针对《A Learned Index for Exact Similarity Search in Metric Spaces》的复现与改进

摘要

本研究围绕度量空间中的精确相似性搜索展开，旨在复现并改进《A Learned Index for Exact Similarity Search in Metric Spaces》中提出的 LIMS 索引。随着数据量与维度的增长，传统树状结构索引在相似性搜索中面临挑战，LIMS 作为一种基于机器学习构建索引的创新方法，展现出优于传统方法的性能。本文在 LIMS 的数据约简阶段引入新的排除规则，有效提高了算法剪枝率，减少假阳性面积，进而提升查询性能。实验在 Windows 11 的 wsl2 子系统（Ubuntu - 20.04）中进行，使用图像色彩直方图特征数据集。结果表明，改进后的方法在不同支撑点数目下，搜索时间均显著优于 LIMS 原始版本，为度量空间相似性搜索提供了更高效的解决方案。

关键词：度量空间；相似性索引；机器学习；范围查询

1 引言

在应对存储与检索复杂、非结构化或高维数据的任务时，度量空间是常用的手段。度量空间是一个通用空间，它不需要任何特定的数据表示，而只需要一个满足三个性质的距离函数，即正定性、对称性和三角形不等式。此类方法基于距离信息进行处理，这在存储对象缺乏可用于组织的自然线性排序的情况下，具有显著的实用价值。专注于此类数据存储与检索的研究领域，被定义为相似性搜索。

相似性搜索已经成为多媒体信息系统基于内容搜索的基本需求，其性能已经成为衡量多媒体系统查询功能的重要指标。常见的度量空间相似性查询有范围查询和最近邻 (K-NN) 查询两种，这两种查询大多都是基于树状结构索引进行加速，目前已有许多成熟的研究。然而，随着数据量和数据维度的增加，树状结构索引也面临越来越多的挑战。一方面，使用这些索引的查询处理需要遍历树结构中遍历的许多索引节点。另一方面，树状索引对存储复杂和大型对象的数据集施加了不可忽视的存储压力，如图像和音频特征数据。

现阶段，最为先进的度量空间索引方法运用支撑点进行划分，将度量空间细分为基于距离的段层次结构。在解析查询过程中，仅需计算查询与代表数据库子集的每个支撑点之间的距离。优先队列依据区域包含最近对象的可能性大小，对需搜索的区域进行排序。显然，这种方法能够大幅削减距离计算的总量。不过，任何此类技术唯有在优先级队列的构建模式能切实保障对包含所需结果的分区进行快速访问时，方能切实发挥效用。

《A Learned Index for Exact Similarity Search in Metric Spaces》提出了一种基于机器学习算法构建索引的方法，名为 LIMS，通过应用一系列预测来处理查询任务。这一创新改变了

索引构建与查询评估的传统范式，塑造出全新的性能特征，并且在诸多应用场景下，无论是检索效率还是有效性，均优于传统的相似性搜索方法。在原有工作基础上，本文在数据约简阶段引入了排除规则，提高了算法在约简阶段的剪枝率，减少了假阳性面积，从而提高了查询性能。

2 相关工作

近年来，大量的研究工作聚焦于利用机器学习来强化标准数据库模型与方法，探讨其可行性。其中，[6] 提出，现有的索引结构可被视作一种模型结构，从理论层面来讲，无论其数学基础如何，都能够被其他任何模型替代。具体而言，机器学习模型经过训练后，能够达成与回答查询相同的目的，也就是将查询对象精准分类至最适配的类别，而这个类别代表着一个子节点，与此同时，还展现出诸多性能优势。他们还指出，除了其他方面的优势，学习得到的模型相较于传统方法，能更好地适配数据分布。与标准的 B-Tree 实现相比，这种方法提供了许多好处，特别是在查找时间和索引大小方面。然而，由于这个学习的模型本质上是概率的，它给检索过程引入了不确定性，从而略微降低了精度和召回率，并消除了所有查询数据将被检索的任何保证。

在接下来的几年里，学习指数的概念在许多不同的方向上进行了进一步的研究。在 [10] 中，简单的线性回归模型和 B+ 树搜索的组合允许插入和非连续的数据存储。相反，[3] 引入了一个名为 PGMindex 的纯学习模型，它也可以保证查询操作中的 I/O-最优性，并学习最优数量的线性模型。然而，这些都没有应用于度量空间或复杂数据。

ZM 索引 [14] 采用 z 阶空间填充曲线 [12] 来建立所有点的排序关系，然后调用 RMI 来支持点和范围查询。通过 z 阶曲线的一个良好的几何性质，即单调排序，保证了其正确性。然而，ZM 索引在细化阶段需要检查许多不相关的点，这对于高维空间会变得更糟。它不支持 kNN 查询和索引更新。

递归空间模型指数（RSMI）建立在 RMI 和 ZM 的基础上 [9]。提出了一种递归划分策略，对原始空间进行划分，然后根据预测对数据进行分组。这导致了一个学习到的点分组，这不同于 RMI，它首先修复数据布局，并训练一个模型来估计位置。对于每个分区，RSMI 首先将点映射到秩空间中，然后调用 ZM 来支持点、范围和 kNN 查询。但是，这并不能保证范围和 kNN 查询的正确性。此外，由于 RSMI 仍然是基于空间填充曲线的，因此 RSMI 的良好性能仅局限于低维空间。

LISA [7] 是一种学习过的空间数据索引结构，与 ZM 相比，可以有效地减少假阳性数，1) 根据数据分布将原始空间划分为网格单元；2) 使用部分单调映射函数对数据进行排序，并根据映射值重新排列数据布局；3) 将大查询范围分解为多个小查询范围。LISA 有一个动态的数据布局作为 RSMI，但是通过模型的单调性可以保证范围和 kNN 查询的正确性得到保证。然而，它在低扫描开销方面的优势在于昂贵的检查程序和高索引构建时间。而基于网格的划分策略使其不适合用于高维空间。

机器学习方法也出现在关系数据库管理系统中，它们可以用于优化查询执行。例如，为关系数据库 [14] 构建自动查询优化器，或通过学习到的基数预测 [7] 来改进成本估计。学习索引背后的思想也被扩展到创建学习存在索引 [9]，它利用索引内和索引外元素之间的差异来进一步压缩索引。在学习的反向索引中，采用分层的机器学习模型，以牺牲性能为代价来减

小索引的大小。

与 LISA 类似，Flood [8] 也沿着维度将数据空间划分为网格单元格，这样对于每个维度，每个分区中的点的数量近似相同。Flood 假设有一个已知的查询工作负载，并利用示例查询来学习索引维度和分区数量的最佳组合。一旦了解了这些信息，Flood 就会维护一个表来记录每个单元格中第一个点的位置。在查询时，Flood 会为每个维度调用 RMI 来识别与查询相交的单元格，并查找单元格表以查找相应的记录。然而，它不能有效地适应相关的数据分布和倾斜的查询工作负载。Tsunami [2] 利用查询倾斜扩展了 Flood，将数据空间划分到一些区域，然后根据数据相关性进一步划分每个区域。然而，简单地选择维度子集会随着维度的增加而降低性能。这些研究没有进一步讨论，因为我们没有假设一个已知的查询工作负载。

ML index [1] 结合了 iDistance [5] 和 RMI 的思想。它首先将数据划分为集群，然后将集群中心标识为参考点。在所有数据点根据到参考点的距离在一维空间中表示后，就可以应用 RMI。与 iDitacte 不同，ML index 使用缩放值而不是常量来扩展数据范围。但是，沿固定半径的点在变换后具有相同的值，导致需要检查许多不相关的点。ML index 不支持数据更新。需要值得注意的是，ML index 不能直接应用于度量空间，因为参考点的选择是由 KMeans 算法 [16] 实现的。作为集群中点的平均值的参考点可能不在数据集中。在度量数据集 [13] 中创建“人工”对象并不总是可能的。

与在多维数据上找到一个排序顺序，然后学习 CDF 不同，一种基于强化学习的空间数据 r-树（RLR-Tree）[4] 的使用机器学习技术来改进经典的 r-树索引。RLR-Tree 不依赖于手工制作的启发式规则，而是建模了 R-tree 中的两个基本操作，即选择一个子树进行插入和分割一个节点，作为马尔可夫决策过程 [11]，因此可以应用强化学习模型。因为它不需要修改 r-树和查询处理算法的基本结构，所以在当前的数据库系统中比学习到的索引更容易部署。然而，由于维数的诅咒，一个叶节点（即使是在一个最优的 r 树中）的最小边界矩形（MBR）几乎可以和整个数据空间一样大，因此 r 树变得无效。与 RLR-Tree 类似，Qd-Tree [17] 使用强化学习，根据给定的查询工作负载优化 kd-tree 的数据划分策略，但也遇到了同样的问题。这些研究没有被进一步讨论，因为它们超出了我们的范围。

应该指出的是，将学习模型迅速引入数据库领域引起了许多关注，即数据组织和检索方面的这种转变是否可以被证明是合理的，因此值得进一步考虑。[11] 已经部分地解决了这个问题，他们引入了一个基准来比较传统索引系统及其学习到的衍生系统的性能。此外，在 [15] 中，已经对数据库系统中的机器学习进行了全面的研究。这两篇论文都为数据库领域中机器学习方法的采用提出了强有力的论据。但这些索引均没有应用到度量空间上，这是因为度量空间中没有维度信息和坐标，唯一能应用的只有数据之间的距离信息，约简规则只能基于三角不等式。

3 本文方法

在本节中，我们首先概述 LIMS 的索引结构，然后介绍本文的改进方法。

3.1 本文方法概述

LIMS 由三个部分组成：数据聚类和支撑点选择，基于支撑点的映射函数和相关的二值关系，以及等级预测模型。图 1给出了在与欧氏距离相关联的度量空间中的 LIMS 索引结构的概

述，尽管其他度量空间也适用于 LIMS。LIMS 首先将基础数据划分为一组集群，例如图 1(a) 中的 2 个集群，以便每个集群遵循相对统一的数据分布，然后为每个集群选择一组与数据相关的支撑点，例如 $O_1(1)$、$O_2(1)$ 和 $O_3(1)$。然后，LIMS 将为每个集群分别维护一个学习到的索引。由于每个集群的索引结构都是相同的，因此我们以集群 C_2 为例。LIMS 计算从集群中的每个对象到精心选择的支撑点的距离，例如，图 1(b). 中的 $dist \cdot O_3(2)$ 存储了从每个支撑点到相应对象的最大和最小距离，例如，$distmax_3^{(2)}$ 和 $distmin_3^{(2)}$，以支持高效的查询。

由于对象是根据距离值排序的，LIMS 可以学习一系列秩预测模型，如图 1(c) 中的 $RP_3^{(2)}$，以便快速计算给定对象到支撑点距离的对象的秩。在此之后，将调用一个定义良好的基于支撑点的映射函数 M，将每个对象转换为一个有序的集合。我们称这个集合中的元素为 LIMS 值。最后，我们在物理上按顺序维护在磁盘上的所有数据对象，LIMS 值和磁盘页面中数据对象地址之间的关系可以通过另一个秩预测模型学习，如 $RP^{(2)}$。在接下来的内容中，我们首先关注每个聚类的特定学习索引结构，然后再转向聚类和支撑点选择方法。换句话说，我们假设数据空间已经被分区，并且每个集群中的支撑点已经被确定。

查询过程的伪代码如 **Algorithm 1** 所示。

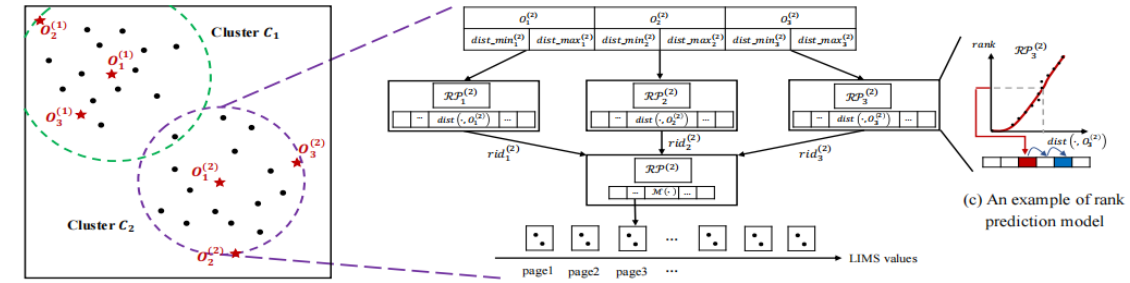
3.2 改进方法

可以看到 LIMS 的查询过程中一开始（**line 2-line 6**）便利用三角不等式进行约简（详见 **Definition 1** 和 **Definition 2**）。使用该性质可以在只跟支撑点做距离计算的情况下排除大多数数据，从而减少计算量。然而，LIMS 只考虑到了数据的排除情况，当查询 q 和支撑点 O 距离相近，且查询半径 r 设置得较大的情况下，支撑点 O 则有可能落在查询半径内，此时若有数据 s 满足 $d(O,s)+d(O,q) \geq r$，则有 $d(q,s) \leq r$，此时亦可在不与数据做距离计算的情况下，把数据判定为查询结果。该规则如图 2所示。

Definition 1. 定义二元组 $(U, dist)$,其中 U 是一组集合,$dist$ 是一个距离函数，$p_1, p_2, p_3 \in U$，满足三个性质：

1. $dist(p1, p2) > 0$, if $p_1 = p_2, dist(p_1, p_2) = 0$
2. $dist(p_1, p_2) = dist(p_2, p_1)$
3. $dist(p_1, p_2) + dist(p_2, p_3) \geq dist(p_1, p_3)$

Definition 2. 给定一个查询对象 q，查询半径 r 和数据集 P，返回 P 中在 q 的距离 r 内的所有对象，即 $(q, r) = \{p \in P | dist(p, q)\ r\}$。此时令 $O_j^{(i)}$ 为 $i-th$ 集群 $j-th$ 支撑点，根据 **Definition 1**, 得 $dist(O_j^{(i)}, q) - r \leq dist(O_j^{(i)}, p) \leq dist(O_j^{(i)}, q) + r$.



(a) Data clustering and pivots in the metric space

(b) Index structure for cluster C_2

(c) An example of rank prediction model

图 1. LIMS 索引结构图

Algorithm 1: Range Query

Input: q: a query object; r: a query radius
Output: $\mathcal{S}$: objects in P satisfying $dist(p,q) \leq r$

Let $flag[K]$ be an array of all $TRUE$, $rid\_min[K][m], rid\_max[K][m]$ be arrays of all 0;
for *each cluster* C_i **do**
  for *each pivot* $O_j^{(i)}$ **do**
    if $dist(O_j^{(i)}, q) > dist\_max_j^{(i)} + r$ *OR* $dist(O_j^{(i)}, q) < dist\_min_j^{(i)} - r$ **then**
      $flag[i] = FALSE$;
      break;
for *each TRUE cluster* C_i **do**
  for *each pivot* $O_j^{(i)}$ **do**
    $r_{min} \leftarrow \max\{dist(O_j^{(i)}, q) - r, dist\_min_j^{(i)}\}$;
    $r_{max} \leftarrow \min\{dist(O_j^{(i)}, q) + r, dist\_max_j^{(i)}\}$;
    $rank'_{min}, rank'_{max} \leftarrow \mathcal{RP}_j^{(i)}(r_{min}), \mathcal{RP}_j^{(i)}(r_{max})$;
    $rank_{min} \leftarrow ExpSearch(rank'_{min}, r_{min})$;
    $rank_{max} \leftarrow ExpSearch(rank'_{max}, r_{max})$; /*assume $rank_{min} < rank_{max}$, otherwise discard C_i */;
    $rid\_min[i][j] \leftarrow rid_j^{(i)}(rank_{min})$;
    if $D_j^{(i)}[rank_{max}] = r_{max}$ **then**
      $rid\_max[i][j] \leftarrow rid_j^{(i)}(rank_{max})$;
    else
      $rid\_max[i][j] \leftarrow rid_j^{(i)}(rank_{max} - 1)$;
generate LIMS-value ranges $\mathcal{R}$ based on rid_min and rid_max;
for *each range* $I \in R$ **do**
  $lbound', ubound' \leftarrow \mathcal{RP}^{(i)}(I.left), \mathcal{RP}^{(i)}(I.right)$;
  $lbound \leftarrow ExpSearch(lbound', I.left)$;
  $ubound \leftarrow ExpSearch(ubound', I.right)$;
  if $D^{(i)}[ubound] = I.right$ **then**
    $ubound \leftarrow ExpSearch2(ubound, I.right)$;
  else
    $ubound \leftarrow ubound - 1$;
  /* assume $lbound < ubound$, otherwise discard I */;
  add to $\mathcal{P}$ all unvisited pages from $\lfloor lbound/\Omega \rfloor$ to $\lfloor ubound/\Omega \rfloor$;
add to $\mathcal{S}$ all objects saved in $\mathcal{P}$ satisfying $dist(p,q) \leq r$;
return $\mathcal{S}$

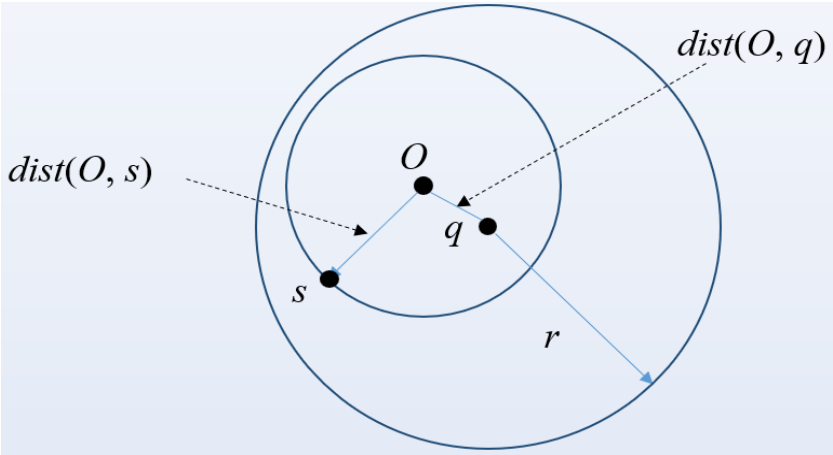


图 2. 包含规则示意图

4 复现细节

4.1 与已有开源代码对比

4.1.1 原文代码约简部分

```
CalCirclePos::CalCirclePos(Point &refPt, double radius_refPt, Point &queryP
{
    this->dim = refPt.coordinate.size();
    double distance = CalDisOfPt(refPt, queryPt);
    if(distance >= radius_refPt + radius_queryPt){
        this->label = 1;
        this->dis_lower = 0x3f3f3f;
        this->dis_upper = 0x3f3f3f;
    }
}
```

4.1.2 本文代码约简部分

```
CalCirclePos::CalCirclePos(Point &refPt, double radius_refPt, Point &queryP
{
    this->dim = refPt.coordinate.size();
    double distance = CalDisOfPt(refPt, queryPt);
    if(distance >= radius_refPt + radius_queryPt){
        this->label = 1;
        this->dis_lower = 0x3f3f3f;
        this->dis_upper = 0x3f3f3f;
    }else if(distance >= fabs(radius_refPt - radius_queryPt)){
        this->label = 2;
```

```
        if(distance > radius_queryPt){
            this->dis_upper = radius_refPt;
            this->dis_lower = distance - radius_queryPt;
        }else{
            this->dis_lower = 0.0;
            this->dis_upper = radius_refPt;
        }
    }else{
        this->label = 3;
        if(distance > radius_queryPt){
            this->dis_lower = distance - radius_queryPt;
            this->dis_upper = distance + radius_queryPt;
        }else{
            this->dis_upper = distance + radius_queryPt;
            this->dis_lower = 0.0;
        }
    }
}
```

4.2 实验环境搭建

本文使用 Windows11 下的 wsl2 子系统，发行版本为 Ubuntu-20.04，CPU 为 Intel(R) Core(TM) i7-9700K 3.60GHz，RAM 大小为 32GB。使用的编程语言为 C++ 和 Python 3.6。所需环境如图 3和图 4所示。

Python 所需环境	版本
Python	3.8.19
pandas	2.0.3
scipy	1.10.1
scikit-learn	1.3.2

图 3. Python 运行环境

C++所需环境	版本
g++	9.4.0
CMake	3.16.3
Makefile	4.2.1

图 4. C++ 运行环境

4.3 复现步骤

(1) 数据集准备. 本文使用的数据类型是图像的色彩直方图特征，即多维数据，使用的数据集为 ImageNet32（https://image-net.org/download-images.php）。在目录下运行命令:python gendata.py，提取图像特征。

(2) 编写 CMake 文件. CMake 文件内容如下所示:

```
# CMake最低版本要求
cmake_minimum_required(VERSION 3.10)
# 项目名称
project(LIMS)
# 设置C++标准为C++17
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
# 添加编译选项
#add_compile_options(-Wall -O3 -ffast-math -march=native)
add_compile_options(-Wall)
# 设置构建类型为Debug
set(CMAKE_BUILD_TYPE "Debug")
# 查找所有的源文件
file(GLOB_RECURSE SOURCES "*.cpp")
file(GLOB_RECURSE REMOVE_CMAKE "${CMAKE_CURRENT_SOURCE_DIR}/build/*")
list(REMOVE_ITEM SOURCES ${REMOVE_CMAKE})
# 添加可执行文件
add_executable(${PROJECT_NAME} ${SOURCES})
```

(3) 编译 C++ 文件. 在 CMakeList 目录下输入以下命令:

```
mkdir build
cd build
cmake..
make clean
make all
```

4.4 创新点

本文为 LIMS 添加了新的约简规则，减少了 LIMS 在搜索过程中的假阳性数据，进而减少了搜索时的计算代价，具体原理参考 3.2 节。

5 实验结果分析

由于本文属于精确查找，因此以搜索时间作为性能指标。图 5为本文改进方法和 LIMS 在不同支撑点数目下，随机使用不同查询对象 query 和查询半径 r 进行 20 次查询的平均用时。

Number of Pivots	time(μs)	
	Our Method	LIMS
1	**39.9**	44.8
2	**28.2**	30.1
3	**30.5**	31.8
4	**35.4**	40.1

图 5. 实验结果

可以看到，本文提出的方法均明显优于 LIMS 原始版本。这是因为包含规则本身没有增加额外的距离计算，仍然使用了在判断是否排除数据时使用到的 $dist(O, q)$。因此，当数据满足包含规则时，则可以在不增加额外距离计算的情况下，被判定为查询对象。

6 总结与展望

本文成功复现了 LIMS 算法，并通过引入新的约简规则对其进行改进。在复现过程中，详细搭建了基于 Windows 11 下 wsl2 子系统的实验环境，使用 Ubuntu - 20.04 系统、C++ 和 Python 编程语言，并对所需的软件版本进行了精确配置。实验结果显示，改进后的方法在搜索时间上明显优于原始的 LIMS 算法，证明了新约简规则的有效性，为度量空间中的相似性搜索提供了更高效的策略。然而，目前的工作仍存在一定的局限性。一方面，实验仅采用了图像的色彩直方图特征数据集，数据类型相对单一，未来可以尝试在更多类型的数据集上进行测试，以验证改进方法的普适性。另一方面，在索引构建和查询性能优化方面，尽管新规则减少了假阳性数据，但仍有进一步提升的空间，例如探索更高效的机器学习模型或数据划分策略，以适应大规模和高维数据的挑战。未来的研究可以从以下方向展开：一是深入研究不同数据分布和查询负载对算法性能的影响，使算法更具鲁棒性；二是结合其他前沿技术，如深度学习方法，探索新的索引构建和查询优化策略，进一步提高度量空间相似性搜索的效率和准确性，为相关领域的应用提供更强大的技术支持。

参考文献

[1] Angjela Davitkova, Evica Milchevski, and Sebastian Michel. The ml-index: A multidimensional, learned index for point, range, and nearest-neighbor queries. In *EDBT*, pages 407–410, 2020.

[2] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. Tsunami: a learned multi-dimensional index for correlated data and skewed workloads. *Proc. VLDB Endow.*, 14(2):74–86, October 2020.

[3] Paolo Ferragina and Giorgio Vinciguerra. The pgm-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proc. VLDB Endow.*, 13(8):1162–1175, April 2020.

[4] Tu Gu, Kaiyu Feng, Gao Cong, Cheng Long, Zheng Wang, and Sheng Wang. The rlr-tree: A reinforcement learning based r-tree for spatial data. *Proc. ACM Manag. Data*, 1(1), May 2023.

[5] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, June 2005.

[6] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 489–504, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. Lisa: A learned index structure for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2119–2133, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Pengfei Li, Hua Lu, Qian Zheng, Long Yang, and Gang Pan. Lisa: A learned index structure for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2119–2133, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Guanli Liu, Jianzhong Qi, Christian S. Jensen, James Bailey, and Lars Kulik. Efficiently learning spatial indices. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 1572–1584, 2023.

[10] Anisa Llaveshi, Utku Sirin, Anastasia Ailamaki, and Robert West. Accelerating b+ tree search by using simple machine learning techniques. In *Proceedings of the 1st International Workshop on Applied AI for Database Systems and Applications*, 2019.

[11] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking learned indexes. 14(1):1–13, September 2020.

[12] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '84, page 181–190, New York, NY, USA, 1984. Association for Computing Machinery.

[13] Caetano Traina, Roberto F. Filho, Agma J. Traina, Marcos R. Vieira, and Christos Faloutsos. The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The VLDB Journal*, 16(4):483–505, October 2007.

[14] Haixin Wang, Xiaoyi Fu, Jianliang Xu, and Hua Lu. Learned index for spatial queries. In *2019 20th IEEE International Conference on Mobile Data Management (MDM)*, pages 569–574, 2019.

[15] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. Database meets deep learning: Challenges and opportunities. *SIGMOD Rec.*, 45(2):17–22, September 2016.

[16] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[17] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Åke Larson, Donald Kossmann, and Rajeev Acharya. Qd-tree: Learning data layouts for big data analytics. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 193–208, New York, NY, USA, 2020. Association for Computing Machinery.