# LMSFC: 一种基于学习单调空间填充曲线的新型多维索引（扩展版）

**摘要：**最近提出的学习型索引引起了广泛关注，因为它们可以适应实际数据和查询分布以获得更好的搜索效率。基于这项技术，一些现有的工作为多维数据建立了索引，提升了查询性能。这些工作的常见范例是（i）使用固定空间填充曲线（SFC）或其变体将多维数据点映射到一维空间，以及（ii）然后应用已有的索引技术。我们注意到（这些工作）第一步通常使用固定的 SFC 方法，例如行主序和𝑧-order。这无疑限制了学习多维索引通过不同查询工作负载适应可变数据分布的潜力。

在本文中，我们提出了一种学习空间填充曲线的新颖想法，该曲线经过精心设计和积极优化以实现高效的查询处理。我们还确定基于 SFC 的学习索引常见的创新离线和在线优化机会，并提供最佳和/或启发式解决方案。 实验结果表明，我们提出的方法 LMSFC 在三个常用的现实数据集和不同的实验设置中优于最先进的非学习或学习方法。

1. 介绍

如今，多维数据量大、种类繁多。 例如，在传统的数据仓库和分析数据库中，大部分关键数据存储在多维事实表中。基于位置的服务和传感器（例如谷歌地图）的广泛部署会产生大量的多维数据。 数据通常是两个或三个空间维度，以及用于各种测量的一维或多个维度。 对这些多维数据集的常见且占主导地位的查询类型是窗口查询，它对多个或所有维度施加范围约束。

多维索引对于有效回答大量多维数据集的窗口查询至关重要。 先前的研究提出了许多传统索引，包括𝑅树[12]、kd树[3]和四叉树[9]。 它们都是基于空间分区，主要区别在于分区之间是否存在重叠。

空间填充曲线是多维索引中最常用的方法之一[16, 30]。 这是因为 SFC 具有**出色的邻近保持特性**，使其成为线性化数据对象的理想选择。 SFC 可分为两类：单调 SFC 和非单调 SFC。 单调 SFC，例如 𝑧-order[25]，可以快速定位搜索范围。 然而，非单调 SFC 可能会导致查询处理期间产生更多计算开销。例如，希尔伯特曲线[14]需要枚举查询窗口边界上的所有值来确定搜索范围。 因此，当使用非单调 SFC 作为线性化方法时，执行范围搜索会更加困难。

最近，在开创性工作 [19] 的推动下，通过机器学习优化数据库索引的浪潮激增。 它利用针对特定数据和查询工作负载实例进行优化的独特优势。 因此，一些工作研究了学习的多维索引。 普遍的方法是将多维数据点映射到一维，并进一步在一维空间上应用学习的索引。

然而，它们有以下的限制：(1) 多维索引独特和关键的部分就是从多维空间映射到一维空间，而这尚未被学习或很好地学习。大多数方法都利用了现有的SFC,例如z-order,因为它具有良好的邻近保持能力。然而，固定的 SFC 不一定在给定的数据集实例上发挥最佳作用。其他的工作，例如Flood,只学习了选择一个特殊的维度，然后在其余的 𝑑 −1 维度上遵循固定的行优先顺序，因此错过了更好地保留局部邻近性的机会。(2) 存储为磁盘页面的线性化数据点的物理布局未完全优化。现有方法通常将固定数量的数据点打包到每个页面中，这可能会导致结果页面的最小边界矩形（MBR）包含大量死空间或彼此严重重叠[34]。(3)即使上述两个问题可以在索引构建时得到缓解，现有的查询处理方法仍然需要访问许多页面，因为它们的MBR或𝑧地址范围不可避免地与查询的范围重叠。

在本文中，我们对学习的多维索引进行了彻底而严格的研究，并提出了 LMSFC 来解决上述局限性。(i) 制定一个合适的参数化 SFC 族是一项具有挑战性的任务，这些 SFC 可以有效地学习并具有有效查询处理的显着属性。为此，我们设计了一个可学习的单调 SFC 族。 基于所提出的 SFC 族，我们设计了一种基于 SMBO 的有效解决方案，以学习最优/次优 SFC 以适应不同的数据分布和工作负载。（ii）我们研究以系统性的方式**将多维数据点打包到大小有限的磁盘页面中的物理存储优化问题。** 由于（学习的）SFC 的线性化，我们能够通过动态规划最优地解决其他 NP 难题。我们进一步提出了一种启发式算法，以最优性换取提高的实际速度，适用于大规模数据集。(iii)除了上述离线优化技术之外，我们通过提出**查询拆分策略**进一步利用独特的在线查询优化机会。 我们证明了我们学习到的 SFC 有助于将查询分成两部分的有效算法，从而最大限度地减少对假阴性数据页的总访问； 然后将该算法扩展为允许通过递归进行多次分割。

该论文的贡献总结如下：

1. 据我们所知，LMSFC 是第一个考虑学习空间填充曲线直接针对给定数据集实例和查询工作负载的最低成本查询处理进行优化的工作。
2. 基于学习到的 SFC，我们提出了离线和在线优化技术。 对于离线优化，我们可以最优或次优地将多维数据点打包到能够使基于密度的成本函数最小化的页面中。 对于在线优化，我们建议将查询递归地拆分为子查询，以便最大限度地减少对不包含任何潜在查询数据点的页面的访问。
3. 我们在实验评估中将 LMSFC 与之前最先进的多维指标进行比较。 LMSFC 在不同的查询选择性、数据大小和查询纵横比下在三个现实数据集上实现了最佳查询性能。 LMSFC 相对于 𝑅\*-tree、ZM-index 和 Flood 分别可以实现高达 38.2×、7.2× 和 2.0× 的加速。

本文的其余部分安排如下。 在第 2 节中，我们定义问题设置并介绍一些关键符号和概念。 然后我们在第 3 节回顾相关文献。第 4 节首先研究基于 SFC 的索引的范围查询处理的要求，并激发了参数化 SFC 系列。 然后，我们概述了基于参数化 SFC 系列的 LMSFC 索引，并在第 5 节中介绍了其构建的关键步骤。我们在第 6 节中用新颖的查询拆分策略激发并介绍了我们的查询处理方法。第 7 节介绍了我们的实验结果和分析 。 最后，我们总结本文并在第 8 节中讨论一些扩展

**2.准备工作**

**2.1 问题定义**

在本文中，我们主要关注多维数据集上窗口查询的精确查询处理。

定义一（多维数据集）多维数据集 𝐷 由 𝑑 维欧几里得空间中的 𝑛 点组成。每个点 𝑥 ∈ 𝐷 都可以表示为 (𝑥 (1) , · · · , 𝑥 (d) ) 其中

𝑥 (𝑖 ) ∈ R𝑑 是 𝑥 的第 𝑖 维值。

为了不失一般性，我们假设，通过适当的缩放，每个坐标的域是[0, 2𝐾 -1]内的整数，因此每个维度值都可以使用𝐾二进制位来表示。

一个多维窗口就是一个d维空间的超矩形，或者正式定义为w=[xL(1), xu(1)]\*…\*[xL(d), xu(d)],其中xL(i)< xu(i)

定义二(多维窗口查询). 给定多维数据集 𝐷，具有窗口约束 𝑞.𝑤 的多维窗口查询 𝑞 返回位于窗口 𝑞.𝑤 内部的 𝐷 的点集 𝑥，也就是 R(q)={ 𝑥 | 𝑥 ∈ 𝐷 ∧

∀1 ≤ 𝑖 ≤ 𝑑, 𝑞.𝑤 𝐿(𝑖 )≤ 𝑥 (𝑖 ) ≤ 𝑞.𝑤 u(𝑖 )}

由于查询的独特特征在于其查询窗口，因此我们将在下文中互换使用 𝑞 和 𝑞.𝑤。

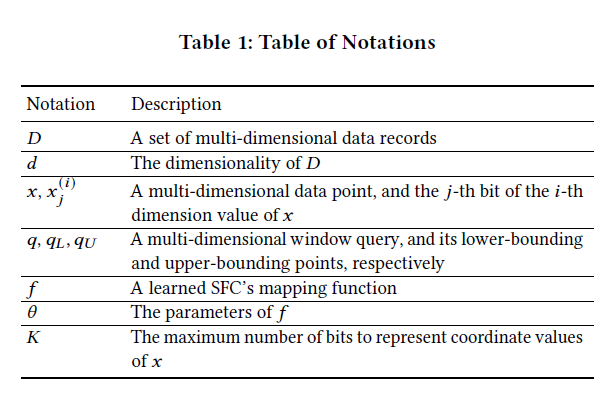
2.2 符号

给定一个多维窗口w,它被(wL,wU)唯一定义,也就是，下限点和上限点分别被定义为wL=( xL(1), …, xL(d)), wU=（xu(1)，…, xu(d)）。这些也适用于𝑞查询窗口的窗口约束，我们将使用简写𝑞𝐿来表示𝑞.𝑤𝐿。

给定一组点，我们将多维最小边界矩形（MBR）定义为包围该组点的最小窗口。

对于二进制整数 𝑣，我们将 𝑣 的二进制表示形式的第 𝑗 位表示为 𝑣 𝑗 。 请注意，𝑗 从 0 开始，对应于 𝑣 的最右位。 𝑣 的最高有效位是设置为 1 且具有最大位索引的位。 例如，令 𝑣 = (00101101)2 （我们用 ()2 表示二进制字符串），𝑣2 = 1，𝑣 的最高位为 5。

表 1 列出了常用的符号。



3.相关工作

索引对于处理大型数据集的查询至关重要。 传统索引针对最坏情况的性能进行了优化。 更重要的是，他们错过了利用有关数据和查询工作负载的统计信息来优化索引结构和物理布局的机会。 最近，已经提出了许多基于机器学习的索引，它们实现了更小的索引大小和/或更快的查询处理速度。RMI [19] 是一项众所周知的工作，它首先注意到一维非降序值数组的精确搜索与经典回归问题之间的相似性。然后，它提出了学习索引的几个实例，这些实例使用复杂的模型来预测目标键的逻辑位置，然后执行本地搜索来修复机器学习模型可能产生的错误。 后续工作进一步提高模型的性能或考虑其他变体。例如，PGM [8] 让用户先验指定误差范围，然后使用更简单的线性回归模型和最佳分割算法来构建学习索引。 Fitting-tree [10] 还使用可调误差参数来权衡索引大小和查找性能。 给定数据集，Fitting-tree 应用成本模型来估计空间消耗和延迟，以找到合适的误差范围。 Radix Spline [18] 考虑使用线性样条来逼近 CDF。 所选样条点的前缀存储在辅助基数表中以加速搜索过程。ALEX [5]和LIPP [40]支持数据更新。 LIPP 通过一种新颖的调整策略来重新分配每个节点中的密钥，进一步消除了本地搜索。 SOSD [17, 23] 提出了一些基准来评估不同的学习指标。

现有的学习索引技术不能直接应用于多维数据，因为多维数据点之间不存在自然排序。 目前，流行的方法是应用线性化方法将问题转换为一维搜索问题，然后可以应用现有的学习索引。 𝑧-order [29, 38] 和row-major order[22, 26] 使用最广泛。其他线性化方法使用学习的线性或非线性映射[21]，例如，聚类后跟到聚类中心的距离[4]。 为了减少查询倾斜和数据相关性的挑战，Tsunami [6] **通过对数据空间分区和建模条件 CDF** 来扩展 Flood。Qd-tree [41]利用强化学习来优化空间划分。SPRIG [42]使用空间插值函数来定位网格中的搜索范围。 [31]学习四叉树结构并在每个节点上应用𝑧阶变体。 [28]将𝑧阶的位交织方法扩展到更一般的情况，并提出了学习SFC的想法。 在他们的工作中，他们采用启发式方法来识别对给定查询模式既具有查询感知能力又具有倾斜容忍能力的 SFC。与他们的方法不同，我们利用 ML 方法在给定的查询工作负载下直接优化查询性能，以找到估计查询时间最少的 SFC。

与大多数基于空间填充曲线的方法类似，学习的多维索引的正交方面是使用简单的线性变换或复杂的非线性变换（例如等级空间变换）来预处理数据. 传统的多维索引通常将空间递归地分解为不相交或重叠的分区。 网格文件[27]、kd-tree[3]和Quadtree[9]是前一类的典型例子，𝑅-tree[12]及其变体[2,16,33]是后一类的典型例子。由于𝑅-tree [12]变体已在商业系统中广泛采用，因此也有人建议将学习集成到𝑅-tree中。 AI+R [1] 采用 ML 模型来预测窗口查询的叶节点集以及备份 𝑅 树。 [11]旨在学习𝑅树构造中的关键子树分裂例程并适应问题实例。 为了减少每个叶节点上的搜索范围，[13]在选定的排序维度上嵌入ML模型以加速搜索过程。

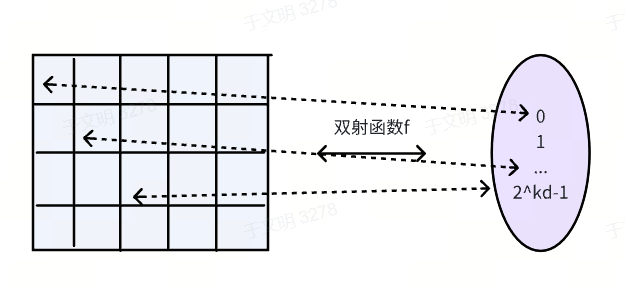
还有其他方法可以将学习集成到数据库索引中。 [7] 使用机器学习模型来学习平衡的空间划分，以很好地保留空间邻近性。 LIMS [36]采用基于机器学习的数据聚类方法来解决度量空间中的相似性搜索。

**4 学习型 SFCS 框架**

在本节中，我们首先总结 SFC 的窗口查询处理问题，以激励我们设计一系列具有查询处理显着属性的参数化 SFC。

**4.1 使用 SFC 进行窗口查询处理**

空间填充曲线（SFC）是将多维数据空间映射到一维数据空间的方法。 由于我们假设数据点具有 [0, 2𝐾 − 1] 范围内的整数坐标值，这自然会导致多维空间划分为 2𝐾𝑑（d个2K相乘） 可能的点或单元（SFC 也可以应用于更粗的粒度。例如，Flood[26]可以被视为在网格上使用固定的SFC）。 SFC 是这些单元格和 [0, 2𝐾𝑑 −1] 内的整数之间的双射函数 𝑓。



我们将多维空间中的 𝑥 称为原始地址，并将 𝑓 (𝑥) 称为其相对于 SFC 𝑓 的相应 𝑧 地址 。直观上，由于 𝑓 (𝑥) 是一个一维整数，它指定了一种恰好遍历所有单元格一次的方法。 SFC 以其保持线性顺序的多维邻近性的能力而闻名[20]。 因此，它们被广泛使用，特别是在需要线性化多维数据（例如图像、表格和空间数据）的应用中。 一些著名的 SFC 包括希尔伯特曲线、Z 阶曲线和格雷曲线。

为了回答𝐷上的范围查询𝑞，假设𝐷中的点已映射到相应的𝑧地址，我们可以（i）计算查询的一维𝑧地址范围𝑞𝑧，（ii）检索其𝑧-的每个点 地址落在区间 𝑞𝑧 内，并且（iii）过滤掉那些不属于查询窗口 𝑞 的点。最严格的𝑧地址范围可以定义为：[min𝑥∈𝑞 𝑓 (𝑥), max𝑥∈𝑞 𝑓 (𝑥)]。 然而，一般来说，计算这些极值是很困难的。 在最坏的情况下，我们可能需要枚举 𝑞 中的所有 𝑥，因此成本与查询量成正比，并且无法达到高效查询处理的目的。对于某些具有更好性能的 SFC，例如希尔伯特曲线，我们仍然需要枚举查询窗口边界上的所有 𝑥，从而导致与查询周长成正比的成本。

尽管如此，我们还是确定了 SFC 的一个子类，以便可以在 𝑂(𝑑) 时间内有效地计算上述最小化和最大化，因此不依赖于查询的大小。

4.2单调空间填充曲线

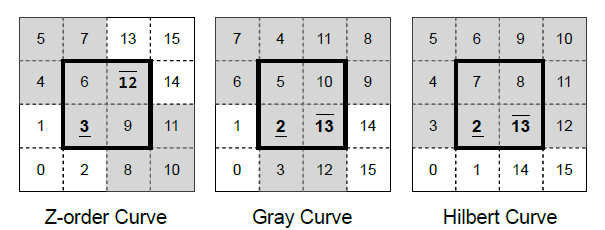
我们定义 SFC 的映射函数的性质必须是单调的，并证明可以从查询窗口的下边界点和上边界点计算最紧的 𝑧 地址范围。

定义 3（多维空间中的单调函数）。 令 𝑎 ⪯ 𝑏 定义为 true 当且仅当 ∀𝑖, 𝑎(𝑖 ) ≤ 𝑏 (𝑖 ) 。 那么函数 𝑔 是单调的，如果对于所有 𝑎 和 𝑏，如果 𝑎 ⪯ 𝑏，则 𝑔(𝑎) ≤ 𝑔(𝑏)。

定理 1. 如果一个 SFC 对应一个单调映射函数 𝑓 ，给定一个空间查询矩形 𝑞 ，则查询结果 𝑟 ⊆ { 𝑥 | 𝑥 ∈ 𝐷 ∧ 𝑓 (𝑞𝐿) ≤ 𝑓 (𝑥) ≤ 𝑓 (𝑞𝑈) }。 换句话说，𝑞的最紧𝑧地址范围可以有效地计算为[𝑓 (𝑞𝐿), 𝑓 (𝑞𝑈 )]。

证明 令qmin=min{ 𝑥 ∈ 𝑞}对于 ⪯ （即 𝑞min是𝑞𝐿）。然后根据任何 𝑥 ∈ 𝑞 的定义，𝑞min ⪯ 𝑥。 由于 𝑓 是单调的，则 𝑓 (𝑞min) ≤ 𝑓 (𝑥)。 类似地，我们可以证明 𝑞max def= max{ 𝑥 ε 𝑞 } （即 𝑞max 是 𝑞𝑈 ）并且对于任何 𝑥∈𝑞，𝑓 (𝑥) ≤ 𝑓 (𝑞max)。

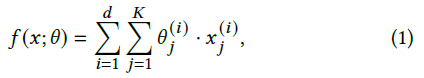
1. 三种常用的SFC中，只有𝑧阶曲线具有单调性。 我们给出了图1中希尔伯特曲线和格雷曲线的反例。



**4.3 参数化 Z 顺序 SFC**

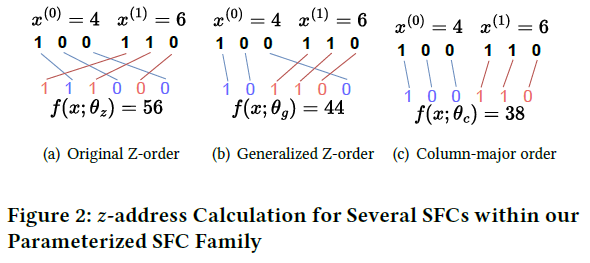
受 𝑧 阶曲线的单调特性的启发，我们确定了一系列单调 SFC来推广 𝑧 阶曲线。 此外，此外，该族中的任何实例都具有支持高效查询处理的有益特征。

我们考虑以下由参数 参数化的 SFC 𝜃 = [𝜃 (1) , . . . , 𝜃 (𝑑) ]:

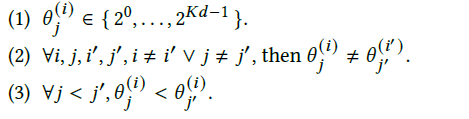


其中每个 𝜃 (𝑖 ) 是一个 𝐾 维向量，𝑥 𝑗 (𝑖 ) 表示第 𝑗 维向量𝑥 的第 𝑖 维度值的bit位，𝑑 是维度，𝐾 是𝑥 (𝑖 ) 的最大位数。 事实上，𝜃 𝑗 (𝑖 ) = 2𝑙 表示𝑥 𝑗 (𝑖 )将被映射到 𝑓 (𝑥; 𝜃) 的二进制表示的第 (𝑙 +1) 位。 如果内容清楚的话，我们简称为𝑓（𝑥），而不是𝑓（𝑥；𝜃）

**示例 2**。图 2 演示了我们参数化系列中 SFC 的几个实例。 图 2(a) 显示了计算 2 维数据点 𝑥 = (4, 6) 的 𝑓 (𝑥) 的 𝑧 阶有序位交织方式。 这里，𝐾 = 3，其参数𝜃𝑧 = [𝜃 (1) , 𝜃 (2) ] = [ [1, 4, 16], [2, 8, 32]]，得到的𝑧地址为56。 图 2(b) 演示了同一数据点的新 SFC，但 𝜃𝑔 = [ [1, 16, 32], [2, 4, 8]], 𝑓 (𝑥; 𝜃𝑔) = 44。最后，图 2 (c) 演示了另一个 SFC，称为列主序，其中 𝜃𝑐 = [ [8, 16, 32], [1, 2, 4]], 𝑓 (𝑥; 𝜃𝑐 ) = 38。



为了确保 SFC 系列保留单调和双射属性，只需对𝜃 𝑗 (𝑖 ) 施加以下约束：



证明。 根据前两个约束，𝑓 (𝑥) 可以表示为二进制形式，它实际上是多维数据点 𝑥 的所有维度的位的组合。 因此，𝑓 (𝑥) 可以由给定的 𝑥 唯一确定，反之亦然。 所以我们可以得出结论 𝑓 是双射的。接下来，我们证明f(x)是单调的。假设我们有两个不同的多维数据点𝑝和𝑞，得到的映射地址分别是𝑓（𝑝）和𝑓（𝑞）。当p ⪯q时，根据定义3，在任意维度i上我们有p(i) ⪯ q(i).如果在维度i上p(i)< q(i), 我们用 𝑙 表示最左边的位，其中 𝑝 (𝑖 ) 和 𝑞(𝑖 ) 不同，因此我们有

𝑝𝑙(𝑖 ) = 0 和q𝑙(𝑖 ) = 1。第三个约束可以保证𝑓(𝑥)中数据点𝑥的每个维度的位序保持不变。 因此，在 𝑓 (𝑝) 和 𝑓 (𝑞) 的二进制形式中，最左边的不同位也分别为 0 和 1，我们可以得出 𝑓 (𝑝) ≤ 𝑓 (𝑞) 的结论，这表明 函数 𝑓 是单调的。

双射和单调属性都是我们工作中重要的有意识的选择，以平衡（1）通过学习 SFC 来利用空间局部性的潜力和（2）保留促进高效查询处理的属性。违反任何约束都会破坏映射函数的属性。 例如，考虑 𝜃 = [ [1, 4], [2, 10]]，其中违反了第一个约束。取f(𝑥) = 8。没有点可以映射到这个z地址，表明映射函数是单射的，但不是双射的。 再例如，当 𝜃 = [ [1, 4], [2, 2]] 时，不满足第二个约束。 考虑数据点 𝐴 = (1, 1) 和 𝐵 = (1, 2)，则 𝑓 (𝐴) = 𝑓 (𝐵) = 3。因此，映射函数不是双射的。 最后，当 𝜃 = [ [1, 4], [8, 2]] 时，违反了第三个约束。 给定 𝐴 = (1, 1) 和 𝐵 = (2, 2)，则 𝑓 (𝐴) = 9 > 𝑓 (𝐵) = 6，但根据定义 3 𝐴 ⪯ 𝐵。因此，映射函数不是单调的。

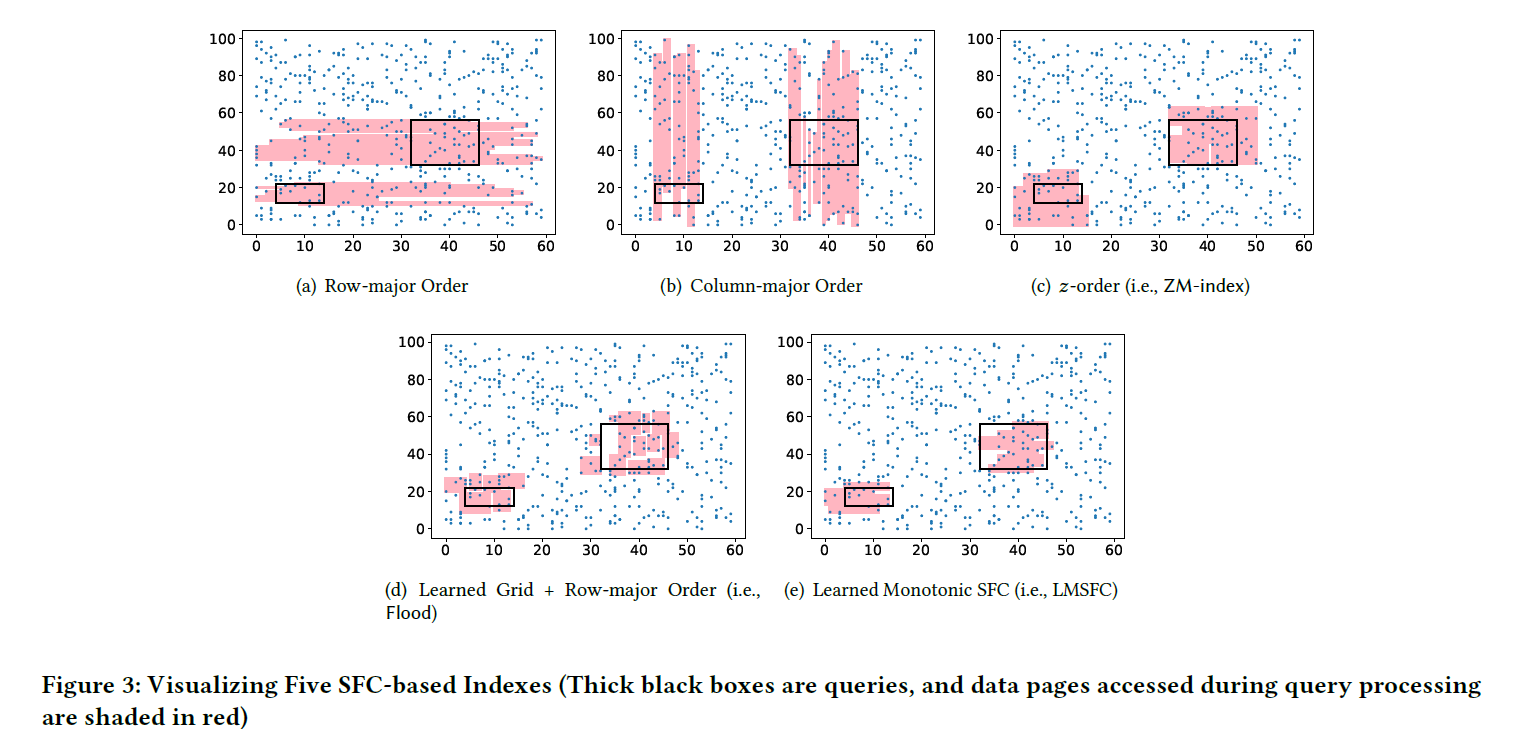
我们注意到，Z 阶曲线因此是上述单调 SFC 的一个特殊实例。具体来说，它对应于𝜃 𝑗(𝑖) =2(𝑗−1).𝑑+(𝑖−1) 。我们还注意到，计算上述 𝑓 (·) 是高效的，因为 𝑓 (𝑥) 可以通过有效地使用位运算根据 𝜃 对 𝑥 的位进行“加扰”来计算。

由于不同的 SFC 会导致数据集中数据点的不同线性排序，因此对于给定的工作负载，它们将导致不同的查询处理成本。 这激励我们学习一个好的 SFC（将在下一节详细介绍）。 下面我们提供了一个带有可视化效果的示例来演示这一点。

*示例 3*。在图 3 中，我们基于相同数据集和查询工作负载的不同线性化方法，在五个索引上构建并可视化查询处理成本。 我们在 2D 空间中生成一组随机点，并使用两个随机生成的查询（表示为粗黑框）作为查询工作负载。

图 3(a) 和图 3(b) 分别是通常的行优先顺序和列优先顺序。 图 3(c) 使用 𝑧 顺序，从而产生 ZM 索引。 图 3(d) 是 Flood，它基于可学习的空间网格划分，然后使用行主顺序（具有重新排列的维度顺序）对网格进行排序。 最后，图 3(e) 显示了我们提出的方法 LMSFC，该方法基于学习的单调 SFC。

我们将查询处理期间的数据点访问用红色表示。1 正如我们从图 3 中看到的，由于对特定实例的学习，学习索引（即 Flood 和我们的 LMSFC）访问的数据点更少。

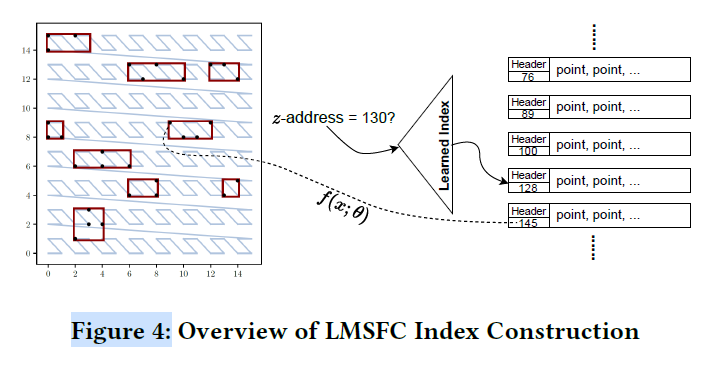


**5 LMSFC 索引构建**

**5.1 概述**

基于上一节中定义的参数化 SFC 系列，我们提出了一种基于学习单调空间填充曲线的新型多维索引 LMSFC。

我们在图 4 中给出了 LMSFC 方法的高级草图。在我们的方法中，我们首先学习一个良好的单调 SFC，图 4 中的 𝑓 (𝑥; 𝜃)，它最大限度地减少了采样查询工作负载的查询处理成本。 学习到的 SFC 为我们提供了多维数据点的总顺序。然后，我们提出了一种基于最优动态规划的算法和一种次优但更快的启发式分页算法，以将数据点加载到页面中（图 4 中的粗红色矩形）。最后，我们从每个页面中提取最小的𝑧地址以形成一个排序数组，并在其上使用最先进的学习索引（例如，pgm [8]），这有助于从𝑧地址到页。

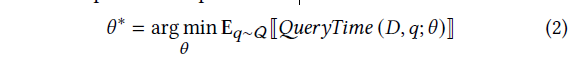


此外，我们还进行了多项优化以加快查询处理速度。 具体来说，我们提出了一种新颖的查询拆分策略，以最大限度地减少由于 SFC 映射的降维效果而对虚假页面的访问。 我们还将排序维度优化[26]扩展到页面级粒度。

在本节的其余部分中，我们将重点关注本节中的三个步骤（即学习最佳 SFC、基于成本的分页和页级排序维度），并将查询处理相关技术留给下一节。

**5.2 学习最佳 SFC**

学习参数化 SFC 的目标是找到 𝜃\*，从而最小化生成的查询处理成本，即，我们将其表述为优化问题：

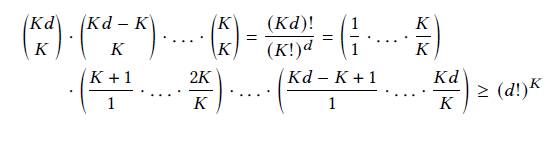


其中 Q 表示查询工作负载的分布，𝑞 是来自分布的独立同分布样本𝑄𝑢𝑒𝑟𝑦𝑇𝑖𝑚𝑒() 函数返回给定数据集上的实际查询执行时间。

为了优化方程（2），我们可以首先应用有限样本近似，即从 Q 中获取采样查询并用样本平均值替换期望。然而，𝑄𝑢𝑒𝑟𝑦𝑇𝑖𝑚𝑒函数很难准确建模或近似，因为它包含各种复杂的优化。 此外，直接评估 𝑄𝑢𝑒𝑟𝑦𝑇𝑖𝑚𝑒 函数会产生高昂的成本。另外，请注意𝜃是一个高维离散参数（有约束），𝜃（因此𝑓（·））的选择数量在𝑑和𝐾维度上都是指数级的，导致无法通过在实践中通过蛮力解决优化问题。

引理1 𝑑 维空间的不同单调 SFC 的数量为 Ω(𝑑!)𝐾。

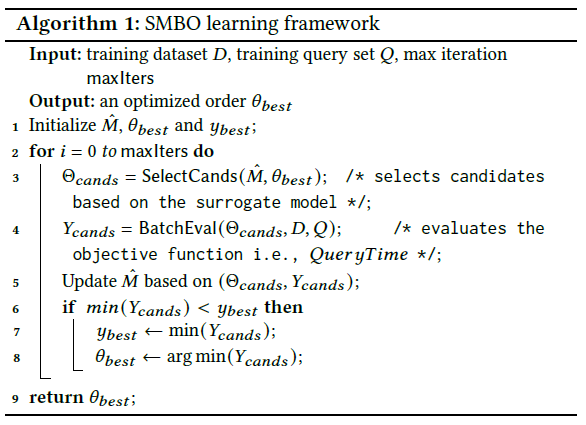
证明。 首先，我们考虑 𝑑 维度的排列 𝜋 = (𝑥1, 𝑥2, ..., 𝑥𝑑 )，其中每个维度有 𝐾 位。 对于维度 𝑥𝑗 ，其𝐾 位可以选择任何剩余的 𝐾𝑑 − ( 𝑗 − 1) · 𝐾 位。 因此，总的选择数为：



由于离散性，之前学习索引工作[26]中使用的梯度下降式算法也不能应用于解决这个优化问题。 此外，有限样本近似还为优化引入了微小但可避免的噪声。

鉴于上述挑战，我们采用最先进的贝叶斯优化算法SMBO [15]来近似解决上述优化问题，并返回平均查询时间较低的高质量SFC。 SMBO建立代理模型来近似参数与实际目标函数值之间的关系。学习过程遵循标准SMBO学习框架。 我们使用随机森林模型作为代理模型，而不是典型的高斯过程模型，提高了学习速度，并且不依赖于多维高斯假设和核函数的选择。 使用代理模型的优点之一是评估成本低廉，同时还能捕获目标函数中的近似噪声。

算法1说明了SMBO学习框架，其中ˆ𝑀是我们需要学习的代理模型。 首先，根据 𝑄𝑢𝑒𝑟𝑦𝑇𝑖𝑚𝑒 评估结果在初始候选人和他们的表现之间建立代理模型（第 1 行）。 其次，SMBO 算法使用在代理模型上计算的获取函数（例如预期改进[15]），通过自动平衡利用和探索（第 3 行）来建议下一次迭代中评估的其他候选者。然后，我们通过在采样数据集上构建索引来评估目标函数，以节省评估时间（第 4 行）。 默认情况下，我们保守地使用 5% 采样数据集作为训练数据集，这样可以保持较低的查询时间和学习成本。 更多学习过程实验在第 7.9 节中介绍。 代理模型在每次迭代期间都会更新（第 5 行）。 最后，我们选择目标函数成本最小的候选者。



**5.3 基于成本的分页**

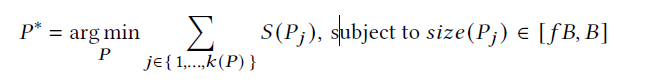
为了适应外部 I/O 并允许额外的优化2，我们需要执行分页，将数据集𝐷划分为多个页面。 像往常一样，我们假设每个页面的最大大小为 𝐵 字节，并且必须满足由 𝑓 ∈ (0, 1]. 3 指定的最小填充因子约束。也就是说，每个页面使用的字节数必须在范围内 [𝑓𝐵, 𝐵] 字节。

为多维数据集找到最佳分页是NP hard [41]，因此现有的多维索引方法通常基于一些启发式进行分页，例如𝑅树及其变体使用启发式来最小化边距、死空间和结果页面的 MBRs 的重叠区域 [2, 33]。毫不奇怪，这种做法被基于SFC的多维索引所继承。 例如，RSMI [29] 只是将最大数量的点加载到每个页面中，我们将其称为固定大小分页。

我们观察到分页很重要，实际上对于基于 SFC 的多维索引可以得到最佳解决。 这是因为我们可以记录每个页面内数据点的MBR，并使用MBR来进一步优化查询处理。一方面，如果页面的 MBR 与查询不相交，则可以跳过该页面； 另一方面，如果查询中包含页面的 MBR，我们可以按顺序处理页面上的数据点，而无需其他过滤开销。 在这两种情况下，如果页面的 MBR 较小，则更有可能进行此类优化。 默认的一维分页方法（例如固定大小分页方法）不知道页面的 MBR，无法对其进行主动优化。

基于以上观察，我们设计了一个评分函数𝑆（𝑃），直观上就是页面𝑃的密度，S(P)=vol(P)/size(P) 其中 𝑣𝑜𝑙 (𝑃) 和 𝑠𝑖𝑧𝑒 (𝑃) 分别给出页面 𝑃 的 MBR 容量和页面 𝑃 中的数据点数量。

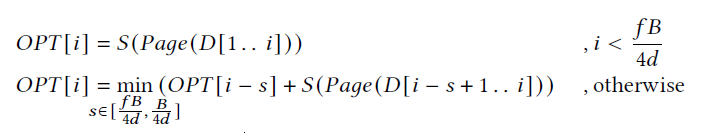
然后，我们将基于成本的最优分页问题制定为寻找分页解决方案，即分区P={ 𝑃1, . . . , 𝑃𝑘 (𝑃 ) }超过 𝐷（其中 𝑘 (𝑃) 表示结果页面的数量），使得解决方案 𝑃 的总分最小化，即



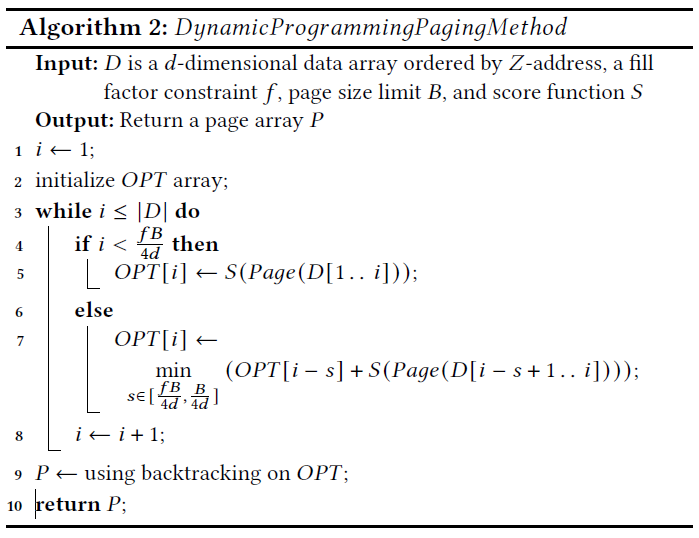
下面，我们首先给出一种基于动态规划（DP）的最优解决上述问题的算法，然后给出一种次优但快速的启发式分页算法。 这两种方法都比固定大小的分页实现了更好的分页布局，从而提高了查询性能。

5.3.1 *动态规划分页方法*。由于SFC为数据点提供了线性顺序，我们能够通过动态规划最优地解决一维分页问题来规避多维分页问题的NP困难。

设 𝑂𝑃𝑇 [𝑖] 是通过基于最优成本的分页算法对第一个 𝑖 数据点获得的最优成本。然后我们可以推导出以下递归方程：



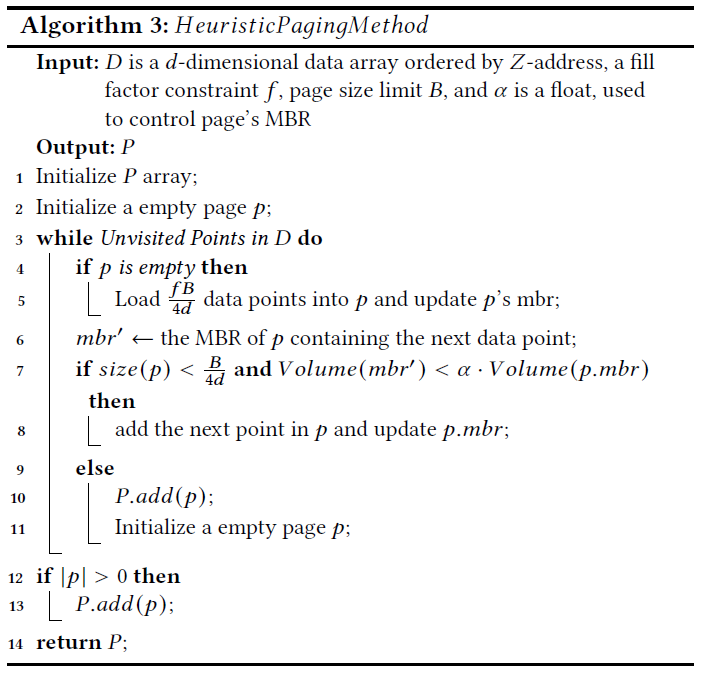
其中𝑃𝑎𝑔𝑒 (𝑧) 表示由一组点形成的页面，表示为𝑧，我们假设每个整数占用 4 个字节。 显然，𝑂𝑃𝑇[𝑛]给出了整个数据集最优分页的成本，并且很容易使用回溯来报告最优分页解决方案𝑃\*。 动态规划分页方法的时间复杂度为𝑂(𝑛𝐵/4𝑑)，因为评分函数𝑆可以通过增量计算在𝑂(1)时间内计算出来。 上述算法在算法2中实现。



*5.3.2 启发式分页方法*

尽管 DP 算法在 𝑛 中是线性的，但在实践中仍然很耗时，因为 𝐵 通常是一个很大的常数（例如，在我们的实验中，𝐵 = 8192）。因此，我们进一步提出了一个启发式的分页方法。与DP方法相比，可以获得可观的查询性能和更快的构建时间。

启发式算法是一种贪婪打包算法，它将尽可能多的数据点打包到当前页面中，直到违反某些条件。我们在算法3中给出了启发式分页方法的伪代码。条件规定，通过将当前数据点添加到页面中而形成的新MBR（第6行）不应将旧MBR放大超过𝛼倍（𝛼） > 1 是一个超参数）（第 7 行）。 这种情况减少了结果页的 MBR 变得太大（相对于其中的数据点数量）的可能性，其中较大的 MBR 可能会导致大量死空间并增加与查询相交的机会。



**5.4 页面级排序维度**

遵循 Flood [26]，我们维护每个页面中的点，这些点按选定的维度（称为排序维度）排序。 与 Flood 中所有页面的排序维度都是固定的不同，我们允许在不同的页面中使用不同的排序维度，这在处理相交页面时提供了更多的跳过机会来过滤尽可能多的不相关点。这是因为不同的排序维度可能会导致细化后的搜索区域大小不同。 因此，我们可以选择能够实现每页搜索成本最小的排序维度。 类似地，我们利用查询工作负载信息来选择每个页面的排序维度，如下所示：对于每个页面，我们收集查询工作负载中的相交查询集。 我们使用每个𝑑维度作为排序维度来估计查询成本，并选择查询成本最小的一个。 如果页面没有相交查询，我们使用默认顺序，其确定方式与 Flood 相同。

一旦我们确定了每个页面的排序维度，我们就通过排序维度的递增顺序对页面中的数据点进行排序。 当我们需要扫描某个页面时，我们可以先根据对应的排序维度细化搜索范围（或物理存储范围）。 具体来说，给定一个窗口查询𝑞，排序维度的范围约束 每页中的点按相应排序维度的升序连续存储。 因此，我们可以使用二分搜索或一维索引模型通过查找在物理存储范围内的下界位置**𝑞𝐿 (𝑑∗)**和上界位置**𝑞U(𝑑∗)**来加速搜索。结果，不满足排序维度范围约束的点被过滤掉，从而减少了扫描开销。

此外，排序维度还可以减少验证时的计算成本。 一旦确定了搜索范围，我们就可以保证该范围内的点满足查询对排序维度的约束。 因此，不需要验证排序维度上的值，从而节省了计算开销。

6.查询过程

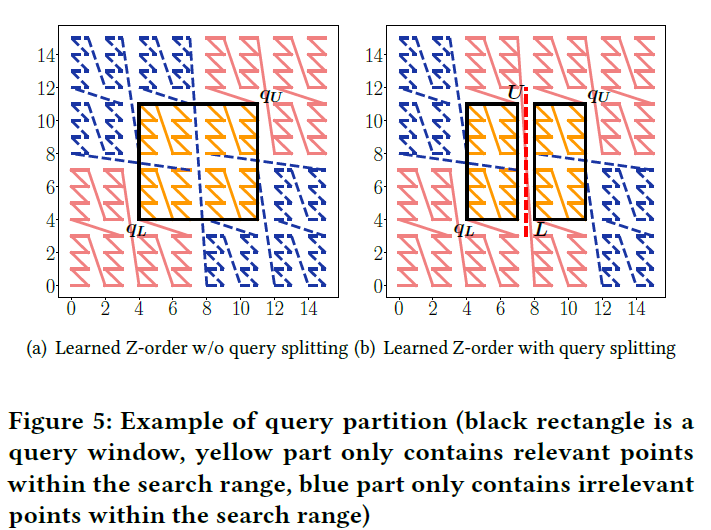
正如第 4 节中提到的，要使用基于 SFC 的索引回答查询 𝑞，需要两个步骤：

（1）投影：给定一个空间查询矩形𝑞，根据定理1，我们确定其在𝑧地址中的扫描范围为[𝑓（𝑞𝐿），𝑓（𝑞𝑈）]。

(2) 过滤扫描：所有𝑧地址在该范围内的页面都需要扫描。 我们可以使用正向索引来定位这些页面。 此外，当我们维护每个页面的 MBR 和特定于页面的排序维度信息时，我们可以执行常规优化，例如跳过不相关的页面，并仅扫描页面的相关部分。

上述框架的主要限制是，由于 SFC 映射，它忽略了𝑧地址范围内潜在的大量误报数据点。

示例4. 考虑图5(a)中的查询（黑色矩形）。相应的𝑧-地址范围可以分解为蓝色部分（误报）和黄色部分。扫描整个范围，即使进行过滤，也会在访问和过滤仅包含误报点的页面时产生许多不必要的开销。



这一现象也在一些方法中观察到，比如UB-Tree [30]，并采用了一种懒惰的跳过策略。在这种策略中，不是扫描𝑧-地址范围中的所有页面，而是在扫描当前页面后调用一个跳过函数FindNextZaddress，以计算下一个包含相对于𝑞的第一个真正例点的页面。尽管此策略保证跳过所有误报页面，但它具有以下缺点：(I) 它产生显着的开销，因为必须为每个真正例页面调用此函数。FindNextZaddress在技术上将计算当前页面中最后一个点之后的下一个𝑧-地址（可能是虚拟的且不对应任何数据点），然后将其转换为一个页面。这将需要访问正向索引。由于我们使用了学习索引，这会带来模型估计和本地搜索的非常规开销。即使我们使用了𝐵+-树，它可能需要访问索引的内部或叶子页面。(II) 跳过函数可能只会跳过很少的页面；实际上，在许多情况下，它只会返回下一个页面。

相反，我们提出了一种基于查询分割的新颖主动跳过策略，该策略在像我们的这样的单调SFCs上特别有效。我们在以下示例中说明其思想。

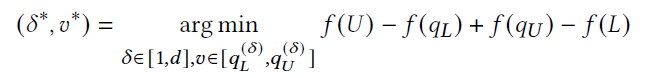
示例5. 考虑图5(a)中相同的查询（黑色矩形）。我们可以通过在𝑥轴上的值8处对其进行切割，将查询分成两部分，如图5(b)所示。我们仍然用蓝色绘制误报的部分。与未进行查询分割的情况相比，它减少了误报部分的数量 — 例如，部分[4, 7] × [12, 15]被消除。

**6.1 递归查询分割**

我们首先介绍一个计算将查询精确分割为两个子查询的最佳方式（即，最优的1-分割）的过程，然后我们将其推广，基于递归分割获得多个子查询。

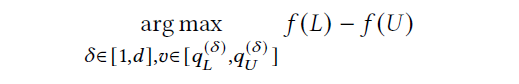
最优1-分割算法。考虑一个对应于𝑧-地址范围[𝑓(𝑞𝐿)，𝑓(𝑞𝑈)]的查询𝑞。不失一般性，假设我们在第𝛿维的值𝑣处进行分割。这将把查询分成两个子查询，相应的𝑧-地址范围为[𝑓(𝑞𝐿)，𝑓(𝑈)]和[𝑓(𝐿)，𝑓(𝑞𝑈)]，如图5(b)所示。我们将分割的成本定义为𝑓(𝑈) − 𝑓(𝑞𝐿) + 𝑓(𝑞𝑈) − 𝑓(𝐿)，或者直观地说，两个结果子查询的𝑧-地址范围之和。选择这个成本函数是因为它与分割后实际查询处理成本高度相关，并且可以在不访问数据点的情况下轻松计算。然后，我们将最优1-分割问题规定为找到分割（即，维度和值），使得这种分割的成本最小。

或者正式地说，

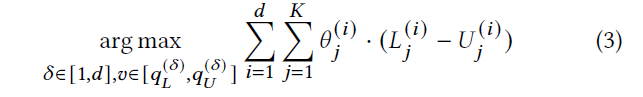


请注意，𝑈 和 𝐿 都由 𝛿 和 𝑣 确定，但为了方便表述，我们省略了符号依赖关系。

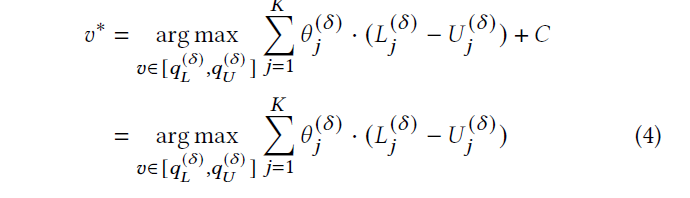
由于对于固定的查询 𝑞，𝑓(𝑞𝐿) 和 𝑓(𝑞𝑈) 是常数，上述最小化等同于以下最大化问题，即找到𝑓(𝑈) 和 𝑓(𝐿) 之间的最大“间隙”：



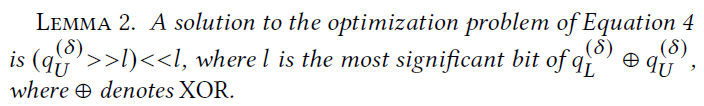
代入𝑓(·)的定义，则变为：



对于固定的𝛿 ∈ [1, 𝑑]，我们可以找到最佳分割值𝑣\*：





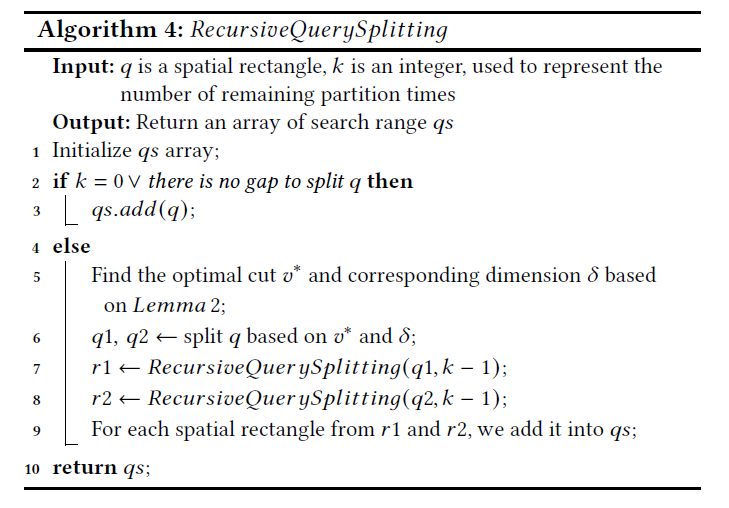


因此，我们可以通过找到每个 𝑑 维度的最优切割值来解决最优1-分割问题，因此，复杂度仅为 𝑂(𝑑)。

我们注意到引理2成立是因为 𝑈(𝛿) + 1 = 𝑣 = 𝐿(𝛿) 以及 𝜃 𝑗+1 ≥ 2𝜃𝑗（从引入的约束轻松推导出来以保证单调性）。如果单调性不成立，那么可能需要检查每个可能的 𝑣 值来执行优化，因此时间复杂度为 𝑂(∥𝑞𝑈 − 𝑞𝐿 ∥1)，这意味着对于“大”查询来说，所得到的过程可能更昂贵。

示例6. 再次考虑图5(a)中的例子。查询 𝑞 = [4, 11] × [4, 11]，学习到的 SFC 对应于参数 𝜃 = [ [20, 23, 25, 27], [21, 22, 24, 26]]。因此，𝑞 的 𝑧-地址范围是 [𝑓(𝑞𝐿), 𝑓(𝑞𝑈)] = [48, 207]。我们的最优1-分割算法将首先考虑 𝑥-轴。在这种情况下，最高有效位 𝑙 = 3，因为 (0100)2 ⊕ (1011)2 = (1111)2。然后，轴上的 𝑣∗ 是 (1011)2 >> 3 << 3 = (1000)2 = 8，然后可以计算在 𝑥-轴上以 8 进行分割的成本。

**递归分割**。由于一次分割通常不足以减少无关页面的数量，我们采用我们的最优1-分割算法进行递归，将查询窗口分割成多个部分。在我们的实现中，停止条件被设置为达到 𝑘maxsplit 的递归深度，或者没有间隙可供分割。𝑘maxsplit 是一个参数，可以平衡索引访问次数与不相交页面跳过机会的数量。较高的 𝑘maxsplit 可以有效地过滤掉不相交的页面，但会导致更多的索引访问开销。相反，较低的 𝑘maxsplit 可以节省索引访问的成本，但可能无法消除足够多的无关页面。我们在算法4中提供了递归查询分割的伪代码。在每次递归中，我们找到每个 𝑑 维度的最优切割，然后选择具有最大“间隙”的分割（第5行）。基于分割值 𝑣∗ 和相应的维度 𝛿，一个空间矩形可以分割成两个，并传递到下一次迭代（第6-8行）。



7.实验

**关于更新**

我们的方法还可以通过使用可更新的前向索引（例如 ALEX [5] 或 LIPP [40]）来处理更新。实际上，我们也可以使用传统的 𝐵+-树索引作为前向索引。

在这一节中，我们选择将 ALEX 作为前向索引，因为 PGM 没有固有的更新方法8。删除可以简单地将记录标记为"已删除"。我们这里仅讨论插入操作。一旦接收到插入查询，LMSFC 首先定位新插入点应该位于的数据页，根据其映射值。如果还有空间，我们将此点插入页面并更新元数据。否则，我们根据 𝑧-地址均匀地将页面分成两部分，还需要更新两个页面的元数据。我们使用来自 OSM 数据集的 128M 采样数据来初始化和学习 LMSFC。我们与两个 R-tree 变体进行比较，即 𝑅∗-tree 和带有线性分割方法的 R-tree（表示为 R-tree），我们还实现了一个可更新的 Flood 和 ZMindex（我们还使用 ALEX 处理数据更新）进行比较。

我们插入了 10%n 到 100%n 的数据点（𝑛 = 128M），并测试了窗口查询性能，报告了插入查询工作负载的平均查询时间和吞吐量。

插入后每次查询性能的结果显示在图13中。我们观察到可更新的学习索引仍然显示出比传统索引模型更好的查询时间。LMSFC实现了比其他索引模型更好的查询性能。我们认识到，尽管这样处理更新是可行的，但它可能会逐渐降低我们的索引质量。因此，在每次插入10%的数据后，我们触发一个学习过程，寻找在LMSFC中更好的学到的排序，称为LMSFCa。因此，性能可以进一步提高。尽管这会导致额外的学习开销，但学习阶段可以在实际系统中的单独实例中进行。因此，学习阶段不会阻塞传入的查询。

图14展示了学习索引的高吞吐量。由于启发式分割方法，R-tree变体显示出较低的吞吐量。当我们插入前10%的数据时，我们观察到LMSFC的吞吐量较低。这是因为许多溢出页面需要进行分割。我们可以在ZM-index中找到类似的趋势，但由于ZM-index在每个页面中不包含任何优化（即，页面级别的排序），因此ZM-index显示出最高的吞吐量。支持学习索引中插入的另一个思想是维护一个增量索引来处理数据插入并延迟合并/分割过程[35]。我们借鉴延迟更新策略的思想，并设计了一种名为LMSFCb的LMSFC变体。在每个数据页中，我们维护一个低容量（即，页面大小的一半）的未排序增量数组来处理每个页面中的溢出。一旦增量数组的大小超过预定义的阈值，我们触发与可更新的LMSFC相同的分割方法。这种方法的优势在于延迟更新策略可以在花费较低的增量数组扫描成本的情况下提高插入吞吐量。因此，图13和图14报告了LMSFCb实现了竞争性的查询性能和更好的插入吞吐量。

**8.结论和未来工作**

在本文中，我们研究了基于学习空间填充曲线的多维数据的学习索引问题。我们设计了一个学习特殊类别的空间填充曲线的框架，使其能够进行高效的查询处理。此外，我们通过优化数据放置到页面和查询分割，进行离线和在线两方面的优化，以进一步提高查询处理效率。广泛的实验结果表明，所提出的方法在三个真实数据集的各种设置下均优于非学习索引（如𝑅∗-tree）和先前先进的学习多维索引（如ZM-index和Flood）。

尽管本文主要关注学习单调空间填充曲线，但我们的思想和方法可以轻松推广以获得学习非单调空间填充曲线。例如，通过放宽对 𝜃 的约束，我们的方法可以学习非单调空间填充曲线。另一个例子是考虑其他参数化的空间填充曲线族，这些族泛化了其他知名的空间填充曲线，如希尔伯特曲线。我们将这样的探索留待将来的工作。

