计算机科学
COMPUTER SCIENCE

基于学习排序的查询优化算法

余阳, 彭煜玮

引用本文

余阳, 彭煜玮. 基于学习排序的查询优化算法[J]. 计算机科学, 2025, 52(8): 109-117.

YU Yang, PENG Yuwei. Query Optimization Algorithm Based on Learning to Rank[J]. Computer Science, 2025, 52(8): 109-117.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250100151

基于学习排序的查询优化算法

余 阳 彭煜玮
武汉大学计算机学院 武汉 430061
(yu_yang@whu.edu.cn)

摘 要 查询优化是关系型数据库中的关键环节。在传统的查询优化过程中,为了获得较优的执行计划,通常需要对查询中的连接和过滤操作进行基数估计。然而,基数估计存在不准确的问题,导致查询优化效果往往不尽如人意。目前,已有部分研究通过基于机器学习的方法改善基数估计问题并取得了一定进展。尽管这些方法在处理查询中数值类型的过滤谓词时表现较好,但对于其他复杂的过滤谓词效果不佳。为解决这一问题,文中提出了一种基于学习排序的查询优化算法。该算法能够为单一查询智能评估多个执行计划并排序,从而选择最佳计划执行。该查询优化算法通过迭代挖掘较优执行计划,并协同机器学习方法,最终筛选出最优计划。实验结果表明,该算法在常规数据集上的性能优于当前基于学习的查询优化算法,并且在复杂数据集中具有更加显著的优势。

关键词:查询优化;计划生成;学习排序;数据库;连接顺序;连接类型;扫描类型
中图分类号 TP392

Query Optimization Algorithm Based on Learning to Rank

YU Yang and PENG Yuwei
School of Computer Science, Wuhan University, Wuhan 430061, China

Abstract Query optimization is a key aspect of relational databases. In the traditional query optimization process, cardinality estimation of join and filter operations in a query is usually required in order to obtain a better execution plan. However, due to the inaccuracy of cardinality estimation, the results of query optimization are often unsatisfactory. Currently, some researches have been conducted to improve the cardinality estimation through machine learning-based methods and have made some progress. This paper finds that although these methods perform better in dealing with filtering predicates of numerical types in queries, they are ineffective for other complex filtering predicates. To address this problem, this paper proposes a query optimization algorithm based on learning to rank. The algorithm is capable of intelligently evaluating and ranking multiple execution plans for a single query to select the best plan for execution. The query optimization algorithm iteratively mines the better execution plans and collaborates with machine learning methods to finally filter out the optimal plan. Experimental results show that the proposed algorithm outperforms current learning-based query optimization algorithms on regular datasets and is more significant on complex datasets.

Keywords Query optimization, Plan generation, Learning to rank, Database, Join order, Join type, Scan type

1 引言

给定一个 SQL 查询,现有的数据库系统通常会通过调用优化器为这条 SQL 查询寻找高效的执行计划。目前常见的优化器主要有两类:基于规则的优化器(Rule Based Optimizer,RBO)和基于代价的优化器(Cost Based Optimizer,CBO)。RBO 基于一组固定的规则来优化查询,CBO 则根据不同的执行计划的代价来选择最优的方案。

现代应用常常需要处理海量的数据,CBO 会考虑统计信息和数据的分布,凭借其灵活性能够适应现代数据处理的复杂场景,提供更高效的查询性能。尽管 CBO 有数十年的研究积累,但在处理复杂查询时仍会因为错误的基数估计产生性能较差的执行计划[1]。

基数估计是影响执行计划生成的核心,准确的基数估计能够有效地帮助优化器生成好的执行计划。为了改善 CBO 的性能表现,许多研究者设计了机器学习的方法来估计基数[2-7],期望为 CBO 提供准确的基数以生成更好的执行计划。但是由于基数估计是对数据分布特征的评估,随着数据规模的扩大以及属性值的增加,这些机器学习的方法开始显现出局限性,无法很好地学习更复杂数据的分布特征。此外,机器学习的方法一旦给出了与真实情况差别很大的基数,可能会使优化器生成比原来更糟糕的计划,缺乏一定的容错性。

到稿日期:2025-01-23 返修日期:2025-03-07
基金项目:国家重点研发计划(2023YFB4503604)
This work was supported by the National Key R&D Program of China(2023YFB4503604).
通信作者:彭煜玮(ywpeng@whu.edu.cn)

因此，只通过机器学习的方法估计基数来生成执行计划具有一定的风险。

执行计划的特征主要有两点：表的连接顺序和计划节点的类型。在处理多表连接查询时，多表的连接顺序决定了查询的执行效率。通常情况下，CBO 会优先选择连接结果集较小的表连接。这样，小结果集能够减少后续操作的工作量，从而提高整体查询性能。在计划节点方面，不同的数据库系统支持的计划节点类型可能存在一定的差别。目前，常见的对执行计划影响较大的节点主要是扫描（SCAN）和连接（JOIN）。SCAN 类型节点一般又可分为 Sequence Scan，Index Scan 和 Index Only Scan。JOIN 类型节点常见的有 Hash Join，NestedLoop Join 和 Merge Join。通常情况下，CBO 会根据 SCAN 和 JOIN 所涉及的表进行统计数据的分析，然后选择合适的节点类型。

2 相关工作

现有的查询优化研究主要关注两个方面：执行计划生成和基数估计。执行计划生成关注如何快速产生高质量的候选执行计划集合供优化器选择，基数估计则关心如何给 CBO 提供准确基数以便从候选计划中选出最优计划。

2.1 执行计划生成

传统的执行计划生成方式采用动态规划算法，自底向上地构造执行节点。构造的过程中根据估计的基数来计算代价，从而选择代价最小的节点类型。

在基数准确的情况下，动态规划算法理论上可以构造出最佳执行计划。但由于基数估计不准，传统方式构造的“最佳计划”通常是伪最佳计划。因此，一些研究者开始尝试用直接影响执行计划生成的方式去探索较优的执行计划。

Bao[8] 尝试使用 6 种 Hint 的组合（3 种 Join Hint 和 3 种 Scan Hint），在不同的 Hint 组合下，首先迫使 CBO 生成一些执行计划，然后通过机器学习的方法对它们的执行时间进行预测，最后根据预测时间选出最佳执行计划。而在 Rong 等提出的框架 Lero[9] 中，他们认为既然基数估计不准确，那就尝试随机修改基数估计值的数量级，从而让 CBO 在不同估计值下尝试找到最佳的执行计划，最后通过学习排序的方法来预测最佳的执行计划。LEON[10] 是一个辅助 CBO 的查询优化框架，它引入了等价集的概念将相同连接（连接顺序可以不同）的子计划归为同一个等价集，利用学习排序的方法获取最佳子计划，并通过向 CBO 提供最佳子计划的方式辅助生成最终的执行计划。

2.2 基数估计

基数估计是对计划节点结果数的估计[11]。传统的基数估计主要使用采样的统计学方法，其结果通常使用柱状图等方法表示[12]。柱状图包括等宽和等高两种：1）在等宽柱状图中，属性值区间等宽，高度由落入该区间的元组数决定；2）在等高柱状图中，属性值区间等高，宽度由该区间元组所覆盖的属性值范围决定。无论是等高还是等宽的柱状图都需要采用统计学中采样的方法去构建，Graham 等[13] 总结了数据库中常用的采样方法。

除了以上的传统基数估计方法外，最近许多机器学习的基数估计方法被提出[14-17]。Benjamin 等[15] 提出了一种纯数据驱动的模型，它是一种不受限于特定工作环境来捕获数据分布的方法，并根据查询信息得到所需数据的分布特征。Shohedul 等[14] 使用深度学习模型有选择地估计查询谓词的选择率。Andreas 等[16] 则专注于查询连接的基数估计。在 Sun 等[17] 提出的方法中，一个查询被分成多个片段，每个片段使用一个神经网络去训练，当训练完成时，一个全局模型将决定用哪些片段模型进行基数估计。上述机器学习方法相比传统方法都有一定的性能提升，但也都存在一些局限性，比如不支持字符串类型的谓词属性等。

2.3 讨论

在执行计划生成的过程中，Lero 通过修改执行计划的基数估计值，来影响 CBO 生成的执行计划。这种方式的原理在于：基数估计值本身并不准确，随机修改估计值数量级有机会使修改后的数量级与真实情况相同，从而使得 CBO 能够按照最接近真实的基数去生成执行计划。但这种方式仍然存在一些问题。首先，由于 Lero 采用随机修改基数估计值的方式，因此不具备方向性。其次，当查询的连接表很多时，多个 Join 节点的基数修改值同时符合真实情况的概率大大降低，Lero 会通过修改基数值的方式产生更多的执行计划，带来生成效率的问题。最后，不当的基数修改甚至可能生成很多劣化的执行计划，对优化器的选择造成误导。

Bao 基数估计不准确会导致执行计划中某一个节点的类型选择错误。因此，其尝试组合 JH（Join Hint）和 SH（Scan Hint）的方式，希望能够让 CBO 按照 Hint 生成对应的执行计划。在基数估计不准时，该方法也能够找到更优的执行计划，但也存在和 Lero 类似的问题：Hint 的设置不具备方向性。JH 和 SH 的组合有很多种，需要枚举大量的执行计划才可能找到较优的执行计划，会带来生成效率不佳的问题。为了提高效率，Bao 目前只采用了 5 种 Hint 集合，但不能保证能够产生较优的执行计划。

本文将执行计划生成过程分为确定连接顺序和选择扫描节点类型两个阶段。该方法相比于 Bao 和 Lero 具备更好的方向性，产生的候选执行计划的质量也更高。同时，其减少了劣化执行计划生成，提高了较优执行计划的比例。

最后，由于本文提出的基于学习排序的查询优化算法主要专注于生成一批候选执行计划，然后排序选择出较优的执行计划，这与 Lero 和 Bao 探索执行计划的处理方式相似，因此主要将 Lero 和 Bao 作为对比对象。而 LEON 是辅助 CBO 逐步生成完整执行计划的框架，因此不与 LEON 进行比较。

3 本文方法整体框架

如图 1 所示，本文方法主要分为两个部分：CBO 和学习排序单元（Learn to Rank，L2R）。CBO 主要负责接受查询信息以及相关 Hint 来生成执行计划。学习排序单元则负责接受 CBO 输出的执行计划并对执行计划按效率进行排序。

针对给定的查询，CBO 在预备阶段利用 Hint 修改类查询模板中表连接的基数来得到多个不同的连接顺序（Join Order，JO）。为了筛选出适合当前查询的 JO 和 JH，在连接顺序

确定阶段，CBO 会将这些 JO 与多个预设的 JH 做组合，生成多个不同的执行计划，这些执行计划与 JO 和 JH 的组合一一对应，并交给学习排序单元处理。学习排序单元会将预测效率最高的执行计划对应的 JO 和 JH 的组合返回。被返回的 JO 和 JH 组合会在扫描类型选择阶段使用，并进一步和预设的多个 SH 分别结合形成多种新的组合。新生成的组合会在 CBO 中生成新的执行计划，这些新生成的执行计划会再次交给学习排序单元，选出最终的执行计划。

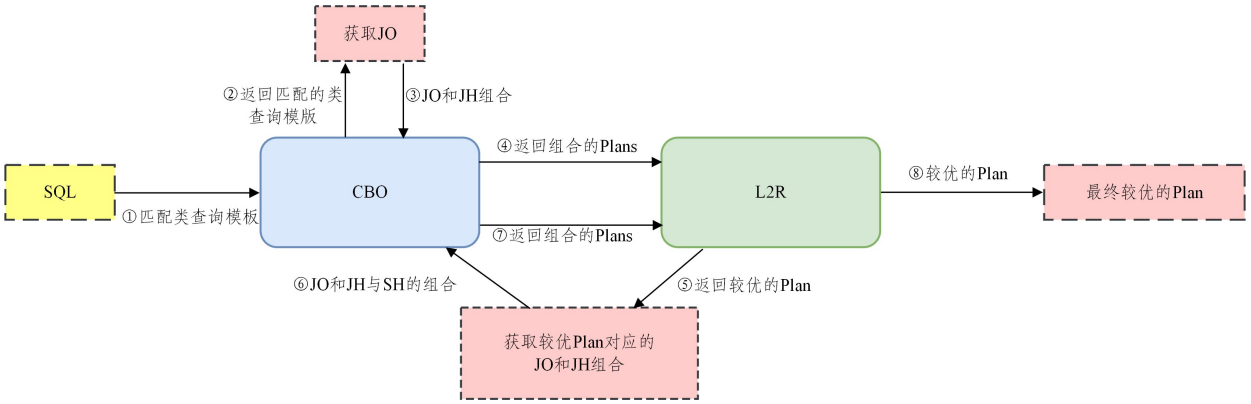


图 1　系统框架

Fig. 1　System framework

4　核心算法

本章重点介绍本文算法中各个阶段的细节。本文算法分为 3 个阶段：预备阶段、连接顺序确定阶段和扫描类型选择阶段。预备阶段主要负责获取目标查询的 JO。连接顺序确定阶段通过不同的 JH 和 JO 的组合获取多个不同的执行计划并将其用于 L2R 算法处理。扫描类型选择阶段主要将连接顺序确定阶段得到的最优组合与多种 SH 再组合，然后由 L2R 算法选出最终的执行计划。

4.1　预备阶段

预备阶段的任务是为目标查询生成备选的连接顺序，其核心是修改目标查询对应的类查询模板中连接基数的估计值，使 CBO 根据修改之后的结果生成其认为最优的连接顺序。

1）获取类查询模板

查询模板定义为一类查询，此类查询在结构上具备一致的特征，即所涉及的表连接相同，连接条件一致以及过滤条件中的属性相同，但过滤条件中的常量值可能有所不同[18]。本文在传统查询模板的基础上定义了一种类查询模板，即保留传统查询模板中的表连接信息，但去掉单表过滤条件。如图 2 所示，输入的目标查询 Q1 的类查询模板是 T1，可以看到 T1 中去掉了 Q1 中的单表条件，包括 a. e＞5、b. f in（‘teacher’，‘students’）和 c. g＝100。

Q1

```
SELECT *
FROM a, b, c
WHERE a.x = b.x AND a.y = c.y AND b.z = c.z
AND a.e > 5 AND b.f in ('teacher', 'students') AND c.g = 100;
```

T1

```
SELECT *
FROM a, b, c
WHERE a.x = b.x AND a.y = c.y AND b.z = c.z;
```

图 2　查询与类查询模板

Fig. 2　Query and similar query template

本文方法根据历史查询采集一个预置的类查询模板集合。对于一个目标查询，本文方法首先将在预置的类查询模板中为其寻找一个匹配的模板（连接表相同且连接条件相同）。如果目标查询是第一次被执行，则无法为它找到匹配的类查询模板，此时本文方法会根据目标查询抽取其所对应的类查询模板并将其加入预设集合中。

2）收集修正因子

收集修正因子的过程如算法 1 所示，根据类查询模板中每个等值连接条件 $cond_1$ 的两表构造多个两表连接查询：（1）以 $cond_1$ 为连接条件形成一个两表连接查询；（2）在两表连接查询基础上随机添加其中一张表的过滤谓词 $cond_2$，形成一个新的查询。这些构造出的查询组成 JSQL 集合，然后统计 JSQL 中查询的真实结果基数和 CBO 估计的基数，并计算真实基数与估计基数的比值 ratio。统计完所有比值后，在这些比值集合 Set_{ratio} 中采集 0.25 分位数、0.5 分位数和 0.75 分位数所对应的比值作为这一对可连接表的 3 种修正因子（$C_{0.25}$，$C_{0.5}$，$C_{0.75}$）。每一个类查询模板中的两表等值连接都会采集这 3 种修正因子。

算法 1　修正因子收集

输入：等值连接条件 EJ，过滤谓词条件 FC

输出：修正因子集 F

1. FOREACH $cond_1 \in$ EJ DO
2. 　$cond_2$＝RandomSample(FC)
3. 　JSQL＝MakeSQL($cond_1$，$cond_2$)
4. 　Set_{ratio}＝{}
5. 　FOREACH sql ∈JSQL
6. 　　ratio＝ExcuteSQL(sql)
7. 　　Add(Set_{ratio}，ratio)
8. 　$C_{0.25}$＝Sample(Set_{ratio}，0.25)
9. 　$C_{0.5}$＝Sample(Set_{ratio}，0.5)
10. 　$C_{0.75}$＝Sample(Set_{ratio}，0.75)
11. 　F[$cond_1$]＝{$C_{0.25}$，$C_{0.5}$，$C_{0.75}$}
12. return F

通过收集修正因子，可以确保两表等值连接的基数估计

误差被正确收集。在目标查询上应用这些修正因子的信息，可以确保对基数的修正是相对正确的，这对后续 CBO 生成连接顺序有一定方向性的指导。此外，这里暂时不考虑多表连接的修正因子收集，因为随着连接表数量的增加，多表连接的基数估计受过滤谓词的影响更大，并且多个连接表的组合会更多，其修正因子对其他查询不具备普适性，不适合用更改基数来探索合适的连接顺序。

3)收集连接顺序

首先，针对每一个类查询模板，本文将采集一组具有代表性的查询实例。这些代表性查询是通过调整同一模板中过滤谓词的常量值生成的，旨在确保能够充分反映该模板在不同条件下的行为特征。此组代表性查询将用于探索不同的连接顺序。

对于每个代表性查询(例如图 3 中的 Q2)，依据其类查询模板等值连接的具体情况，将在每一对表的 3 种修正因子中随机选取一种组成 Hint(C_{ab}, C_{ac}, C_{bc})，并将其应用于原查询上形成新的查询(图 3 中的 Q3)。新查询将被提交给 CBO 以获取执行计划，记录这些执行计划所对应的 JO，其将被视作对应模板的候选 JO。

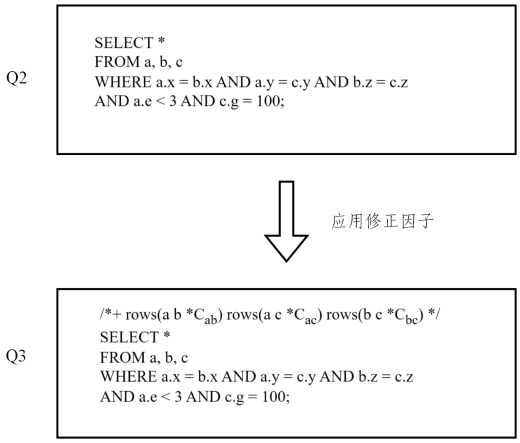


图 3 查询应用修正因子

Fig. 3 Query applied correction factors

在需要确定目标查询的连接顺序时，仅从其所属类查询模板的候选 JO 中进行选择。值得注意的是，上述类查询模板、修正因子以及候选连接顺序，都可以预先准备好，作为系统元数据存在。如果目标查询的类查询模板不在已有的模板集合中，则需要即时执行预备阶段，以补充新类查询模板的相关信息。

4.2 连接顺序确定阶段

连接顺序确定阶段的主要任务是从所属类查询模板对应的候选 JO 集合中挑选出最佳的 JO 和 JH 的组合。这里所指的 JH 是预先配置好的 Hint 集合，其涵盖了目标数据库系统中与连接类型相关的设置，例如 PostgreSQL 中的 enable_hashjoin，enable_nestloop 和 enable_mergejoin。本文预设了 4 种 JH 集合，如表 1 所列。其中 1 表示开启相应的连接类型，即允许使用该类型的连接；0 表示关闭相应的连接类型。

在复杂查询的执行计划中，可能会出现多种类型的连接节点。这是由于随着执行计划从最底层逐步向上构建，上层节点需要处理的数据量往往会减少，导致不同层次的连接节点采用的连接类型有所差异。此外，即使最优执行计划中只包含一种连接类型，在预设的 JH 集合下，如果 CBO 的基数估计误差在可接受范围内，这样的执行计划仍然可以在预设的 JH 下产生。因此，本文并未尝试枚举每一种可能的 JH 组合，而是基于上述预设的 4 种集合进行优化。这种方法不仅简化了优化过程，而且考虑到实际执行计划的动态性和多样性，能够有效地平衡优化效率与性能提升之间的关系，且能够在确保查询性能的同时，避免不必要的开销，使优化策略更加实用和高效。

该算法旨在通过从目标查询对应的类查询模板中提取所有候选连接顺序，并逐一搭配预设的 JH 集合，生成多种执行计划。然后利用学习排序单元 L2R 对这些执行计划进行排序，最终确定目标查询的最佳 JO 和 JH 组合。

表 1 Join Hint 设置

Table 1 Join Hint setting

Join Hint	[Hash Join, Nestloop, Merge Join]
Hint 1	[0,1,1]
Hint 2	[1,0,1]
Hint 3	[1,1,0]
Hint 4	[1,1,1]

4.3 扫描类型选择阶段

在 L2R 确定目标查询执行计划的 JO 和 JH 的组合后，扫描类型选择阶段将会探索最佳的 SH，同时确定目标查询最终的执行计划。

本阶段将使用数据库系统中与扫描类型相关的 Hint 组成多种预设的集合，例如针对 PostgreSQL，本文预设了 7 种 SH 集合，由 enable_seqscan，enable_indexscan 和 enable_indexonlyscan 这 3 种 Hint 组成。如表 2 所列，每个 SH 集合通过一个三位二进制数组表示，其中 0 表示关闭相应的扫描方式，1 表示开启。由于执行计划至少需要包含一种连接类型和扫描类型，因此全 0 的 Hint 集合不具备实际意义，未被纳入预设的 SH 集合。

表 2 Scan Hint 设置

Table 2 Scan Hint setting

Scan Hint	[SeqScan, IndexScan, IndexOnlyScan]
Hint 1	[1,0,0]
Hint 2	[0,1,0]
Hint 3	[0,0,1]
Hint 4	[1,1,0]
Hint 5	[1,0,1]
Hint 6	[0,1,1]
Hint 7	[1,1,1]

每一种 SH 集合都将与前一阶段由 L2R 模型筛选出的 JO 和 JH 组合，并交由 CBO 生成新的执行计划。随后，这些新生成的执行计划将再次提交给 L2R 模型进行评估，以确定最优的执行计划。最终，L2R 预测的最优执行计划将作为目标查询的最终执行计划。

4.4 L2R

受文献[19]的启发，本文选择将 TreeCNN[20] 作为 L2R 的组成部分。TreeCNN 的独特结构使其特别适用于执行计划这种树状结构的特征提取，相较于将树状结构转换为一维向量进行处理，TreeCNN 能够更有效地保留和利用树状结构的丰富信息。

Bao 运用 TreeCNN 来直接预测执行计划的确切运行时间。相比之下，Lero 并不致力于估算具体的执行时间，而是侧重于评估两个执行计划之间的相对效率。在处理多个执行计划时，Lero 通过一系列成对比较来确定最优方案，这一过程可能导致计算复杂度较高。为了改进这一流程，本文提出了一种增强架构，在 TreeCNN 的基础上引入了两个全连接层(Fully Connected Neural Networks，FCNN)。第一个全连接层旨在接收并处理所有执行计划的特征向量，进而计算出一个综合性的全局特征表示；第二个全连接层则专注于单个执行计划的特征提取，并将这些特征与全局特征相结合，从而实现多执行计划的同时比较。此方法能够一次性为每个执行计划提供一个介于 0～1 之间的评估分数，该分数是对每个执行计划的预期执行时间的评估，其中较高的数值代表更长的执行时间。图 4 给出了所提模型的整体框架。

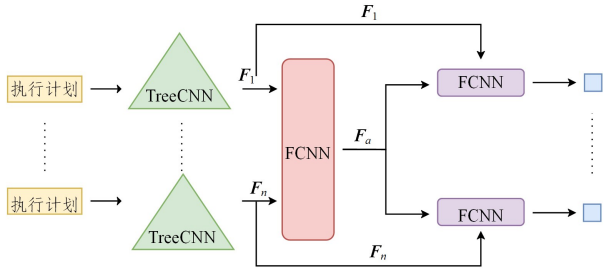


图 4　模型框架

Fig. 4　Framework of the proposed model

图 4 中，$\boldsymbol{F}_1$ 到 $\boldsymbol{F}_n$ 表示 n 个执行计划通过 TreeCNN 处理后得到的特征向量。所有的执行计划共享同一个 TreeCNN，即在处理各个执行计划时，TreeCNN 的权重是共享的。经过 TreeCNN 处理后的所有特征向量$\{\boldsymbol{F}_1,\boldsymbol{F}_2,\cdots,\boldsymbol{F}_n\}$被共同输入到第一个全连接层，生成一个综合特征向量 $\boldsymbol{F}_a$。特征向量 $\boldsymbol{F}_a$ 表征了输入的多个执行计划的整体特征，其会与每个单独的特征向量 $\boldsymbol{F}_i$(其中 $i=1,2,\cdots,n$)一起输入到第二个全连接层中进行进一步处理。类似于 TreeCNN，第二个全连接层也采用共享权重机制，以确保对所有执行计划的一致性处理。最终，当 $\boldsymbol{F}_a$ 与 n 个执行计划特征向量在第二个全连接层中处理完成后，每个计划的计算结果都会输出一个评估值作为执行计划执行效率的评估结果。

1)执行计划特征编码

鉴于执行计划本质上是具有层级结构的二叉树，因此首先对每个树节点进行编码。在执行计划树中，每个节点均包含由 CBO 提供的信息，如节点类型、估计基数和执行代价等关键信息。为了将这些信息转换为深度学习模型可处理的形式，本文设计了一种特征编码方案，该方案能够有效地捕捉并表示每个节点的特性。

为了对节点类型进行编码，采用 One-Hot 编码，将常见的连接节点和扫描节点类型转换为特征向量。具体而言，对于连接节点，区分了 Hash Join，NestLoop Join 和 Merge Join 几种常见类型；对于扫描节点，涵盖了 Seq Scan，Index Scan 和 Index Only Scan。这些节点类型分别对应独立的 One-Hot 编码向量，以确保模型能够准确识别和处理不同类型的节点。对于其他类型的节点，如聚合(Aggregation)、排序(Sort)、哈希(Hash)和物化(Materialization)等，将它们归为同一类别，并使用相同的编码向量表示。这种统一的处理方式简化了特征空间，同时保留了模型对主要节点类型的关注。此外，对 CBO 提供的估计基数(rows)和执行代价(cost)进行了对数变换处理。由于这两个属性在执行计划中可能存在较大的数值范围，直接使用原始数据可能会导致模型训练过程中的数值不稳定。通过对估计基数和执行代价取对数，可以有效压缩数值范围，减少极端值对模型的影响，从而提高模型的稳定性和预测准确性。

在完成执行计划节点的编码处理之后，整个执行计划的树状特征提取由 TreeCNN 根据各执行节点在二叉树中的分布位置进行。具体而言，TreeCNN 通过卷积操作分别对左子树节点、右子树节点和父节点进行特征提取，以全面捕捉执行计划树的结构信息和节点间的关系[21]。

2)训练集标签定义

在训练集中，执行计划的标签评估值 e 是根据一组执行计划的实际执行时间确定的。具体而言，训练集中执行计划的执行时间相对大小关系与评估值的相对大小关系保持一致。假设存在一组包含 n 个执行计划的数据集，每个执行计划的执行时间分别表示为 $T_1,T_2,\cdots,T_n$。每个执行计划的标签值 e_i 定义为其执行时间 T_i 与该组执行计划中所有执行时间总和的比值，如式(1)所示：

$$e_i=\frac{T_k}{\sum_{i=1}^{n}T_i} \tag{1}$$

其中，e_i 表示第 i 个执行计划的评估值，T_i 表示第 i 个执行的实际执行时间，$\sum_{i=1}^{n}T_i$ 表示该组执行计划中所有执行时间的总和。

3)损失函数定义

对于模型的损失函数，采用 MSE 计算损失函数值。对于一组包含 n 个执行计划评估值的预测值$\bar{y}_i$和真实值y_i，其损失函数的计算式如式(2)所示：

$$Loss_{\mathrm{MSE}}=\frac{1}{n}\sum_{i=1}^{n}(y_i-\bar{y}_i)^2 \tag{2}$$

4)训练模式

模型的训练集基于类查询模板生成。在给定的多个类查询模板中，每个模板随机抽取固定数量的查询实例，这些查询实例同时用于连接顺序确定阶段和扫描类型选择阶段。在连接顺序确定阶段，每个查询根据其所归属的类查询模板应用所有可能的 JO 和 JH 组合，生成一组执行计划，并实际执行这些计划以获取真实执行信息，形成训练样本；在扫描类型选择阶段，对同一组查询实例应用不同 SH，生成另一组执行计划并实际执行，记录其真实执行时间信息作为训练样本。值得注意的是，连接顺序确定阶段和扫描类型选择阶段使用的模型是相同的，因此训练过程由两个阶段生成的训练集混合而成，以确保模型能够同时学习到连接顺序和扫描类型的优化策略。

4.5　时间复杂度分析

1)预备阶段

预备阶段的时间复杂度与类查询模板本身的可连接表和

采样查询数目强相关。如果类查询模板为 n，采样的查询数目为 m，则会生成并执行 $n+m$ 个查询，那么预备阶段收集连接因子的时间复杂度为 $O(n)$，收集连接顺序的时间复杂度为 $O(m)$，预备阶段总时间复杂度为 $O(m+n)$。

2)连接顺序确定阶段

连接顺序确定阶段的时间复杂度与类查询模板对应的在预备阶段收集的连接顺序数目和 JH 数目强相关。如果类查询模板在预备阶段生成了 n 个连接顺序，并且 JH 的数目设置为 m，那么连接顺序确定阶段的时间复杂度为 $O(mn)$。

3)扫描类型选择阶段

扫描类型选择阶段的时间复杂度与 SH 高度相关。如果设置的 SH 数目为 n，那么扫描类型选择阶段的时间复杂度为 $O(n)$。

5 实验评估

为了与现有代表性工作进行比较，在 PostgreSQL 13.1[1] 上实现了本文算法，并借助了插件 pg_hint_plan[2] 在各个阶段中应用所需的 Hint。

5.1 实验设置

除了对比本文方法、Bao 和 Lero 的优化效果之外，本文还增加了一种基于 Full hint 的优化方法作为对比对象。实验主要研究了各种方法在查询优化策略方面的表现以及整体端到端的性能。

实验在一台 CentOS Linux 7(Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz+64 GB 内存+NVIDIA GeForce RTX 3090)的服务器上进行。

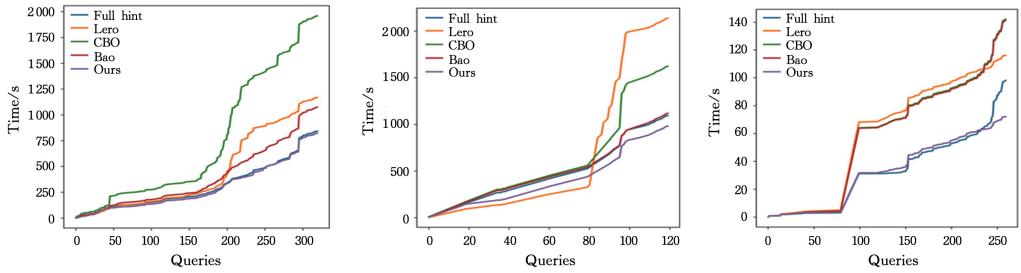
实验使用的数据集包括 IMDB[3]，TPC-H[4] 以及 STACK[5]。IMDB 是一个有 21 张有关电影主题的表的数据集，基于该数据集，本文实验选择了常用于基数估计测试的查询集 CEB-13k[6]；TPC-H 是一个 OLAP 的基准测试，有 8 张商业相关的表，本文实验中使用 5 GB 规模的 TPC-H 数据集和其自带的测试查询；STACK 是一个真实世界的数据集，包含了 2008－2019 这 10 年来自 StackExchange 网站的超过 1 800 万个问题和答案，本文实验选择 Bao 使用的测试查询用于实验。在 IMDB 数据集中，实验使用的测试查询集 CEB-13k 中包含 16 个类查询模板。在 TPC-H 数据集中，使用了所属于 Q3，Q5，Q7，Q8，Q9，Q10 的 6 个类查询模板(与 Lero 在 TPC-H 上的实验设置保持一致)。在 STACK 数据集中，选取了其中 13 个类查询模板(删除了 Lero 不支持的类查询模板以及默认执行时间很长的类查询模板)。

5.2 候选最佳执行计划对比实验

在该实验中，分别在 IMDB，TPC-H 和 STACK 选定的每个类查询模板中随机抽取 20 条查询，对比不同方法为这些查询生成候选计划中最佳候选计划的执行时间。

Lero 通过随机修改执行计划中连接节点的基数数量级来探索执行计划。因此，将 Lero 产生的所有执行计划保存并实际执行，选取执行耗时最短的作为 Lero 方法的最佳候选计划。Bao 通过设置不同 Hint 来产生不同的执行计划，因此实际执行所有产生的执行计划并同样选取执行耗时最短的作为 Bao 方法的最佳候选计划。由于 Bao 的源码中仅提供了 5 种 Hint，为了进一步探索利用 Hint 生成候选计划的能力，枚举了每一种 Hint 组合(3 种 Join Hint 和 3 种 Scan Hint 进行组合)，最终可得 64 种 Hint 组合，本文将这种方法称为 Full hint。在实验中，采用与 Lero，Bao 类似的方式执行 Full hint 产生的全部候选计划，选择其中执行耗时最短的作为 Full hint 方法的最佳候选计划。由于本文方法由多个阶段组成，为了统计最佳候选计划，默认 L2R 每次都能选择出最佳的执行计划，即在连接顺序确定阶段，首先实际执行每一个执行计划，选择出耗时最短的执行计划对应的 JO 和 JH 的集合作为该阶段的输出，然后将扫描类型选择阶段生成的所有执行计划作为候选执行计划，采用和对比方法相同的方法选择实际执行耗时最短的执行计划作为本文方法的最佳执行计划。此外，本实验还将 PostgreSQL 的原生 CBO 优化器产生的执行计划加入比较(图 5 中标记为 CBO)。

图 5 中给出了每种方法的查询累计执行时间，即累计每一条查询的执行计划的执行时间。



(a)IMDB (b)TPC-H 5x (c)STACK

图 5 不同数据集上最佳执行计划的累计时间

Fig. 5 Cumulative time for the best execution plan on different datasets

[1] PostgreSQL[OL]. http://www.postgresql.org/.

[2] pg_hint_plan[OL]. https://github.com/ossc-db/pg_hint_plan.

[3] IMDB[OL]. http://homepages.cwi.nl/~boncz/job/imdb.tgz

[4] TPC-H[OL]. https://www.tpc.org/tpch/.

[5] STACK[OL]. https://rm.cab/stack.

[6] CEB13k[OL]. https://github.com/RyanMarcus/imdb_pg_dataset/tree/master.

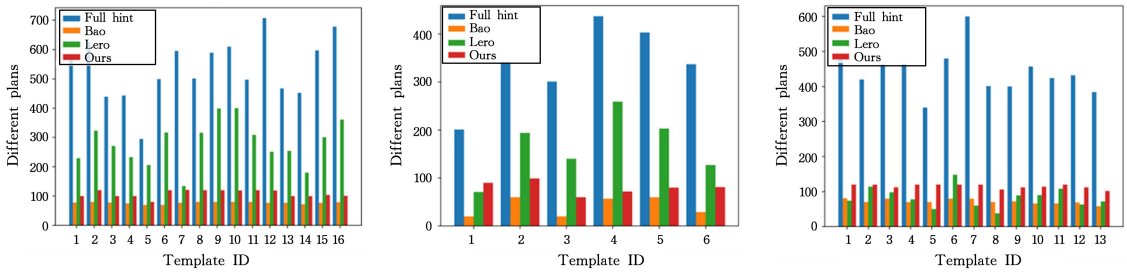
从图中可知，随着查询数量的增加，本文方法探索到的最佳执行计划在累计执行时间方面总体优于对比方法。在 IMDB 上，Full hint 方法的累计执行时间虽然与本文方法基本持平，但 Full hint 是通过枚举每一种 Join Hint 和 Scan Hint 集合得到的，它需要产生大量的候选执行计划（64 个），而本文方法通过分阶段逐步选择出合适的 JO，Join Hint 和 Scan Hint，相比于 Full hint 减少了很多候选执行计划。此外，对比 Bao 和 Full hint 可以发现，由于 Full hint 的 Hint 组合远多于 Bao 的 5 种 Hint 集合，因此 Full hint 确实能够探索到更多更好的执行计划，但这同样是以产生大量候选计划为代价。对于 TPC-H 上的前 80 个查询，Lero 产生的累计执行时间更短，但在 81～100 个查询这个区间，Lero 出现了明显的劣化现象。通过对比 Lero 和 CBO 在该区间内的曲线斜率可以发现，相比于 CBO 产生的默认计划，Lero 得到的最佳计划的表现更差，侧面说明了 Lero 这种随机修改基数数量级的方法并不稳定，它虽然确实可能找到更好的计划，但也可能探索到大量糟糕的执行计划。

5.3　候选执行计划质量对比实验

本节实验通过统计各对比方法生成的候选执行计划的相关信息，进一步对比候选执行计划的质量，以评估各种方法。该部分实验仍然从 IMDB，TPC-H 以及 STACK 中选择的每个类查询模板中随机抽取 20 个查询。对比的候选计划选择方式与 5.2 节相同。

1）不同的候选计划数

从图 6 可知，首先，Full hint 在每一种数据集下都能够产生大量的候选计划，可探索的计划空间远大于其他方法。Hint 集合越多，就能产生更多的候选计划。其次，Lero 能够探索到的计划空间大小仅次于 Full hint，表明修改计划基数数量级的方法能够有效地探索计划空间。而本文方法和 Bao 能够探索到的计划空间较小，这与这两种方法可枚举的 Hint 信合较少有关系：Bao 只选取了 5 种 Hint 组合，本文方法则选取了 8 种 Scan Hint 集合。



（a）IMDB　（b）TPC-H 5x　（c）STACK

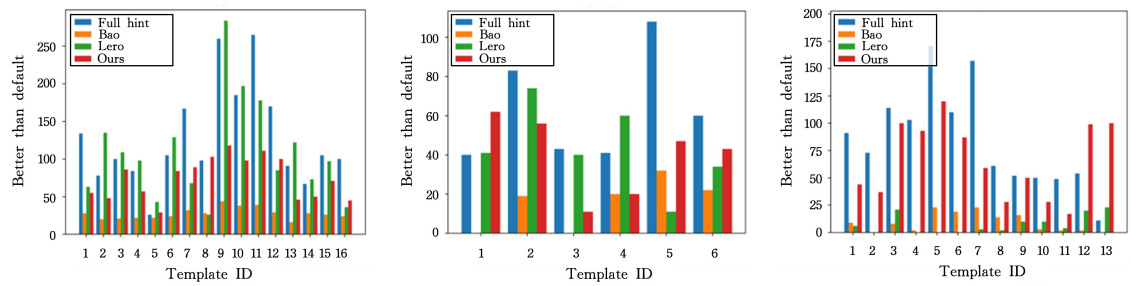
图 6　3 个数据集上各方法的不同候选计划数目

Fig. 6　Number of different candidate plans for different methods on three datasets

3）有效计划数

不管是本文方法，还是 Lero，Bao 等对比方法，其目标都是产生比现有优化器更好的执行计划。为了评估各方法超越现有优化器技术的程度，引入“有效计划”的概念：对于同一个查询，如果某种方法产生的一个候选执行计划的实际执行时间比 PostgreSQL 默认 CBO 优化器产生的执行计划的时间更短，则将这个候选执行计划归为“有效计划”。

在图 7 中，由于 Full hint 能产生大量的候选执行计划，其中有效计划数自然也较多。本文方法虽然不能产生大量候选执行计划，但是其中的有效计划数和 Lero 没有太大的差距。对于 TPC-H 数据集上的一些类查询模板（T1，T5，T6），本文方法产生的有效计划数甚至比 Lero 更多。在 STACK 上，本文方法探索到的有效计划数明显多于 Lero，甚至基本接近 Full hint 探索到的有效计划数。



（a）IMDB　（b）TPC-H 5x　（c）STACK

图 7　3 个数据集上各方法的有效计划数目

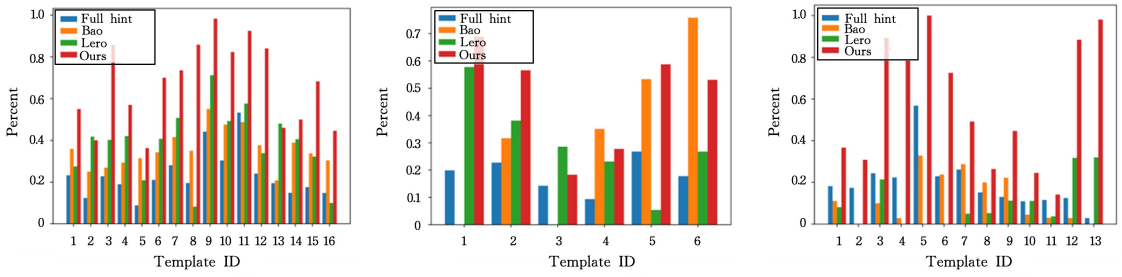
Fig. 7　Number of effective plans generated for different methods on three datasets

4）有效计划占比

从图 8 可知，在 3 个数据集上，本文方法探索到的有效计划在所有候选计划中的占比都明显优于其他方法。Full hint 和 Lero 虽然能够探索到大量候选计划，但其中的有效计划数量并不多，甚至含有大量劣化严重的执行计划。这也间接证明本文方法采用的分阶段方式能够更有效地逼近最优执行计划，从而改善 CBO 的执行计划生成。

结合图 6－图 8 可知，本文方法通过探索少量的候选计划就能够得到较多的有效计划，表明本文方法的分阶段处理能够指引 CBO 逐步生成一些高质量计划，而不是和其他方法

一样没有目的性地尝试产生大量候选计划，以期碰到较优计划。此外，结合候选最佳执行计划对比实验的结果可知，本文方法不论是在最佳执行时间还是有效计划的占比上都优于 Lero 和 Bao 等方法。



(a)IMDB (b)TPC-H 5x (c)STACK

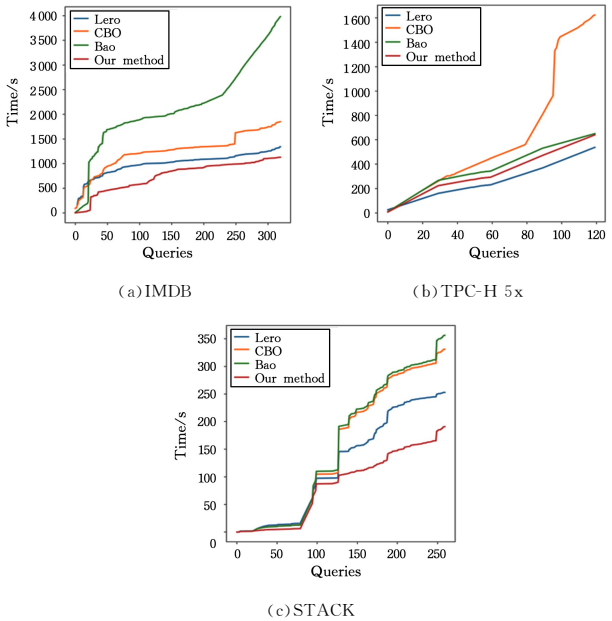
图 8 3 个数据集上各方法的有效计划占比

Fig. 8 Proportion of effective plans for different methods on three datasets

5.4 端到端查询优化对比实验

该实验对比各方法的端到端执行时间，即从查询语句进入数据库系统到数据库系统返回查询结果所需的总时间，其中包括候选计划生成耗时、模型预测耗时以及计划执行耗时等多个组成部分。在本实验中，模型训练采用的训练集由每个类查询模板中随机抽取的 10 条查询组成，并采用预训练的方式训练好模型，实验过程中不再根据查询执行反馈做再训练。测试查询则由每个类查询模板中随机抽取的与训练集不同的 20 条查询组成。

如图 9 所示，在 IMDB 和 STACK 上，本文方法的端到端执行时间优于其他方法，但在 TPC-H 上比 Lero 差，这是因为本文方法找到的最终执行计划虽然也是有效计划，但是其质量比 Lero 找到的最终执行计划差。



(a)IMDB (b)TPC-H 5x

(c)STACK

图 9 不同方法的端到端查询执行累计时间对比

Fig. 9 Comparison of cumulative execution time of end-to-end queries using different methods

在候选最佳执行计划对比实验中，Lero 在前 80 个查询上都优于其他方法，说明 Lero 在 TPC-H 的某些特定类查询模板上表现较优。此外，Lero 在第 81－100 个查询的查询区间上出现了性能劣化的情况，但是在本实验中并没有出现此情况，其原因主要是 Lero 将 PostgreSQL 默认 CBO 产生的计划也用于模型比较，Lero 的模型认为 PostgreSQL 产生的计划优于其自身的候选计划，故而采用了前者实际执行来避免劣化的产生，使得端到端执行时间有所缩短。在 IMDB 和 STACK 上可以看到，Bao 的端到端执行时间高于 PostgreSQL 的默认 CBO。这可能是由于 IMDB 和 STACK 相比于 TPC-H 是更复杂且更接近真实场景的数据集，表的数目与属性类型也更为复杂，导致了 Bao 的模型在训练时无法很好地收敛，进而在模型预测执行计划时出现预测错误，使得模型选择了糟糕的执行计划。此外，Bao 的错误预测也可能与训练集过大有关，过大的训练集使 Bao 无法捕获较优执行计划的特征。

综上，虽然在 TPC-H 上，本文方法在端到端执行时间上略差于 Lero，但在更复杂的 IMDB 和 STACK 数据集上，本文方法优于其他方法。

结束语 本文借助排序学习对数据库查询优化的策略进行了改进。考虑到查询优化器存在基数估计不准的问题，采用分阶段的策略逐步探索较优的执行计划：预备阶段通过修改两表的基数探索较优连接顺序；连接顺序确定阶段利用 Join Hint 确定较优的连接顺序和连接类型；扫描类型选择阶段采用多种 Scan Hint 的集合得到一批较优的候选执行计划。此外，提出了一种 L2R 的机器学习方法，将候选执行计划评估的问题转化为多个执行计划执行效率排序的问题，实现了对候选执行计划的高效选择。实验结果表明，相对于 Lero 和 Bao 等现有最优方法，本文算法在复杂的数据集上能获得更好的候选计划质量以及最优执行计划。

参 考 文 献

[1] LEIS V, GUBICHEV A, MIRCHEV A, et al. How Good Are Query Optimizers, Really? [J]. Proceedings of the VLDB Endowment, 2015, 9(3): 204-215.

[2] WU P, CONG G. A Unified Deep Model of Learning from both Data and Queries for Cardinality Estimation[C]// International Conference on Management of Data, Virtual Event (SIGMOD '21). ACM, 2021: 2009-2022.

[3] WU Z, SHAIKHHA A. BayesCard: A Unified Bayesian Framework for Cardinality Estimation[J]. arXiv: 2012.14743, 2020.

[4] YANG Z, KAMSETTY A, LUAN S, et al. NeuroCard: One Car-

dinality Estimator for All Tables[J]. Proceedings of the VLDB Endowment，2020，14(1)：61-73.

[5] YANG Z，LIANG E，KAMSETTY A，et al. Deep Unsupervised Cardinality Estimation[J]. Proceedigns of the VLDB Endowment，2019，13(3)：279-292.

[6] ZHANG J，ZHANG C，LI G，et al. AutoCE：An Accurate and Efficient Model Advisor for Learned Cardinality Estimation[C]// 39th IEEE International Conference on Data Engineering (ICDE 2023). IEEE，2023：2621-2633.

[7] ZHU R，WU Z，HAN Y，et al. FLAT：Fast，Lightweight and Accurate Method for Cardinality Estimation[J]. Proceedings VLDB Endowment，2021，14(9)：1489-1502.

[8] MARCUS R，NEGI P，MAO H，et al. Bao：Making Learned Query Optimization Practical[C]// SIGMOD. ACM，2021：1275-1288.

[9] ZHU R，CHEN W，DING B，et al. LERO：A Learning-to-Rank Query Optimizer[J]. Proceedings of the VLDB Endowment，2023，16(6)：1466-1479.

[10] CHEN X，CHEN H，LIANG Z，et al. LEON：A new framework for ml-aided query optimization[J]. Proceedings VLDB Endowment，2023，16：2261-2273.

[11] LAN H，BAO Z，PENG Y. A Survey on Advancing the DBMS Query Optimizer：Cardinality Estimation，Cost Model，and Plan Enumeration[J]. Data Sci. Eng.，2021，6(1)：86-101.

[12] OLKEN F，ROTEM D. Random Sampling from Database Files：A Survey[C]// Statistical and Scientific Database Management，5th International Conference SSDBM，Charlotte，NC (Lecture Notes in Computer Science，Vol. 420). Springer，1990：92-111.

[13] CORMODE G，GAROFALAKIS M N，HAAS P J，et al. Synopses for Massive Data：Samples，Histograms，Wavelets，Sketches[J]. Found. Trends Databases，2012，4(1/2/33)：1-294.

[14] HASAN S，THIRUMURUGANATHAN S，AUGUSTINE J，et al. Deep Learning Models for Selectivity Estimation of Multi-Attribute Queries[C]// Proceedings of the 2020 International Conference on Management of Data，SIGMOD Conference 2020. ACM，2020：1035-1050.

[15] HILPRECHT B，SCHMIDT A，KULESSA M，et al. DeepDB：Learn from Data，not from Queries! [J]. Proceedings of the VLDB Endowment，2020，13(7)：992-1005.

[16] KIPF A，KIPF T，RADKE B，et al. Learned Cardinalities：Estimating Correlated Joins with Deep Learning[J]. arXiv：1809.00677，2018.

[17] SUN J，LI G，TANG N. Learned Cardinality Estimation for Similarity Queries[C]// International Conference on Management of Data(SIGMOD '21). ACM，2021：1745-1757.

[18] ILYAS I F，BESKALES G，SOLIMAN M A. Probabilistic query optimization in database systems[J]. ACM SIGMOD Record，2008，37(3)：4-11.

[19] ROY D，PANDA P，ROY K. Tree-CNN：A Deep Convolutional Neural Network for Lifelong Learning[J]. arXiv：1802.05800，2018.

[20] MOU L，LI G，ZHANG L，et al. Convolutional Neural Networks over Tree Structures for Programming Language Processing[C]// AAAI. AAAI Press，2016：1287-1293.

[21] MARCUS R，PAPAEMMANOUIL O. Plan-Structured Deep Neural Network Models for Query Performance Prediction[J]. Proceedings VLDB Endowment，2019，12(11)：1733-1746.

YU Yang，born in 2000，postgraduate. His main research interests include database，query optimization，and AI4DB.

PENG Yuwei，born in 1980，Ph.D，associate professor. His main research interests include database systems，big data，and digital watermarking.

（责任编辑：喻藜）