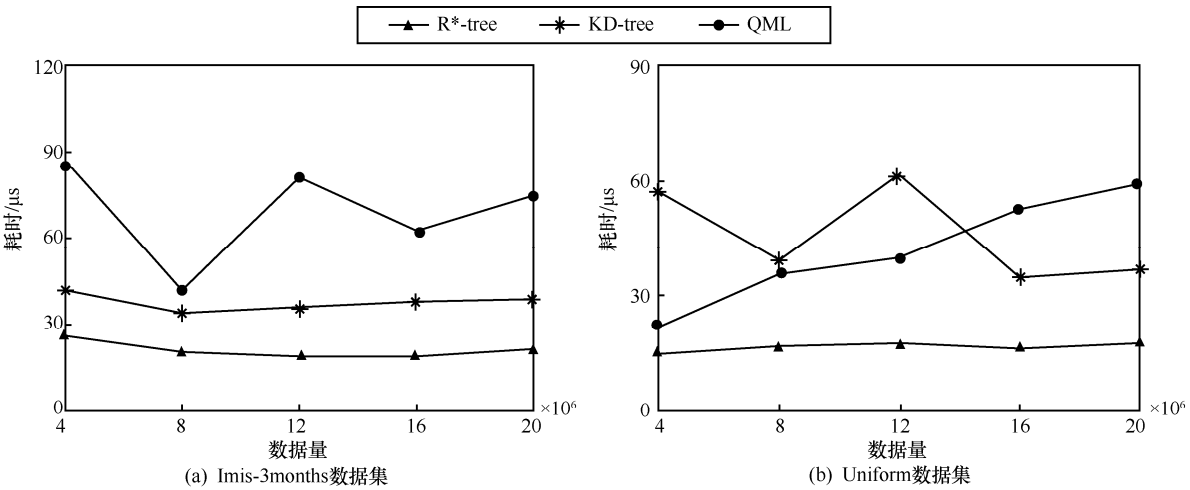


图 11　KNN 查询比较

6) KNN 查询表现

对于每个数据集，本文随机生成 5 000 个 KNN 查询点 K，然后在索引 QML、R*-tree 和 KD-tree 进行检索，记录返回的时间。最后对所有的检索结果取平均值。

实验结果如图 11 所示。从图 11 中可以看出，QML 索引的 KNN 查询结果相比于 R*-tree 和 KD-tree 不是很稳定。原因在于 QML 索引的 KNN

查询性能与数据分布特征有关。QML 索引的 KNN 查询算法引入了空间密度的概念，通过计算最小单元格 cell 的均匀空间密度计算初始空间距离 δ_0。当单元格 cell 内的数据有较大数据偏移时，会导致初始的空间距离 δ_0 计算不精确，从而增加查询周期。同时 cell 的大小也会对空间密度产生一定影响。如果 Cell_max_count 偏大，必然会导致单元格空间密度的失真。如何增加 QML 索引 KNN 查询的稳定性是接下来的工作重点之一。

7.3　高维空间的拓展

本文提出的 QML 索引可以很容易拓展到 n 维空间领域。对于 n 维数据集可以使用 2^n 叉树代替原有的四叉树结构，例如三维数据可以使用八叉树代替四叉树。使用 n 维子空间 space 作为叶节点存储单位，space 存储单位阈值的数据。Z 顺序曲线变换可以将 n 维数据降维到一维。DDSA 可以根据数据的分布规律增加多项式函数的类型以适应更复杂的曲线，比如三阶多项式函数。

8　结束语

在本文中提出了一种新的空间数据学习索引结构——QML。通过本文设计的动态数据分段算法 DDSA，结合四叉树和 Z 顺序曲线构建的 QML 索引可以利用空间数据分布规律优化检索速度。DDSA 保证了通过一次性的数据遍历构建索引模型。通过引入四叉树和 Z 顺序曲线，以 cell 单元格为最小单位训练本地模型保证了单位索引相互之间的独立性。QML 索引在存储空间和查询速度上都达到了一个理想的效果，并能够灵活快速地实现索引构建和更新。QML 采用四叉树结构作为内部节点，因此可以进一步支持使用压缩方案与节点级压缩技术来减少索引的大小。它是学习索引在空间数据集上的一种新的尝试。

参考文献：

[1] KRASKA T, BEUTEL A, CHI E H, et al. The case for learned index structures[C]//Proceedings of the 2018 International Conference on Management of Data. New York: ACM Press, 2018: 489-504.

[2] ZHANG H C, ANDERSEN D G, PAVLO A, et al. Reducing the storage overhead of main-memory OLTP databases with hybrid indexes[C]//Proceedings of the 2016 International Conference on Management of Data. New York: ACM Press, 2016: 1567-1581.

[3] GALAKATOS A, MARKOVITCH M, BINNIG C, et al. FITing-tree: a data-aware index structure[C]//Proceedings of the 2019 International Conference on Management of Data. New York: ACM Press, 2019: 1189-1206.

[4] DING J L, MINHAS U F, YU J, et al. ALEX: an updatable adaptive learned index[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 969-984.

[5] FERRAGINA P, VINCIGUERRA G. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds[C]//Proceedings of the VLDB Endowment. Cairo: Morgan Kaufmann, 2020: 1162-1175.

[6] TANG C Z, WANG Y Y, DONG Z Y, et al. XIndex: a scalable learned index for multicore data storage[C]//Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2020: 969-984.

[7] 高远宁, 叶金标, 杨念祖, 等. 基于中间层的可扩展学习索引技术[J]. 软件学报, 2020, 31(3): 620-633.
GAO Y N, YE J B, YANG N Z, et al. Middle layer based scalable learned index scheme[J]. Journal of Software, 2020, 31(3): 620-633.

[8] WANG H X, FU X Y, XU J L, et al. Learned index for spatial queries[C]//Proceedings of 2019 20th IEEE International Conference on Mobile Data Management (MDM). Piscataway: IEEE Press, 2019: 569-574.

[9] DAVITKOVA A, MILCHEVSKI E, MICHEL S. The ML-Index:a multidimensional, learned index for point, range, and nearest-neighbor queries[C]//Proceedings of the 23rd International Conference on Extending Database Technology (EDBT). Copenhagen: OpenProceedings.org, 2020: 407-410.

[10] LI P F, LU H, ZHENG Q, et al. LISA: a learned index structure for spatial data[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 2119-2133.

[11] NATHAN V, DING J L, ALIZADEH M, et al. Learning multi-dimensional indexes[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 985-1000.

[12] BECKMANN N, KRIEGEL H P, SCHNEIDER R, et al. The R*-tree: an efficient and robust access method for points and rectangles[J]. ACM SIGMOD Record, 1990, 19(2): 322-331.

[13] RIGAUX P, SCHOLL M, VOISARD A. An introduction to spatial databases[C]//Spatial Databases. Amsterdam: Elsevier, 2002: 1-28.

[14] NIEVERGELT J, HINTERBERGER H, SEVCIK K C. The grid file: an adaptable, symmetric multikey file structure[C]//ACM Transactions on Database Systems. New York: ACM Press, 1984: 38-71.

[15] GUTTMAN A. R-trees: a dynamic index structure for spatial searching[C]//Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD'84. New York: ACM Press, 1984: 47-57.

[16] BAYER R. The universal B-tree for multidimensional indexing: general concepts[C]//Lecture Notes in Computer Science. Berlin: Springer, 1997: 198-209.

[17] RAMSAK F, MARKL V, FENK R, et al. Integrating the UB-tree into a database system kernel[C]//Proceedings of the 26th International Conference on Very Large Data Bases. Cairo: Morgan Kaufmann, 2000: 263-272.

[18] 张洲, 金培权, 谢希科. 学习索引: 现状与研究展望[J]. 软件学报,

2021, 32(4): 1129-1150.
ZHANG Z, JIN P Q, XIE X K. Learned indexes: current situations and research Prospects[J]. Journal of Software, 2021, 32(4): 1129-1150.

[19] 孟小峰, 马超红, 杨晨. 机器学习化数据库系统研究综述[J]. 计算机研究与发展, 2019, 56(9): 1803-1820.
MENG X F, MA C H, YANG C. Survey on machine learning for database Systems[J]. Journal of Computer Research and Development, 2019, 56(9): 1803-1820.

[20] 李国良, 周煊赫, 孙佶, 等. 基于机器学习的数据库技术综述[J]. 计算机学报, 2020, 43(11): 2019-2049.
LI G L, ZHOU X H, SUN J, et al. A survey of machine learning based database Techniques[J]. Chinese Journal of Computers, 2020, 43(11): 2019-2049.

[21] 柴茗珂, 范举, 杜小勇. 学习式数据库系统: 挑战与机遇[J]. 软件学报, 2020, 31(3): 806-830.
CHAI M K, FAN J, DU X Y. Learnable database systems: challenges and Opportunities[J]. Journal of Software, 2020, 31(3): 806-830.

[22] 孙路明, 张少敏, 姬涛, 等. 人工智能赋能的数据管理技术研究[J]. 软件学报, 2020, 31(3): 600-619.
SUN L M, ZHANG S M, JI T, et al. Survey of data management techniques powered by artificial Intelligence[J]. Journal of Software, 2020, 31(3): 600-619.

[23] JAGADISH H V, OOI B C, TAN K L, et al. iDistance[J]. ACM Transactions on Database Systems, 2005, 30(2): 364-397.

[24] GARCIA E K, GUPTA M R. Lattice regression[C]//Proceedings of the 22nd International Conference on Neural Information Processing Systems (NIPS'09). New York: Curran Associates Inc., 2009: 594-602.

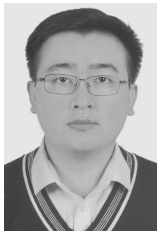
[25] GULLO F, PONTI G, TAGARELLI A, et al. A time series representation model for accurate and fast similarity detection[J]. Pattern Recognition, 2009, 42(11): 2998-3014.

[26] LIU X Y, LIN Z J, WANG H Q. Novel online methods for time series segmentation[J]. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(12): 1616-1626.

[27] XU Z H, ZHANG R, KOTAGIRI R, et al. An adaptive algorithm for online time series segmentation with error bound guarantee[C]//Proceedings of the 15th International Conference on Extending Database Technology - EDBT'12. New York: ACM Press, 2012: 192-203.

[28] HAKLAY M, WEBER P. OpenStreetMap: user-generated street maps[J]. IEEE Pervasive Computing, 2008, 7(4): 12-18.

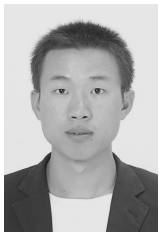
[作者简介]



崔栋（1991− ），男，山东潍坊人，北京邮电大学博士生，主要研究方向为学习索引、移动互联网安全。

温巧燕（1959− ），女，陕西户县人，博士，北京邮电大学教授、博士生导师，主要研究方向为现代密码理论、量子密码与量子信息、网络安全技术等。

张华（1978− ），女，吉林四平人，博士，北京邮电大学副教授、博士生导师，主要研究方向为网络安全、隐私保护等。



王华伟（1989− ），男，河南驻马店人，博士，北京邮电大学在站博士后，主要研究方向为区块链安全、去中心化的身份安全认证技术。