DBMS 查询优化技术综述: 基数估计、代价模型和计划枚举

Hai Lan
RMIT University
Melbourne, Australia
hai.lan@rmit.edu.au

Zhifeng Bao
RMIT University
Melbourne, Australia
zhifeng.bao@rmit.edu.au

Yuwei Peng*
Wuhan University
Wuhan, China
ywpeng@whu.edu.cn

摘要

查询优化器是数据库系统的核心。目前几乎所有的数据库系统都采用了本文所研究的基于成本的优化器。基于成本的优化器引入计划枚举算法来找到一个 (子) 计划，然后使用成本模型来获得该计划的成本，并选择成本最低的计划。在成本模型中，基数 (cardinality)，即通过一个算子得到的元组数量，起着至关重要的作用。由于基数估计的不准确、代价模型的误差以及巨大的计划空间，优化器无法在合理的时间内找到复杂查询的最优执行计划。在本文中，我们首先深入研究上述局限性背后的原因。接下来，我们回顾了用于提高基于成本的优化器、基数估计、成本模型和计划枚举中三个关键组件质量的技术。我们还提供了我们对上述每个方面的未来方向的见解。

KEYWORDS

查询优化器; 基数估计; 成本模型; 计划枚举

1 介绍

查询优化器是关系数据库管理系统 (rdbms) 和一些大数据处理引擎的核心，例如 SCOPE [7]。给定一个用声明性语言 (例如 SQL) 编写的查询，优化器会找到最高效的执行计划 (也称为物理计划)，并将其提供给执行器。因此，大多数时候，用户只考虑如何将他们的需求转换为有效的查询，而不需要分析如何高效地运行查询。几乎所有的系统都采用基于 System R [79] 或 Volcano/Cascades [27, 28]。

图 1 说明了基于成本的优化器中的三个最重要的组件: **基数估计 (CE)**, **成本模型 (CM)**, 和 **计划枚举 (PE)**。**CE** 使用数据统计和一些关于数据分布、列相关性和连接关系的假设来获得由中间操作符[1] 生成的元组的数量，这对于其他搜索问题也很重要，例如, [101, 102]. **CM** 可以被认为是一个复杂的函数，它将数据库的当前状态和估计的基数映射到执行一个 (子) 计划的成本。**PE** 是一种探索语义等价连接顺序空间并以最小代价找到最优连接顺序的算法。寻找最优连接顺序主要有两种方法: 通过动态规划的自底向上的连接枚举和通过记忆的自顶向下的连接枚举 理论上，只要估

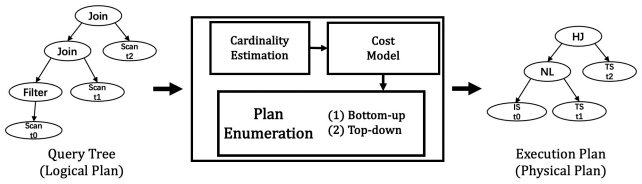

图 1: 查询优化器架构。IS、HJ、NL 和 TS 分别指索引扫描、哈希连接、嵌套循环连接和表扫描。

计的基数和成本是准确的，并且计划枚举组件可以高效地遍历巨大的搜索空间，这种架构可以在合理的时间内获得最优的执行计划。然而，它在现实中失败了。尽管经过了几十年的努力，基于成本的查询优化器仍然会在“困难”的查询上犯错误，这是由于 **CE** 中的错误、构建精确 **CM** 的困难以及为复杂查询寻找最佳**连接顺序 (PE)** 的痛苦。细节在第2节中介绍，即为什么现有的优化器仍然远远不能令人满意。有很多研究提

[1]在某些上下文中，数据库区域中的基数指的是不同的计数 [35].

出了提高优化器的能力。本文将对这些研究进行综述。具体来说，我们回顾了提出提高优化器中三个关键组件 (即 **CE**, **CM**, **PE**) 能力的出版物。

本文做出了以下贡献:

(1) 我们总结了 **CE**、**CM** 和 **PE** 表现不佳的原因 (第2节) 。
(2) 我们回顾了为更准确地估计基数而提出的研究。根据所使用的技术，将其分为基于概要的方法、基于抽样的方法和基于学习的方法 (第3节)。
(3) 回顾了成本模型的改进工作。我们将它们分为三组: 现有成本模型的改进，成本模型的替代方案，以及单个查询的性能预测 (第4) 节)。
(4) 我们回顾了计划枚举中使用的技术，并研究了用于处理大型查询的非学习方法。此外，我们回顾了最近提出的方法，这些方法采用强化学习来选择连接顺序 (第 5节)。
(5) 在第3-5, 我们分别介绍了我们对未来方向的见解。

有两个相关的调查。1998 年，Chaudhuri [8] 用非学习方法综述了关于查询优化器的工作。近二十年来，提出了许多方法来提高查询优化器的性能。有必要对这些新工作进行回顾。最近，Zhou et al. [107] 研究了如何在 DBMS 的不同部分，如监控、调优、优化器中引入 AI。在本文中，我们将重点放在查询优化器上，并对优化器的三个关键组件进行了全面的综述。我们同时总结了基于学习的和非学习的方法，详细回顾了这些工作，并提出了每一种方法可能的未来方向。

2 为什么优化器中的关键组件仍然不准确?

在这一节中，我们总结了基数估计、成本模型和计划枚举分别表现不佳的原因。本文回顾的研究试图通过处理这些不足来提高优化器的质量。

2.1 基数估计

基数估计是估计由一个算子生成的元组的能力，并在成本模型中用于计算该算子的成本。Lohman [62] 指出，成本模型最多可以引入 30% 的误差，而基数估计很容易引入多个数量级的误差。Leis et al. [56] 通过实验重新审视了具有复杂工作负载的经典优化器中的组件 **CE**, **CM**, and **PE** in the classical optimizers with complex workloads. 他们关注多连接查询的物理计划的质量，并得到与 Lohman 相同的结论。

基数估计的误差主要在三种情况下介绍:

(1) **带有谓词的单表中的误差**。数据库系统通常采用直方图作为数据的近似分布。直方图比原始数据要小。因此，它不能完全代表真实的分布，提出了一些假设 (如单一属性上的均匀性，不同属性之间的独立性假设)。当这些假设不成立时，就会发生估计误差，导致次优的方案。表中属性之间的相关性并不罕见。多直方图已经被提出。然而，它的存储量很大。
(2) **多连接查询中的误差**。来自不同表的列可能存在相关性。然而，没有有效的方法来获取两个或更多表之间的概要。针对这种情况，引入了包含原则。连接操作符的基数是使用包含原则及其子操作符的基数来计算的。当假设不成立时，它会有很大的误差。此外，对于具有多表的复杂查询，估计误差可以从规划的叶子传播和放大到根部。商业和开源数据库系统的优化器仍然在多连接查询 [56] 的基数估计方面挣扎。
(3) **用户自定义函数误差**。大多数数据库系统都支持用户定义函数 (UDF)。当条件中存在 UDF 时，没有通用的方法来估计有多少元组满足它 [8].

2.2 成本模型

基于成本的优化器使用成本模型来生成一个 (子) 查询的成本估计。(子) 计划的成本是其中所有算子成本的总和

操作的成本取决于部署数据库的硬件、操作的实现、操作处理的元组数量以及当前的数据库状态 (例

如，缓冲区中的数据、并发查询) [65]。因此，在结合所有因素时，应该确定大量的魔数，基数估计的误差也会影响成本模型的质量。此外，当基于成本的优化器部署在分布式或并行数据库、云环境或跨平台查询引擎中时，成本模型的复杂性正在急剧增加。此外，即使有真实基数，查询的成本估计与运行时间也不是线性的，这可能会导致次优的执行计划 [46, 81]。

2.3 计划枚举

使用计划枚举算法从语义等价的连接顺序空间中寻找最优连接顺序，使查询代价最小。它已被证明是一个 NP-hard 问题 [42]. 对于具有多连接查询的大型数据库来说，穷举查询计划枚举是一项令人望而却步的任务。因此，探索由最优连接顺序或近似最优连接顺序组成的正确搜索空间并设计高效的枚举算法至关重要。搜索空间中的连接树可以是 zig-zag 树、左深树、右深树和茂密树或它们的子集。不同的系统考虑不同形式的连接树。在传统的数据库系统中有三种枚举算法:(1) 通过动态规划 (DP) 进行自底向上的连接枚举 (例如 System R [79]), 通过记忆进行自顶向下的连接枚举 (例如, Volcano/Cascades [27, 28]), 以及 (3) 随机化算法 (例如 PostgreSQL 中的遗传算法 [12]，具有多个表连接)。

计划枚举有三个局限性:(1) 基数估计和成本模型的误差，(2) 用于剪枝搜索空间的规则，(3) 处理具有大量表的查询。当查询涉及大量表时，优化器必须牺牲最优性并使用启发式方法来保持优化时间合理，例如 PostgreSQL 中的遗传算法和 DB2 中的贪心方法，这些方法通常会生成糟糕的计划。

我们应该注意到基数中的错误将传播到成本模型中，并导致次优的连接顺序。消除或减少基数中的错误是构建一个有能力的优化器的第一步正如 Lohman [62] 所说的“万恶之源，查询优化的致命弱点，是对中间结果大小的估计，称为基数”。

在以下三个部分中，我们总结了为处理 CE、CM 和 PE 中的限制所做的研究工作，即如何使查询优化器更好。

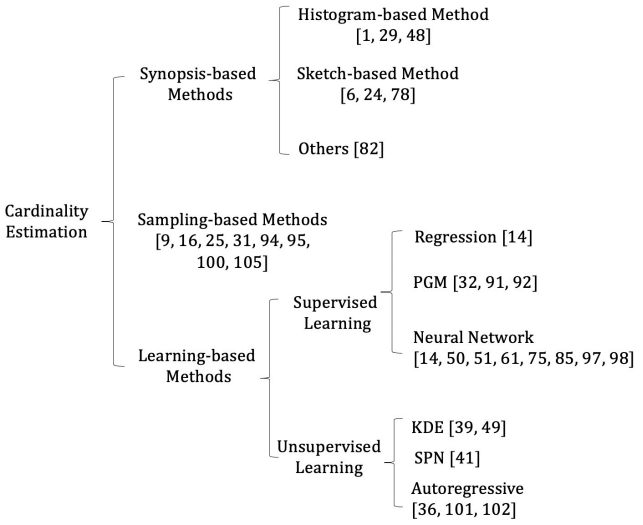

图 2: 基数估计方法的分类

3 基数估计

目前，基数估计主要有三种策略，如图2所示。我们只列出了每个类别的一些代表性工作。每种方法都试图近似分布. 用较少的存储来很好地估计数据。一些提出的方法结合了不同的技术，例如，[91, 92].

3.1 基于概要的方法

基于概要的方法引入了新的数据结构来记录统计信息。直方图和草图是被广泛采用的形式。2012 年 [10] 提出了概要的调查，重点是区分概要与近似查询处理 (Approximate Query Processing, AQP) 相关的方面。

3.1.1 **直方图**. 直方图有两种类型: *1* 维直方图和 d 维直方图, 其中 $d \geq 2$. d 维直方图可以捕捉到不同属性之间的相关性。

属性 A 上的一维直方图是通过将排序后的元组划分为 B (≥ 1) 个互不相交的子集来构建的，称为桶 (buckets)，并以某种常见的方式近似每个桶中的频率和值，例如均匀分布和连续值。属性组 A 上的 d 维直方图是通过对 A 的联合数据分布进行划分来构建的，因为不同属性之间没有顺序，所以划分规则需要更加复杂。2003 年，Ioannidis [44]遵循 [77] 的分类方法，对直方图进行了全面的综述。Gunopulos et al. [30]也在

2003 年提出了一项调查，重点是用于估计多属性选择性的工作。他们总结了多维直方图和核密度估计器。2003 年以后，在直方图方面的工作可以分为三类:(1) 快速的直方图构建算法 [1, 29, 33, 34, 43];(2) 新的划分方法，将数据分成不同的桶，以达到更好的精度 [15, 59, 88];(3) 基于查询反馈的直方图构建 [10] 。查询反馈方法也在 [10](第 3.5.1.2 节) 中进行了总结，读者可以参考其细节。

Guha et al. [29]分析了之前的算法 VODP [45]，发现了一些关于最小误差平方和 (SSE) 可以减少的计算。他们设计了一种高效的算法 AHistL-Δ 时间复杂度 $O(n+B^3(\lg n+\epsilon^{-2}))$，而 VODP 需要 $O(n^2B)$，其中 n 是域大小，B 是桶的数量，ϵ 是精度参数。Halim et al. [33, 34]提出了一种基于贪婪局部搜索的快速直方图构建算法 GDY。GDY 生成良好的样本边界，然后使用 VODP 最优地构建 B final 分区。本研究比较了 GDY 变体与 AHistL-*Delta* [29] 在最小化所有桶的总误差方面的差异，并显示了其在解决效率-质量权衡方面的优势。Indyk et al. [43]没有扫描整个数据集 [29]，而是设计了一种贪婪算法，在时间复杂度 $O((B^5/\epsilon^8)\log^2 n)$ 和样本复杂度为 $O((B\epsilon)^2\log n)$ 的数据集随机样本上构建直方图。Acharya et al. [1]用 [43] 研究了同样的问题，并提出了时间复杂度 $O(1/\epsilon^2)$ 的合并算法。[1, 29, 43] 中的方法可以通过分段多项式扩展到近似分布。

考虑到基于树的索引将数据划分为不同的段 (节点)，这与直方图中的桶非常相似，Eavis and Lopez [15]基于 R-tree 构建了多维直方图。他们首先在 Hilbert 类型的数据上建立了一个原生 R-tree 直方图，然后提出了一种滑动窗口算法来增强朴素直方图，该算法寻求最小化桶点之间的死区。Lin et al. [59]为一个属性设计了一个两级直方图，这与 B-tree 索引的思想相当相似。第一级用于定位要使用哪些叶子直方图，叶子直方图存储了统计信息。To et al. [88]基于最小化直方图的熵减原则构建直方图。他们为相等查询设计了两种不同的直方图，并设计了一种增量算法来构建直方图。然而，它只考虑了一维直方图，并没有很好地处理范围查询。

3.1.2 **草图**. 草图将一列建模为向量或矩阵，以计算不同值的计数 (例如 HyperLogLog [22]) 或元组的频率 (例如 count Min [11])。Rusu and Dobra [78]总结了如何使用不同的草图来估计连接大小。这项工作考虑了两个没有过滤器的表 (或数据流) 的情况。它们的基本思想是:(1) 在连接属性上构建草图 (一个向量或矩阵)，同时忽略所有其他属性，(2) 根据向量或矩阵的乘法估计连接大小。这些方法只支持等值连接和单列上的连接。如 [94] 所示，在草图中引入一个过滤器的一种可能的方法是建立一个虚表，该表只由满足过滤器的元组的连接值组成。然而，这使得估计结果大大变差。Skimmed sketch [24] 是基于双焦采样 [25] 的思想来估计连接大小。然而，它需要知道最频繁的连接属性值的频率。最近关于连接大小估计的工作 [6] 引入了草图来记录一个值的度。

3.1.3 **其他技术**. TuG [82] 是一种基于图形的摘要。TuG 的节点表示来自同一表的一组元组或同一属性的一组值。边代表不同表之间或属性与值之间的连接关系。作者采用三步算法构造 TuG，并引入直方图来总结节点上的值分布。当出现新查询时，通过遍历 TuG 来估计选择性。构建过程相当耗时，并不能在大型数据集上使用。如果没有不同表之间的关系，就无法构建 TuG。

3.2 基于采样的方法

基于概要的方法很难捕捉不同表之间的相关性。一些研究人员试图使用特定的采样策略从表中收集一组样本 (元组)，然后在样本上运行 (子) 查询来估计基数。只要得到的样本分布接近原始数据，基数估计就是可信的。因此，人们提出了大量的工作来设计一个好的采样方法，从独立采样到相关采样技术。基于抽样的方法也在 [10] 中进行了总结。2011 年之后，有大量的研究利用了抽样技术。与 [10] 不同，我们主要总结新工作。此外，我们根据它们的出版时间对这些作品进行了回顾，并呈现了它们之间的关系，即前一部作品的不足之处，以及后一部作品试图克服的地方。1993 年，Haas et al. [31]分析了等值连接查询的 6 种固定步长

的采样方法。他们得出结论，如果有一些建立在连接键上的索引，那么结合索引的页面级采样是最好的方法。否则，页面级的跨积采样是最有效的方法。然后，作者将固定步长方法扩展到固定精度程序。

Ganguly et al. [25]引入双焦采样来估计等值连接的大小。他们根据频率将每个关系中的连接属性值分为两组，稀疏 (*s*) 和稠密 (*d*)。因此，元组之间的连接类型可以是 *s-s*、*s-d*、*d-s* 和 *d-d*。作者首先采用 *t_cross* sampling [31] 来估计 *d-d* 的连接大小，然后采用 *t_index* 来估计剩余案例的连接大小，最后将所有的估计相加作为连接大小估计。然而，它需要额外的一遍来确定不同值的频率，并需要索引来估计 *s-s*、*s-d* 和 *d-s* 的连接大小。如果没有索引，这个过程是非常耗时的。端偏置采样 [16] 存储 (v, f_v) 如果 $f_v \geq T$，其中 v 是 join 属性域中的一个值，f_v 是值 v 的元组的数量，T 是一个定义的阈值。它应用一个哈希函数 $h(v) : v \mapsto [0, 1]$。如果 $h(v) \leq \frac{f_v}{T}$，它存储 (v, f_v) 或不存储。不同的表采用相同的哈希函数来关联它们对低频率元组的采样决策。然后可以使用存储的 (v, f_v) 对来估计连接大小。但是，它只支持两个表上的等值连接，不能处理其他过滤条件。注意，端偏置采样与双焦采样非常相似。不同之处在于: 前者使用哈希函数对相关元组进行采样，后者使用索引。它们都需要对数据进行额外的遍历，以计算连接属性值的频率。

Yu et al. [105]将相关采样作为 CS2 算法的一部分引入。他们 (1) 在连接图中选择一个表作为源表 R_1，(2) 使用随机抽样方法获取 R_1 的样本集 S_1(将 R_1 标记为已访问)，(3) 在连接图中遵循一条未访问的边 $< R_i, R_j >$ (R_i 被访问) 并收集来自 R_j 的元组, 这些元组可与元组在 S_i 中连接为 S_j, , in the join graph and collect the tuples from R_j which are joinable with tuples in S_i as S_j(4) 估计样本上的连接大小。以支持无源的查询表，他们提出了一个反向估计器，跟踪到源表来估计连接大小。然而，由于多次遍历连接图，在没有索引的情况下非常耗时。此外，它需要一个不可预测的大空间来存储样本。Vengerov et al. [94] 提出了一种不需要 join 属性频率先验知识的相关采样方法，如 [16, 25]。如果 $h(v) < p$，其中 $p = \frac{n}{T}$，$h(v)$ 是类似于 [16] 中的哈希函数，则包含在样本集中的具有连接值 v 的元组，n 是样本量，T 是表大小。然后，我们可以使用获得的样本来估计连接大小，并处理指定的过滤条件。此外，作者将该方法扩展到更多的表连接和复杂的连接条件。在大多数情况下，相关采样比独立伯努利采样 (*t_cross*) 具有更低的方差，但当许多连接属性的值以大频率出现时，伯努利采样更好。作者提出的一种可能的解决方案是采用 One-pass 算法来检测频率较高的值，这又回到了 [16] 中的方法。

通过实验，Chen and Yi [9]得出结论，不存在一种适用于所有情况的采样方法。他们提出了一种基于独立伯努利抽样、端偏抽样 [16] 和相关抽样 [94] 的两级抽样方法。一级抽样对一个值 v 进行抽样将属性域加入值集 (V)，如果 $h(v) < p_v$。h 是类似于 [16] 的散列函数，p_v 是值 v 的定义概率。在第二级抽样之前，他们为每个在 V 的 v 元组集合抽样一个随机元组，称为哨兵。二级采样用概率 q 对值为 v ($v \in V$) 的元组进行采样。然后，我们可以利用元组样本估计连接大小。显然，第一级是相关抽样，第二级是独立伯努利抽样。作者分析了如何根据不同的连接类型设置 p_v 和 q 以及连接属性中值的频率。Wang and Chan [95]在五个参数方面将 [9] 扩展到一个更一般的框架。基于新的框架，他们提出了一类新的相关采样方法，称为 CSDL，它是基于离散学习算法的。作为 CSDL 的一种变体，CSDL-opt 在样本较小或连接值密度较小时表现优于 [9] 。Wu et al. [100]采用在线采样来纠正优化器生成的计划中可能存在的错误。

3.3 基于学习的方法

由于基于学习的方法的能力，许多研究人员引入了基于学习的模型来捕获数据的分布和相关性。我们将它们分为:(1) 有监督的方法，(2) 无监督的方法。

3.3.1 **有监督的方法**. Malik et al. [64]将查询分组成模板，并采用机器学习技术 (如线性回归模型、树模型) 来学习每个家族的查询结果大小的分布。其中使用的特征包括查询属性、常量、操作符、聚合和 UDFs 的参数。

表 1: 基于学习的基数估计的方法

Refs	Model	Model Count	Encoding	Multi-columns	Multi-tables	UDF	Workload S
[64]	LR	1 Model/1 Template	predicates, arguments	✓	✓	✓	×
[76]	MixModel	1 Model	predicates	✓	×	×	✓
[91, 92]	BN	1 Model	predicates	✓	✓	×	×
[32]	BN	1 Model/1 Table	predicates	✓	×	×	×
[54]	NN	1 Model/1 UDF	arguments	×	×	✓	×
[61]	NN	1 Model	predicates	✓	×	×	×
[98]	NN/PR/MLR	1 Model/1 Subquery	predicates, input cardinalities	✓	✓	✓	×
[50]	MSCN	1 Model	predicates, tables, joins	✓	✓	×	×
[14]	Tree-Ensemble/NN	1 Model	predicates	✓	×	×	×
[96, 97]	NN	1 Model/1 Template	predicates	✓	✓	×	×
[75]	DNN/RNN/Tree	1 Model	predicates, tables, joins	✓	✓	×	×
[85]	tree-LSTM	1 Model	predicate, operator, metadata	✓	✓	×	×
[39]	KDE	1 Model	samples	✓	×	×	✓
[49]	KDE	1 Model 1 Model / 1 Table	samples	✓	✓	×	✓
[41]	SPN	1 Model	tuples; predicates	✓	✓	×	✓
[36, 104]	Autoregression	1 Model	tuples; predicates	✓	×	×	✓
[103]	Autoregression	1 Model	tuples; predicates	✓	✓	×	✓

表 2: 基数估计的不同方法的初步比较

Methods	Workload Shift	Data Change	Update Time	Storage Usage	Multi-Columns	Multi-Tables
1-histogram[44]	×	×	Short	Small	×	×
d-histogram [30]	×	×	Short	Large	✓	×
Sampling [9, 95, 100]	✓	×	Medium	Large	✓	✓
Supervised Learning [50, 85]	×	×	Long	Small	✓	✓
Unsupervised Learning [41, 103]	✓	×	Long	Small	✓	✓

Park et al. [76]在查询驱动范式中提出了一个类似于 [48, 58, 83] 的模型 QuickSel，用于估计一个查询的选择性。而不是采用直方图，QuickSel 引入均匀混合模型来表示的分布数据。他们通过最小化混合模型和均匀分布之间的均方误差来训练模型。

Tzoumas et al. [91, 92]构建了一个贝叶斯网络，将多个属性的复杂统计数据分解为小的一维/二维统计数据，这意味着该模型最多捕获两个关系之间的依赖关系。他们为这些小维度统计建立直方图，并采用动态规划来计算新查询的选择性。与之前的方法 [26] 不同，它可以处理更一般的连接，并且由于捕获了更小的依赖关系，因此具有更高效的构造算法。然而，作者并没有在多表连接和大型数据集上验证他们的方法。此外，用来自不同表的属性构造二维统计需要连接操作。Halford et al. [32]也介绍了一种基于贝叶斯网络的方法。为了快速构建模型，他们只对每个关系内部的属性分布进行因式分解，并使用之前的假设进行连接。然而，与 [91, 92] 相比，他们并没有展示他们的方法有多好。

1998 年，Lakshmi and Zhou [54]首次将神经网络引入到用户定义函数 (UDF) 的基数估计中，这是直方

图和其他统计数据无法支持的。他们设计了一个双层神经网络 (NN)，并使用反向传播来更新模型。

Liu et al. [61]形式化一个选择性函数，$Sel: R^{2N} \mapsto R, (l_1, u_1, ..., l_n, u_n) \mapsto c$，其中 N 为属性数量，l_i 和 u_i 为查询的第 i 个属性的上下边界。他们采用 3 层神经网络学习选择性函数。为了支持 > 和 <，他们添加了 2N 个极小的神经网络来生成 l_i 和 u_i

Wu et al. [98]在共享云中使用基于学习的工作负载方法，其中的查询在本质上经常是重复和重叠的。他们首先在多个查询图中提取重叠子图模板。然后，他们学习这些子图模板的基数模型。

Kipf et al. [50]引入了多集卷积网络 (MSCN) 模型来估计相关连接的基数。它们将查询表示为一组表 T, 连接 J, 和预测 P 的集合，并为每个表连接和预测构建单独的 2 层神经网络。然后，将三个神经网络的输出经过平均运算后串联起来，并馈入最终的输出网络。Deep sketch [51] 建立在 [50] 之上，是它的一个包装器。

Dutt et al. [14]将估计形式化为类似于 [61] 的函数，他们认为这是一个回归问题。他们对回归问题采用了两种不同的方法，基于神经网络的方法和基于树的集成。与 [61] 不同的是，作者还使用直方图和领域知识 (例如 AVI、EBO 和 MinSel) 作为模型中的额外特征，这提高了估计精度。由于当数据分布发生变化时，领域知识会快速更新，因此模型对数据集上的更新具有鲁棒性。

Woltmann et al. [97] 认为在整个数据库模式上构建一个称为全局模型的单一神经网络具有稀疏编码，并且需要大量样本来训练模型。因此，他们为不同的查询模板建立了不同的模型，称为局部模型。每个局部模型都采用多层感知器 (MLP) 来产生基数估计。为了收集真实基数，在训练过程中会发出许多样本查询，这是非常耗时的。此外，Woltmann et al. [96]引入了使用数据立方体概念对基础数据进行预聚合的方法，并对这些预聚合的数据执行示例查询。

Ortiz et al. [75]对基数估计中使用的各种深度学习方法进行了实证分析，包括深度神经网络 (DNN) 和循环神经网络 (RNN)。DNN 模型与 [97] 类似。为了采用 RNN 模型，作者将重点放在左深计划上，并将查询建模为一系列动作。每个动作代表一个操作 (即选择或连接)。在每个时间戳 ⊠ 中，模型接收两个输入:x_t，操作的编码 t^{th}，和 h_{t-1}，从时间戳 $t-1$ 生成的隐藏状态，可以视为子查询的编码，并捕获关于中间结果的重要细节。

Sun and Li [85] 引入了一种 tree-LSTM 模型来学习算子的表示，并在 tree-LSTM 模型上添加了一个估计层来同时估计基数和代价。

3.3.2 **无监督方法**. Heimel et al. [39] 将核密度估计器 (Kernel Density Estimator, KDE) 引入到使用多个谓词估计单个表上的选择性。首先采用高斯核和一定规则得到的带宽构造初始 KDE，然后利用历史查询通过最小化初始 KDE 估计误差来选择最优带宽。为了支持工作负载和数据集的变化，他们在每次传入查询后更新带宽，并为只插入的工作负载和更新/删除工作负载设计新的样本维护方法。此外，在 2017 年，Kiefer et al. [49]将该方法扩展到估计 join 的选择性。他们设计了两种不同的模型: 连接样本上的单一模型和基表上的模型，不需要连接操作，用独立假设估计连接的选择性。

Yang et al. [104]提出了一种名为 Naru 的模型，该模型采用深度自回归模型在一组 n-dimensional 元组上产生 n 条件密度 $\hat{P}(x_i|x_{<i})$。然后他们使用乘积法则来估计选择性:

$$\begin{aligned}\hat{P}(\mathbf{x}) =& \hat{P}(x_1, x_2, ..., x_n) \\ =& \hat{P}(x_n|x_1, ..., x_{n-1})\hat{P}(x_{n-1}|x_1, ..., x_{n-2})...\hat{P}(x_2|x_1)\hat{P}(x_1)\end{aligned} \tag{1}$$

为了支持范围条件，他们引入了一种渐进式采样方法，根据训练好的自回归模型从更有意义的区域采样点，这对偏斜数据具有鲁棒性。此外，他们采用通配符跳跃式来处理通配符条件

Hasan et al. [36]也采用了深度自回归模型，并引入了自适应采样方法来支持范围查询。与 Naru 算法相比，采用二进制编码方法，采样过程并行进行，使得模型比 Naru 算法更小，推理速度更快。此外，在定义交叉熵损失函数时，它可以根据工作负载为元组分配权重，从而与工作负载相结合。

Hilprecht et al. [41]引入了关系和积网络 (Relational Sum Product Network, RSPN) 来捕获单个属性的分布和联合概率分布。他们关注的是 Tree-SPNs，其中一个叶子是单一属性的近似，内部节点是 Sum 节点 (将行分割成簇) 或 Product 节点 (将一个簇的列分割)。为了支持连接的基数估计，他们在连接结果上构建 RSPN。

Yang et al. [103]扩展了他们之前的工作，Naru，以支持连接。他们在所有表的全外连接之上建立了一个自回归模型。他们为大基数列引入了无损列分解，并采用连接计数表来支持表子集上的任何查询。

3.4 我们的见解

3.4.1 **总结**. 基本的直方图类型 (如等宽、等深、d 维) 在 2000 年之前就已经介绍过了。最近的研究主要集中在如何快速构建直方图和提高直方图的准确性。通过查询反馈更新直方图是提高直方图质量的一个很好的方法。然而，在直方图中仍然存在两个限制:(1) 在构建 d 维直方图时，存储大小急剧增加;(2) 直方图无法捕获来自不同表的属性之间的相关性。如果为来自不同表的属性建立直方图，则需要连接操作，如 [91, 92]，并且很难更新此直方图。

草图可以用来估计一个属性的不同计数或等值连接结果的基数。但是，它不能很好地支持更一般的情况，例如，与过滤器连接。当一个或两个子关系都是中间关系时，基于概要的方法不能估计最终关系或中间关系的大小。

抽样是一个很好的方法来捕获不同表之间的相关性。然而，当表中的元组被更新时，样本可能会变得过时。基于抽样的方法还会受到用于存储样本的存储空间和用于检索样本的时间的影响，特别是在原始数据较多的情况下。此外，目前的采样方法只支持等值连接。

监督学习方法大多是查询驱动的，这意味着模型是针对特定的工作负载进行训练的。如果工作负载发生变化，则需要对模型进行重新训练。因此，数据驱动 (无监督学习) 的方法就出来了，即使工作量发生变化，它仍然可以估计基数。如 [104](第 6.3 节) 所示，Naru 对工作负载转移具有鲁棒性，而 MSCN 和 KDE 对训练查询敏感。此外，监督学习和无监督学习方法都受到数据变化的影响。如 [103](第 7.6 节) 和 [85](第 7.5 节) 所述，它们都对数据变化敏感，模型将以增量模式更新或从头开始重新训练。然而，他们只考虑到新的元组被追加到一个表中，并且不存在删除或更新操作。由于实验设置的差异，我们只对表2所示的基数估计方法进行了初步比较。有时，基于学习的方法的规模仍然不小，如 [103] 所示。最先进的方法是 [103]，在他们的实验中与其他方法相比显示出了优越性。

3.4.2 **未来可能的方向**. 有几个可能的方向如下:

(1) **基于学习的方法**。关于基数估计的许多研究都是最近两年基于学习的方法。目前集成在真实系统中的基于学习的模型是轻模型或一个（子）查询图的一个模型 [14, 98]，可以快速训练和更新。然而，这些模型的准确性和通用性都是有限的。更复杂的模型 (达到更好的精度) 仍然会受到长时间训练和更新时间的影响。训练时间受硬件影响。例如，在 [103] 中只需要几分钟，而在 [85] 中使用性能相对较差的 GPU 则需要 13 小时。一个数据库实例，尤其是在云环境中，处在一个资源受限的环境中。应该考虑如何高效地训练模型。模型和优化器之间的交互也需要考虑，这在数据库系统 [14] 上不应该有太多的开销。如上所述，目前提出的数据更改方法不能处理删除或更新操作。一种可能的方法是采用主动学习的思想来更新模型 [63]。
(2) **混合方法**查询驱动的方法对工作量敏感。虽然数据驱动方法可以支持更多的查询，但它可能无法对所有的查询都达到最好的准确性。如何在这两个不同的目录中结合两种方法是一个可能的方向。实际上，前面的查询-反馈直方图就是这种情况的一个实例。另一件有趣的事情是，利用查询反馈信息将帮助模型意识到数据的变化。
(3) **实验研究**虽然已经提出了许多方法，但缺乏对这些方法进行验证的实验研究。不同的方法有不同的特点，如表2所示。对所提出的方法进行

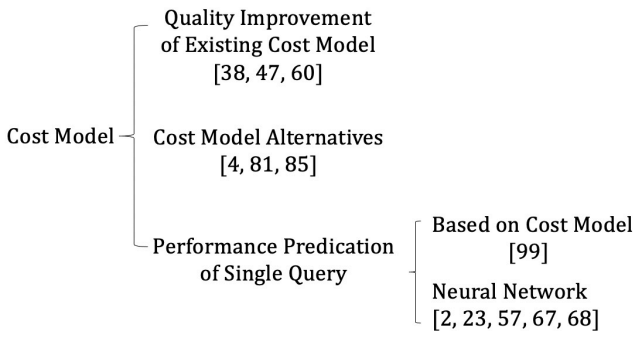

图 3: A classification of cost estimation methods

全面的实验研究至关重要。我们认为应该包括以下几个方面:(1) 该方法是否容易集成到一个真实的数据库系统中;(2) 在不同的工作负载模式(例如,静态或动态工作负载,OLTP 或 OLAP)和不同的数据规模和分布下，该方法的性能如何;(3) 研究人员应注意候选估计的存储使用和准确性之间的权衡，以及模型更新效率和准确性之间的权衡。

4 成本模型

在这一节中，我们介绍了为解决成本模型中的局限性而提出的研究。我们将这些方法分为三组:(1) 改进现有代价模型的能力，(2) 构建新的代价模型，以及 (3) 预测查询性能。我们包括关于单个查询性能预测的工作。因为优化器中使用的成本是性能的指标。这些方法可能集成到基于成本的优化器中，并取代成本模型来估计一个 (子) 计划的成本，如 [67]。但是，我们没有考虑并发上下文下的查询性能预测(例如,[108])。一方面，优化过程中存在的并发查询可能与执行过程中的查询有很大的不同。在另一方面，它也需要收集比预测单个查询性能的模型更多的信息。我们在图3中列出了成本模型中的代表性工作。

4.1 现有成本模型的质量改进

一些研究试图估算 UDF 的成本 [3, 37, 38]. Boulos and Ono [3]使用不同的输入值多次执行 UDF，收集不同的成本，然后使用这些成本构建多维直方图。直方图存储在树形结构中。在估计具有特定参数的 UDF 时，自顶向下遍历树以获得估计的代价，从而定位具有类似参数和输入的叶子。但是，这种方法需要知道每个参数的上界和下界，无法解决输入参数和成本之间的复杂关系。与 [3] 中使用的静态直方图不同，He et al. [37]引入了一种基于四分树的动态方法来存储 UDF 执行信息。当执行查询时，将使用执行 UDF 的实际成本来更新成本模型。He et al. [38]引入了内存受限的 k 近邻 (MLKNN) 方法。他们设计了一个名为 PData 的数据结构来存储执行成本和一个多维索引，该索引用于快速检索给定查询点 (UDF 中的参数) 最近的 k 个 PData，并快速插入新的 PData。Liu and Blanas [60] 为内存数据库引入了一种基于哈希连接的代价模型。他们将查询的响应时间建模为与四种基本访问模式的代价加权的操作数量成正比。他们首先采用微基准来获得每个访问模式的成本,然后通过基本访问模式对顺序扫描、哈希连接、不同顺序的哈希连接的成本进行建模。之前的成本模型大多只考虑执行成本，在云环境下可能不合理。云数据库系统的用户关心的是经济成本。Karampaglis 等人，Karampaglis et al. [47]首先提出了一个双目标查询成本模型，用于在多云环境下同时推导运行时间和货币成本。他们基于 [99] 中的方法对执行时间进行建模。对于经济成本估算，他们首先对收费政策进行建模，并通过结合政策和时间估算来估算货币成本。

4.2 成本模型替代方案

成本模型是将带有注释信息的 (子) 计划映射到标量 (成本) 的函数。由于数据上的神经网络主要近似于从输入到输出的未知底层映射函数，因此用于取代原始成本模型的大多数方法都是基于学习的，尤其是基于 NN 的。Boulos et al. [4]首先介绍了用于成本评估的神经网络。他们设计了两个不同的模型: 为每个查询类型设计一个单一的大型神经网络，为每个操作员设计一个单一的小型神经网络。在第一个模型中，他们还训练另一个模型，将查询分类为某一类型。第一个模型的输出是一个 (子) 计划的成本，而第二个模型需要将小模型的输出相加才能得到成本。Sun and Li [85]采

用 tree-LSTM 模型来学习算子的表示，并在 tree-LSTM 模型上增加一个估计层来估计查询计划的成本。由于大数据系统，特别是现代云数据服务中收集统计数据的困难和挑选资源的需要，Siddiqui 等人 Siddiqui et al. [81]提出了一种基于学习的成本模型，并将其集成到 SCOPE [7] 的优化器中。他们构建了大量的小型模型来预测公共 (子) 查询的成本，这些成本是从工作负载历史中提取的。编码到模型中的特征与 [98] 非常相似。此外，为了支持资源感知的查询规划，他们将分配给操作员的分区数量添加到特征中。为了提高模型的覆盖率，他们引入 operator-input 模型和 operator- subgraphapprox 模型，并采用元集成模型将上述模型组合为最终模型。

4.3 查询性能预测

一个查询的性能主要是指延迟。Wu et al. [99]采用离线 profiling 在特定的软硬件条件下校准成本模型中的系数。然后，他们采用抽样的方法来获得物理算子的真实基数来预测执行时间。Ganapathi et al. [23]将核典型相关分析 (Kernel Canonical Correlation Analysis, KCCA) 引入到 CPU 时间等资源估计中。他们只对计划级信息进行建模，例如每个物理算子类型的数量及其基数，这太脆弱。为了估计资源 (CPU 时间和逻辑 I/O 时间)，Li et al. [57]为数据库中的每个算子训练了一个增强的回归树，该计划的消耗是算子的总和。为了使模型更鲁棒，他们为每个算子训练一个单独的缩放函数，并将缩放函数与回归模型结合起来，以处理数据分布、大小或查询的参数与训练数据有很大差异的情况。与 [23] 不同的是，这是一个算子级别的模型。

Akdere et al. [2]提出了基于学习的模型来预测查询性能。他们首先设计了一个计划级模型 (如果工作量提前已知) 和一个算子级模型 (operator level model)。考虑到计划级模型的预测精度很高，而算子级模型的泛化能力很好，对于算子级预测精度较低的查询，他们使用计划级建模为特定查询子计划训练模型，并将两种类型的模型组合为预测整个计划的性能。然而，采用的模型是线性的。Marcus and Papaemmanouil [68]引入了一种计划结构神经网络来预测查询的延迟。他们为每个逻辑算子设计了一个小的神经网络，称为神经单元 (neural unit)，同一个逻辑算子的任何实例共享同一个网络。然后，这些神经单元根据规划树组合成一个树状。一个神经单元的输出由两部分组成，当前算子的延迟和发送到其父节点的信息。一个计划的根神经单元的延迟就是该计划的延迟。Neo [67] 是一个基于学习的查询优化器，它引入了一个叫做价值网络的神经网络来估计 (子) 计划的延迟。查询级编码 (用预测连接图和列) 通过几个全连接层，然后与计划级编码连接，这是一个表示物理计划的树向量。接下来，将连接后的向量输入到一个树卷积和另几个全连接层中，以预测输入物理计划的延迟。

4.4 我们的洞见

4.4.1 **总结**. 试图改进现有成本模型的方法集中在不同方面，例如 udf、主内存中的哈希连接。这些研究给我们留下了一个重要的教训: 当引入一个新的逻辑或物理算子，或重新实现现有的物理算子时，我们应该考虑如何将它们添加到优化过程中，并为它们设计相应的成本估计公式 (例如，[55, 71])。基于学习的方法采用该模型来捕捉成本与影响因素之间的复杂关系，而传统的成本模型由数据库专家定义为一个确定的公式。3.3.1节中用于预测性能、估计成本和估计基数的基于 NN 的方法在特征和模型选择方面非常相似。例如，Sun and Li [85]使用相同的模型来估计成本和基数，[67] 使用 (子) 计划的延迟 (性能) 作为成本。一个能够捕捉数据本身、算子级别信息和子计划信息的模型，可以准确地预测成本。例如，最先进的方法之一的工作 [85]，采用 tree-LSTM 模型来捕获上述信息。然而，它们都是有监督的方法。如果工作量转移或数据更新，模型需要从头开始重新训练。

4.4.2 **未来可能的方向**. 可能的方向有以下两种:

(1) **云数据库系统**云数据库系统的用户需要以最低的价格满足其延迟或吞吐量。将运行查询的经济成本整合到成本模型中是一个可能的方向。将这些相关信息考虑到成本模型中是很有趣的。例

如，Siddiqui et al. [81]将容器数量考虑到他们的成本模型中。

(2) **以学习为基础的方法**基于学习的成本估计方法也会遇到与基数估计方法相同的问题(第3.4.2节)。在实际系统中采用的模型是一个轻模型 [81]。精度和训练时间之间的权衡仍然是一个问题。基数估计中采用的可能解决方案也可以用于成本模型。

5 计划枚举

在本节中，我们介绍了已发表的关于计划枚举问题的研究成果。我们将计划枚举的工作分为两类，非学习型方法和学习型方法。

5.1 非学习型方法

1997 年，Steinbrunn et al. [84]提出了一项关于选择最优连接顺序的代表性调查。因此，我们主要关注 1997 年以后的研究。

5.1.1 **动态规划**. Selinger et al. [79]提出了一种动态规划算法来为给定的连接查询选择最优连接顺序。他们按照增大大小的顺序生成计划，并将搜索空间限制在左深树，这显著加快了优化速度。Vance and Maier [93]提出了一种动态规划算法，通过考虑不同的部分表集来寻找最优连接顺序。他们用它来生成包含叉积的最优茂密树连接树。[79, 93] 是生成-测试范式，大部分操作用于检查子图是否连通和两个子图是否合并。因此，它们都不满足 [74] 中的下界。Moerkotte and Neumann [69]提出了一种基于图的动态规划算法。他们首先引入了一种基于图的方法来生成连通子图。因此，它不需要检查连接和组合，并更有效地执行。然后，他们采用 DP over 它们来生成无叉积的最优茂密树。Moerkotte and Neumann [70]扩展了 [69] 中的方法来处理非内连接和更广义的图，超图，其中连接谓词可以涉及两个以上的关系。

5.1.2 **自顶向下策略**. TDMinCutLazy 是 DeHaan and Tompa [13]提出的第一种高效的自顶向下连接枚举算法。他们利用最小割的思想对连接图进行划分，并在自顶向下的划分搜索中引入两种不同的剪枝策略，预测代价边界和累积代价边界，可以避免穷举枚举。自顶向下的方法几乎和动态规划一样高效，并且具有其他诱人的性质，例如，剪枝，有趣的顺序。Fender and Moerkotte [18] 提出了另一种自顶向下的连接枚举策略 (TDMinCutBranch)。TDMinCutBranch 引入了一种基于图的枚举策略，它只生成有效的连接，即 cross-product free partitions，而 TDMinCutLazy 采用生成和测试的方法。次年，Fender et al. [21]提出了另一种自上而下的枚举策略 TD- MinCutConservative，与 TDMinCutBranch 相比，该策略更容易实现，运行时性能也更好。此外，Fender and Moerkotte [19, 20]提出了一个处理非内连接的通用框架和一个更广义的图，超图，用于自顶向下的连接枚举。[80] 中提出了一种基于 top-down 连接枚举的新型连接转换规则 RS-Graph，以高效地生成跨积自由连接树的空间。

5.1.3 **大型查询**. 对于大型查询，贪婪算子 [17] 通过首先添加最有利可图 (具有最小的中间连接大小) 的连接，自下而上地构建茂密的连接树。为了支持大型查询，Kossmann and Stocker [52]提出了两种不同的增量动态规划方法，IDP-1 和 IDP-2。对于给定规模 k, IDP-1 运行 [79] 中的算法，构造该规模最便宜的计划，并将其视为基关系，重复该过程。在给定大小 k 的情况下，在每次迭代中，IDP-2 首先执行贪心算法构建有 k 个表的连接树，然后在生成的连接树上运行 DP 生成最优计划，并将该计划作为基关系，然后重复该过程。Neumann [72]提出了一种两阶段算法: 首先，通过贪心启发式对查询图进行简化，直到图对 DP 变得可处理，然后运行 DP 算法，为简化后的连接图找到最优的连接顺序。Bruno et al. [5]引入了 enumerate-rank-merge 框架，对之前的启发式算法进行了概括和扩展 [17, 86]。枚举步骤考虑了茂密的树。排序步骤用于评估每一步的最佳连接对，它采用最小大小的度量标准。合并步骤构建选定的连接对。Neumann and Radke [73]根据查询图类型和表的数量，将查询分为三种类型: 小型查询、中型查询和大型查询。然后，他们采用 DP 来

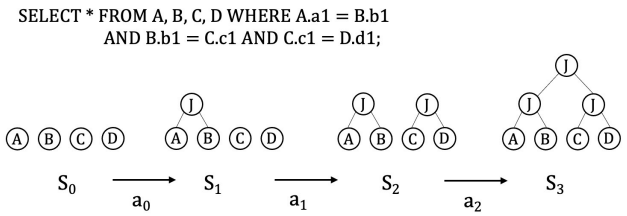

图 4: One possible join order episode

解决小型查询，通过线性化中等查询的搜索空间来限制 DP，并使用 [52] 中的思想来解决大型查询。

5.1.4 **其他**. 。Trummer and Koch [89]将连接排序问题转化为一个混合整数线性规划，以最小化计划的成本，并采用现有的 MILP 求解器得到线性连接树 (左深树)。为了满足 MILP 的线性特性，他们通过线性函数来近似扫描和连接操作的代价。现有的大多数 OLAP 系统主要关注于 start/snowfake 连接查询 (PK-FK 连接关系)，并生成左深二叉连接树。在处理 FK-FK 连接时，就像在 TPC-DS(暴风雪模式) 中一样，它们会产生大量的中间结果。Nam et al. [71] 引入了一种新的 n 元连接操作符，扩展了计划空间。他们定义了核心图来表示连接图中的 FK-FK 连接，并采用 n 元连接算子对其进行处理。他们为该运营商设计了一个新的成本模型，并将其集成到现有的 OLAP 系统中。

5.2 基于学习的方法基于学习的方法

所有基于学习的方法都采用强化学习 (DL)。在 RL 中，代理通过行动和奖励与环境相互作用。在每一步 t，代理使用一个策略 π 选择一个动作 a_t 根据当前状态 s_t 和转换到新状态 s_{t+1}。然后环境应用该动作 a_t 返回奖励 r_t 给代理。RL 的目标是学习一个策略函数，它根据当前状态自动采取行动，并获得最大的长期回报。在 join order selection 中，state 是当前的子树，action 是结合两个子树，如图4所示。中间动作的奖励是 0，最后一个动作的奖励是查询的成本或延迟。ReJoin [66] 采用深度强化学习 (deep reinforcement learning, DRL) 来识别最优连接顺序，该方法在影响最大化 [87] 等其他领域也得到了广泛应用。DRL 中的状态表示当前子树。每个动作都会将两个子树合并成一棵树。它使用从优化器中的成本模型中获得的成本作为奖励。ReJoin 对状态中的 (子) 计划、连接谓词和选择谓词的树结构进行编码。与 [66] 不同，Heitz and Stockinger [40]在每一行中创建一个矩阵来表示一个表或子查询，并采用 [56] 中的代价模型来快速获得一个查询的代价。DQ [53] 也是一种基于 drl 的方法。它使用 one-hot 向量对 (子) 查询中的可见属性进行编码。DQ 还通过添加另一个单热向量对物理算子的选择进行编码。在训练模型时，DQ 首先使用从优化器的成本模型中观察到的成本，然后使用真实的运行时间对模型进行微调。Yu et al. [106]采用 DRL 和 tree-LSTM 相结合的方式进行连接顺序选择。与以往的方法 [40, 53, 66] 不同，tree-LSTM 可以捕获更多查询树的结构信息。与 [53] 类似，它们也使用成本来训练模型，然后切换到运行时间作为微调的反馈。注意，他们还讨论如何处理数据库模式中的变化，例如添加/删除表/列。[90] 采用了 UCT，一种强化学习算法，并从当前查询中学习，而之前基于学习的连接顺序方法是从以前的查询中学习。它将一个查询的执行划分为许多小切片，不同的小切片可能会选择不同的连接顺序，并从以前的执行切片中学习。

5.3 我们的见解

5.3.1 **摘要**. 非基于学习的研究侧重于提高现有方法的效率和能力 (处理更一般的连接情况)。与动态规划方法相比，自顶向下的策略具有更好的可扩展性，例如，添加新的转换规则、分支和边界修剪。这两种方法都已在许多数据库系统中得到实现。与非学习方法相比，基于学习的方法具有更快的规划时间。所有基于学习的方法都采用了强化学习。它们之间的主要区别是:(1) 选择哪些信息作为状态以及如何对它们进行编码，(2) 采用哪些模型。具有更多相关信息的更复杂的模型可以实现更好的性能。最先进的方法 [106] 采用与 [85] 类似的 tree-LSTM 模型来生成子计划的表示。由于成本模型的不准确性，它可以通过使用延迟对模型进行微调来提高模型的质量。尽管目前最先进的方法 [106] 优于非基于学习的方法在他们的实验中显示，如何将

基于学习的方法集成到真实的系统中是必须解决的问题。

5.3.2 **未来可能的方向**. 有以下两种可能的方向:

(1) **处理大型查询**所有被提议处理大型查询的方法都是基于自底向上的 dp 方法。剩下的问题是: 如何使自顶向下的搜索策略支持大规模查询。此外，目前最先进的用于大型查询的方法 [73] 不能支持一般的连接情况。

(2) **基于学习的方法**目前的倾斜方法只关注 PK-FK 连接，连接类型为内连接。如何处理其他连接类型是一个可能的方向。所提出的方法中，没有一个讨论过如何将它们集成到一个真实的系统中。在他们的实验中，他们将该方法实现为一个单独的组件，以获得正确的连接顺序，然后发送到数据库。数据库仍然需要对其进行优化，才能得到最终的物理方案。如果一个查询有子查询，它们可能会交互多次。强化学习方法是在一定的环境下进行训练的，在连接顺序选择问题中是指一定的数据库。如何处理表模式和数据的变化也是一个可能的方向。

6 结论

在基于成本的优化器中，基数估计、成本模型和计划枚举在生成最优执行计划方面发挥着关键作用。本文回顾了为提高其质量而提出的工作，包括传统方法和基于学习的方法。此外，我们还分别提供了可能的未来方向。越来越多的基于学习的方法被引入，并优于传统方法。然而，它们的训练和更新时间很长。如何使模型对工作量变化和数据变化具有鲁棒性或快速更新模型仍然是一个开放的问题。具有理论保证的传统方法在现实系统中被广泛采用。用新的算法和数据结构改进传统方法的可能性很大。此外，我们相信传统方法背后的思想可以用来增强基于学习的方法。

ACKNOWLEDGMENTS

Zhifeng Bao is supported in part by ARC DP200102611, DP180102050, and a Google Faculty Award.

REFERENCES

[1] Jayadev Acharya, Ilias Diakonikolas, Chinmay Hegde, Jerry Zheng Li, and Ludwig Schmidt. 2015. Fast and Near-Optimal Algorithms for Approximating Distributions by Histograms. In *PODS*. 249–263.

[2] Mert Akdere, Ugur Çetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. 2012. Learning-based Query Performance Modeling and Prediction. In *ICDE*. 390–401.

[3] Jihad Boulos and Kinji Ono. 1999. Cost Estimation of User-Defined Methods in Object-Relational Database Systems. *SIGMOD Rec.* 28, 3 (1999), 22–28.

[4] Jihad Boulos, Yann Viemont, and Kinji Ono. 1997. A neural networks approach for query cost evaluation. *Transaction of Information Processing Society of Japan* 38, 12 (1997), 2566–2575.

[5] Nicolas Bruno, César A. Galindo-Legaria, and Milind Joshi. 2010. Polynomial heuristics for query optimization. In *ICDE*. 589–600.

[6] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic Cardinality Estimation: Tighter Upper Bounds for Intermediate Join Cardinalities. In *SIGMOD*. 18–35.

[7] Ronnie Chaiken, Bob Jenkins, Per-Åke Larson, Bill Ramsey, Darren Shakib, Simon Weaver, and Jingren Zhou. 2008. SCOPE: easy and efficient parallel processing of massive data sets. *VLDB* 1, 2 (2008), 1265–1276.

[8] Surajit Chaudhuri. 1998. An Overview of Query Optimization in Relational Systems. In *SIGMOD*. ACM Press, 34–43.

[9] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *SIGMOD*. 759–774.

[10] Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends Databases* 4, 1-3 (2012), 1–294.

[11] Graham Cormode and S. Muthukrishnan. 2004. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications. In *LATIN 2004: Theoretical Informatics, 6th Latin American Symposium, Buenos Aires, Argentina, April 5-8, 2004, Proceedings*. 29–38.

[12] PostgreSQL Database. 2020. http://www.postgresql.org/. Accessed June 10, 2020.

[13] David DeHaan and Frank Wm. Tompa. 2007. Optimal top-down join enumeration. In *SIGMOD*. 785–796.

[14] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *VLDB* 12, 9 (2019), 1044–1057.

[15] Todd Eavis and Alex Lopez. 2007. Rk-hist: an r-tree based histogram for multi-dimensional selectivity estimation. In *CIKM*. 475–484.

[16] Cristian Estan and Jeffrey F. Naughton. 2006. End-biased Samples for Join Cardinality Estimation. In *ICDE*. 20.

[17] Leonidas Fegaras. 1998. A New Heuristic for Optimizing Large Queries. In *DEXA*. 726–735.

[18] Pit Fender and Guido Moerkotte. 2011. A new, highly efficient, and easy to implement top-down join enumeration algorithm. In *ICDE*. 864–875.

[19] Pit Fender and Guido Moerkotte. 2013. Counter Strike: Generic Top-Down Join Enumeration for Hypergraphs. *VLDB* 6, 14 (2013), 1822–1833.

[20] Pit Fender and Guido Moerkotte. 2013. Top down plan generation: From theory to practice. In *ICDE*. 1105–1116.

[21] Pit Fender, Guido Moerkotte, Thomas Neumann, and Viktor Leis. 2012. Effective and Robust Pruning for Top-Down Join Enumeration Algorithms. In *ICDE*. 414–425.

[22] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frederic Meunier. 2007. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics and Theoretical Computer Science* (2007), 137–156.

[23] Archana Ganapathi, Harumi A. Kuno, Umeshwar Dayal, Janet L. Wiener, Armando Fox, Michael I. Jordan, and David A. Patterson. 2009. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *ICDE*. 592–603.

[24] Sumit Ganguly, Minos N. Garofalakis, and Rajeev Rastogi. 2004. Processing Data-Stream Join Aggregates Using Skimmed Sketches. In *EDBT*. 569–586.

[25] Sumit Ganguly, Phillip B. Gibbons, Yossi Matias, and Abraham Silberschatz. 1996. Bifocal Sampling for Skew-Resistant Join Size Estimation. In *SIGMOD*. 271–281.

[26] Lise Getoor, Benjamin Taskar, and Daphne Koller. 2001. Selectivity Estimation using Probabilistic Models. In *SIGMOD*. 461–472.

[27] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.

[28] Goetz Graefe and William J. McKenna. 1993. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*. IEEE Computer Society, 209–218.

[29] Sudipto Guha, Nick Koudas, and Kyuseok Shim. 2006. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.* 31, 1 (2006), 396–438.

[30] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. 2005. Selectivity estimators for multidimensional range queries over real attributes. *VLDB J.* 14, 2 (2005), 137–154.

[31] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. 1993. Fixed-Precision Estimation of Join Selectivity. In *PODS*. 190–201.

[32] Max Halford, Philippe Saint-Pierre, and Franck Morvan. 2019. An Approach Based on Bayesian Networks for Query Selectivity Estimation. In *DASFAA*. 3–19.

[33] Felix Halim, Panagiotis Karras, and Roland H. C. Yap. 2009. Fast and effective histogram construction. In *CIKM*. 1167–1176.

[34] Felix Halim, Panagiotis Karras, and Roland H. C. Yap. 2010. Local Search in Histogram Construction. In *AAAI*.

[35] Hazar Harmouch and Felix Naumann. 2017. Cardinality Estimation: An Experimental Survey. *Proc. VLDB Endow.* 11, 4 (2017), 499–512.

[36] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries. In *SIGMOD*. 1035–1050.

[37] Zhen He, Byung Suk Lee, and Robert R. Snapp. 2004. Self-tuning UDF Cost Modeling Using the Memory-Limited Quadtree. In *EDBT*, Vol. 2992. 513–531.

[38] Zhen He, Byung Suk Lee, and Robert R. Snapp. 2005. Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *TODS* 30, 3 (2005), 812–853.

[39] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. In *SIGMOD*. 1477–1492.

[40] Jonas Heitz and Kurt Stockinger. 2019. Join Query Optimization with Deep Reinforcement Learning Algorithms. *CoRR* abs/1911.11689 (2019).

[41] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *VLDB* 13, 7 (2020), 992–1005.

[42] Toshihide Ibaraki and Tiko Kameda. 1984. On the Optimal Nesting Order for Computing N-Relational Joins. *ACM Trans. Database Syst.* 9, 3 (1984), 482–502.

[43] Piotr Indyk, Reut Levi, and Ronitt Rubinfeld. 2012. Approximating and testing k-histogram distributions in sub-linear time. In *PODS*. 15–22.

[44] Yannis E. Ioannidis. 2003. The History of Histograms (abridged). In *VLDB*. Morgan Kaufmann, 19–30.

[45] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. 1998. Optimal Histograms with Quality Guarantees. In *VLDB*. 275–286.

[46] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. 2020. ML-based Cross-Platform Query Optimization. In *ICDE*. 1489–1500.

[47] Zisis Karampaglis, Anastasios Gounaris, and Yannis Manolopoulos. 2014. A Bi-objective Cost Model for Database Queries in a Multi-cloud Environment. In *MEDES*. 109–116.

[48] Raghav Kaushik and Dan Suciu. 2009. Consistent Histograms In The Presence of Distinct Value Counts. *VLDB* 2, 1 (2009), 850–861.

[49] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating Join Selectivities using Bandwidth-Optimized Kernel Density Models. *VLDB* 10, 13 (2017), 2085–2096.

[50] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.

[51] Andreas Kipf, Dimitri Vorona, Jonas Müller, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, Thomas Neumann, and Alfons Kemper. 2019. Estimating Cardinalities with Deep Sketches. *CoRR* abs/1904.08223 (2019).

[52] Donald Kossmann and Konrad Stocker. 2000. Iterative dynamic programming: a new class of query optimization algorithms. *TODS* 25, 1 (2000), 43–82.

[53] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph M. Hellerstein, and Ion Stoica. 2018. Learning to Optimize Join Queries With Deep Reinforcement Learning. *CoRR* abs/1808.03196 (2018).

[54] M. Seetha Lakshmi and Shaoyu Zhou. 1998. Selectivity Estimation in Extensible Databases - A Neural Network Approach. In *VLDB*. 623–627.

[55] Jyoti Leeka and Kaushik Rajan. 2019. Incorporating Super-Operators in Big-Data Query Optimizers. *VLDB* 13, 3 (2019), 348–361.

[56] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *VLDB* 9, 3 (2015), 204–215.

[57] Jiexing Li, Arnd Christian König, Vivek R. Narasayya, and Surajit Chaudhuri. 2012. Robust Estimation of Resource Consumption for SQL Queries using Statistical Techniques. *VLDB* 5, 11 (2012), 1555–1566.

[58] Lipyeow Lim, Min Wang, and Jeffrey Scott Vitter. 2003. SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads. In *VLDB*. 369–380.

[59] Xudong Lin, Xiaoning Zeng, Xiaowei Pu, and Yanyan Sun. 2015. A Cardinality Estimation Approach Based on Two Level Histograms. *J. Inf. Sci. Eng.* 31, 5 (2015), 1733–1756.

[60] Feilong Liu and Spyros Blanas. 2015. Forecasting the cost of processing multi-join queries via hashing for main-memory databases. In *Socc*. 153–166.

[61] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *CASCON*. 53–59.

[62] Guy Lohman. 2014. IS QUERY OPTIMIZATION A "SOLVED"PROBLEM? http://wp.sigmod.org/?p=1075. Accessed June 10, 2020.

[63] Lin Ma, Bailu Ding, Sudipto Das, and Adith Swaminathan. 2020. Active Learning for ML Enhanced Database Systems. In *SIGMOD*. 175–191.

[64] Tanu Malik, Randal C. Burns, and Nitesh V. Chawla. 2007. A Black-Box Approach to Query Cardinality Estimation. In *CIDR*. 56–67.

[65] Stefan Manegold, Peter A. Boncz, and Martin L. Kersten. 2002. Generic Database Cost Models for Hierarchical Memory Systems. In *VLDB*. 191–202.

[66] Ryan Marcus and Olga Papaemmanouil. 2018. Deep Reinforcement Learning for Join Order Enumeration. In *aiDM@SIGMOD*. 3:1–3:4.

[67] Ryan C. Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A Learned Query Optimizer. *VLDB* 12, 11 (2019), 1705–1718.

[68] Ryan C. Marcus and Olga Papaemmanouil. 2019. Plan-Structured Deep Neural Network Models for Query Performance Prediction. *VLDB* 12, 11 (2019), 1733–1746.

[69] Guido Moerkotte and Thomas Neumann. 2006. Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. In *VLDB*. 930–941.

[70] Guido Moerkotte and Thomas Neumann. 2008. Dynamic programming strikes back. In *SIGMOD*. 539–552.

[71] Yoon-Min Nam, Donghyoung Han, and Min-Soo Kim. 2020. SPRINTER: A Fast n-ary Join Query Processing Method for Complex OLAP Queries. In *SIGMOD*. 2055–2070.

[72] Thomas Neumann. 2009. Query simplification: graceful degradation for join-order optimization. In *SIGMOD*. 403–414.

[73] Thomas Neumann and Bernhard Radke. 2018. Adaptive Optimization of Very Large Join Queries. In *SIGMOD*. 677–692.

[74] Kiyoshi Ono and Guy M. Lohman. 1990. Measuring the Complexity of Join Enumeration in Query Optimization. In *VLDB*. 314–325.

[75] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. 2019. An Empirical Analysis of Deep Learning for Cardinality Estimation. *CoRR* abs/1905.06425 (2019).

[76] Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. 2020. QuickSel: Quick Selectivity Learning with Mixture Models. In *SIGMOD*. 1017–1033.

[77] Viswanath Poosala, Yannis E. Ioannidis, Peter J. Haas, and Eugene J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *SIGMOD*. 294–305.

[78] Florin Rusu and Alin Dobra. 2008. Sketches for size of join estimation. *ACM Trans. Database Syst.* 33, 3 (2008), 15:1–15:46.

[79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD*. 23–34.

[80] Anil Shanbhag and S. Sudarshan. 2014. Optimizing Join Enumeration in Transformation-based Query Optimizers. *VLDB* 7, 12 (2014), 1243–1254.

[81] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost Models for Big Data Query Processing: Learning, Retrofitting, and Our Findings. In *SIGMOD*. 99–113.

[82] Joshua Spiegel and Neoklis Polyzotis. 2006. Graph-based synopses for relational selectivity estimation. In *SIGMOD*. 205–216.

[83] Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. 2006. ISOMER: Consistent Histogram Construction Using Query Feedback. In *ICDE*. 39.

[84] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. 1997. Heuristic and Randomized Optimization for the Join Ordering Problem. *VLDB J.* 6, 3 (1997), 191–208.

[85] Ji Sun and Guoliang Li. 2020. An End-to-End Learning-based Cost Estimator. *VLDB* 13, 3 (2020), 307–319.

[86] Arun N. Swami. 1989. Optimization of Large Join Queries: Combining Heuristic and Combinatorial Techniques. In *SIGMOD*. 367–376.

[87] Shan Tian, Songsong Mo, Liwei Wang, and Zhiyong Peng. 2020. Deep Reinforcement Learning-Based Approach to Tackle Topic-Aware Influence Maximization. *Data Sci. Eng.* 5, 1 (2020), 1–11.

[88] Hien To, Kuorong Chiang, and Cyrus Shahabi. 2013. Entropy-based histograms for selectivity estimation. In *CIKM*. 1939–1948.

[89] Immanuel Trummer and Christoph Koch. 2017. Solving the Join Ordering Problem via Mixed Integer Linear Programming. In *SIGMOD*. 1025–1040.

[90] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. In *SIGMOD*. 1153–1170.

[91] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2011. Lightweight Graphical Models for Selectivity Estimation Without Independence Assumptions. *VLDB* 4, 11 (2011), 852–863.

[92] Kostas Tzoumas, Amol Deshpande, and Christian S. Jensen. 2013. Efficiently adapting graphical models for selectivity estimation. *VLDB J.* 22, 1 (2013), 3–27.

[93] Bennet Vance and David Maier. 1996. Rapid Bushy Join-order Optimization with Cartesian Products. In *SIGMOD*. 35–46.

[94] David Vengerov, Andre Cavalheiro Menck, Mohamed Zaït, and Sunil Chakkappen. 2015. Join Size Estimation Subject to Filter Conditions. *VLDB* 8, 12 (2015), 1530–1541.

[95] TaiNing Wang and Chee-Yong Chan. 2020. Improved Correlated Sampling for Join Size Estimation. In *ICDE*. 325–336.

[96] Lucas Woltmann, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2020. Machine Learning-based Cardinality Estimation in DBMS on Pre-Aggregated Data. *CoRR* abs/2005.09367 (2020).

[97] Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. 2019. Cardinality estimation with local deep learning models. In *aiDM@SIGMOD*. 5:1–5:8.

[98] Chenggang Wu, Alekh Jindal, Saeed Amizadeh, Hiren Patel, Wangchao Le, Shi Qiao, and Sriram Rao. 2018. Towards a Learning Optimizer for

Shared Clouds. *VLDB* 12, 3 (2018), 210–222.

[99] Wentao Wu, Yun Chi, Shenghuo Zhu, Jun'ichi Tatemura, Hakan Hacigümüs, and Jeffrey F. Naughton. 2013. Predicting query execution time: Are optimizer cost models really unusable?. In *ICDE*. 1081–1092.

[100] Wentao Wu, Jeffrey F. Naughton, and Harneet Singh. 2016. Sampling-Based Query Re-Optimization. In *SGMOD*. 1721–1736.

[101] Yang Yang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Liping Wang. 2019. Selectivity Estimation on Set Containment Search. *Data Sci. Eng.* 4, 3 (2019), 254–268.

[102] Yang Yang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Liping Wang. 2019. Selectivity Estimation on Set Containment Search. In *DASFAA, Part I*. 330–349.

[103] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. *CoRR* abs/2006.08109 (2020). arXiv:2006.08109

[104] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Peter Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *VLDB* 13, 3 (2019), 279–292.

[105] Feng Yu, Wen-Chi Hou, Cheng Luo, Dunren Che, and Mengxia Zhu. 2013. CS2: a new database synopsis for query estimation. In *SIGMOD*. 469–480.

[106] Xiang Yu, Guoliang Li, Chengliang Chai, and Nan Tang. 2020. Reinforcement Learning with Tree-LSTM for Join Order Selection. In *ICDE*. 1297–1308.

[107] X. Zhou, C. Chai, G. Li, and J. SUN. 2020. Database Meets Artificial Intelligence: A Survey. *TKDE* (2020), 1–1.

[108] Xuanhe Zhou, Ji Sun, Guoliang Li, and Jianhua Feng. 2020. Query Performance Prediction for Concurrent Queries using Graph Embedding. *VLDB* 13, 9 (2020), 1416–1428.