中国科学 信息科学
SCIENTIA SINICA Informationis

# RLO: a reinforcement learning-based method for join optimization

张心怡, 张智鹏, 张铁赢, 崔斌 and 范举

Citation: 中国科学: 信息科学 **50**, 637 (2020); doi: 10.1360/SSI-2019-0179

View online: http://engine.scichina.com/doi/10.1360/SSI-2019-0179

View Table of Contents: http://engine.scichina.com/publisher/scp/journal/SSI/50/5

Published by the 《中国科学》杂志社

中国科学: 信息科学 2020 年 第 50 卷 第 5 期: 637–648
SCIENTIA SINICA Informationis
论文



# RLO: 一个基于强化学习的连接优化方法

张心怡[1], 张智鹏[1], 张铁赢[2], 崔斌[1*], 范举[3]

1. 北京大学计算机科学与技术系高可信软件技术教育部重点实验室, 北京 100871
2. 阿里巴巴集团达摩院数据库与存储实验室, 北京 100029
3. 中国人民大学信息学院, 北京 100872
* 通信作者. E-mail: bin.cui@pku.edu.cn

收稿日期: 2019–08–25; 接受日期: 2019–11–06; 网络出版日期: 2020–04–27

国家自然科学基金 (批准号: 61832001, 61702016, 61572039) 资助项目

**摘要** 连接优化是数据库领域最重要的研究问题之一. 传统的连接优化方法一般应用基础启发式规则, 他们通常搜索代价很高, 并且很难发现最优的执行计划. 主要原因有两个: (1) 这些基于规则的优化方法只能探索解空间的一个子集, (2) 他们没有利用历史信息, 不能够很好地衡量执行计划的代价, 经常重复选择相同的糟糕计划. 为了解决以上两个问题, 我们提出 RLO (reinforcement learning optimization), 一个基于强化学习的连接优化方法. 我们将连接优化问题建模成马尔可夫 (Markov) 决策过程, 并且使用深度 $Q$- 学习来估计每一种可能的执行计划的执行代价. 为了进一步增强 RLO 的有效性, 我们提出了基于树形结构的嵌入方法和集束搜索策略来尽量避免错过最好的执行计划. 我们在 Apache Calcite 和 Postgres 上实现了 RLO. 实验表明: (1) 在 Apache Calcite 上, 与一系列剪枝的启发式算法相比, RLO 搜索计划的效率为它们的 10~56 倍, 并且生成的计划能更快地执行 (80% 的加速); (2) 与原生的 Postgres 相比, RLO 搜索计划的效率是其 14 倍, 并且在端到端的执行中达到 12.9% 的加速.

**关键词** 连接优化, 强化学习, 嵌入方法, 集束搜索

## 1 引言

连接优化 (join optimization) 是数据库领域最重要的研究问题之一. 给一个特定查询, 连接优化任务是确定关系的连接顺序, 并将其转化为物理计划, 使得查询任务能够高效执行. 它是关系型数据库查询优化的关键问题 [1], 同时也是 NP-hard 问题 [2].

大数据时代下, 数据仓库的联机分析处理 (online analytical processing, OLAP) 中经常会碰到多个关系进行连接的复杂查询. 而高效处理多表连接的能力 (避免响应时间过长), 对用户的使用体验至关

**引用格式:** 张心怡, 张智鹏, 张铁赢, 等. RLO: 一个基于强化学习的连接优化方法. 中国科学: 信息科学, 2020, 50: 637–648, doi: 10.1360/SSI-2019-0179
Zhang X Y, Zhang Z P, Zhang T Y, et al. RLO: a reinforcement learning-based method for join optimization (in Chinese). Sci Sin Inform, 2020, 50: 637–648, doi: 10.1360/SSI-2019-0179

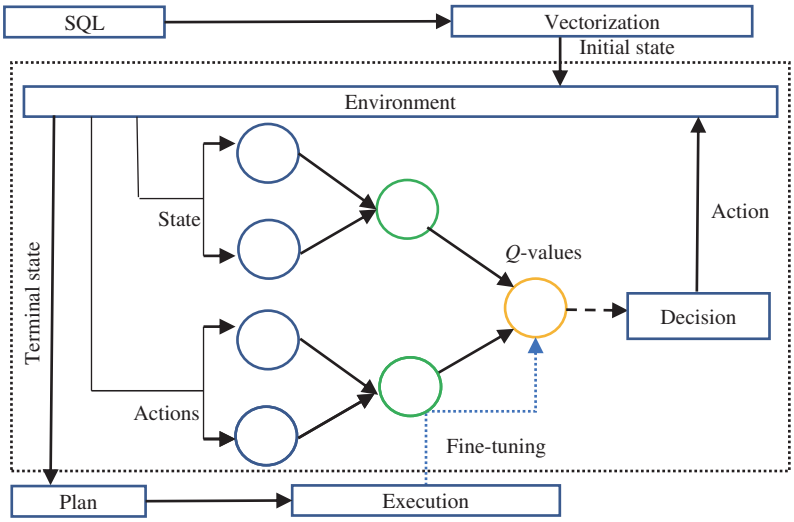



**图 1　(网络版彩图) RLO 优化器框架**

**Figure 1**　(Color online) Overview for RLO

重要. 经典的 Selinger 优化器 [3] 采用动态规划查找具有最小代价的查询计划, 然而其详尽的列举需要巨大的空间和时间复杂度, 因此现有的优化器大多采用启发式方法来减小搜索空间. 第 1 类方法是通过剪枝方法来限制搜索空间. 常见的剪枝方法包括左深树剪枝、右深树剪枝、之字形树剪枝等. 然而在数据库不同的代价模型下, 这类限制在某一子搜索空间的方法难以保证生成的查询计划的质量, 且极易错过最佳计划 [4]. 同时, 剪枝方法的时间复杂度为指数级, 随着关系数量增多, 优化时间依然迅速增长. 第 2 类方法是随机性更强的算法 (遗传算法、QuickPick 等 [5]), 这类算法适用于关系数量较多的查询. 他们的基本思想是随机地产生连接决策, 并依据代价进行更新, 因此产生的执行计划的质量难以得到保证.

这两类基于启发式的方法都依赖于优化器的代价估计, 而不接受实际运行的反馈. 因此优化器的代价模型很可能偏离实际运行时间, 造成计划不佳, 并重复选择不好的计划. 总结来说, 现有查询优化机制存在以下挑战:

(1) 由启发式算法组成. 这些启发式算法或是以固定的方法剪枝, 造成错过最优计划, 且时间复杂度较高; 或是采用随机的方式, 缺乏对计划质量的保证.

(2) 执行结果的反馈不会被优化器利用. 优化器难以从错误中学习总结, 会重复做出不好的计划.

针对以上问题, 本文提出 RLO (reinforcement learning optimization), 一个基于深度强化学习 [6] 的优化算法. RLO 以数据为导向进行搜索空间的剪枝, 并且能够从执行结果的反馈中学习. 具体来说, RLO 把连接优化问题建模成一个马尔可夫 (Markov) 决策过程. 我们将待连接的关系视为状态, 把连接某两个关系的决策视为动作, 然后利用深度 $Q$- 学习 [7], 计算每个动作的 $Q$ 值. 在决策的过程中, 我们依次选择 $Q$ 值低的动作, 最终生成连接优化的执行方案. RLO 的执行逻辑参见图 1.

为了验证此方案, 我们在 Postgres 和 Apache Calcite [8] 上分别实现了 RLO, 并且在开放数据集 Join Order Benchmark (JOB) [4] 上进行了充分的实验. 实现结果表明: (1) 在 Apache Calcite 上, 与一系列剪枝的启发式算法相比, RLO 搜索计划的效率为它们的 10~56 倍, 并且在执行代价的实验中性能提升 80%; (2) 与原生的 Postgres 相比, RLO 搜索计划的效率是其 14 倍, 并且在端到端的执行中速度提升 12.9%. 总结来说, 本文作出了如下贡献:

• 提出一个基于深度强化学习的连接优化算法 RLO, 使得系统能够以数据为导向来剪枝搜索空间, 从执行计划的反馈中学习.

• 为了增强 RLO 的有效性, 本文进一步提出一个基于树形结构的嵌入方法和集束搜索策略来尽量避免错过最好的执行计划.

• 本文分别基于 Postgres 和 Apache Calcite 实现了 RLO 方法. 实验表明了 RLO 能够更快更好地选择连接优化的执行方案.

## 2 相关工作

本节主要介绍连接优化和强化学习两个部分的相关工作.

### 2.1 连接优化

连接优化指的是数据库领域中, 给定两个或者多个关系表的连接查询, 确定可行且高效的逻辑计划与物理计划的过程. 其中, 逻辑计划指关系之间的连接顺序. 物理计划指在逻辑计划的基础上, 确定实现连接的方法 (如哈希连接、索引连接). 为了从巨大的计划搜索空间中, 高效地选择合适的计划, 一系列相关工作被提出. 我们首先介绍 Selinger 优化器, 接下来分别介绍基于启发式规则和基于学习的方法.

#### 2.1.1 Selinger 优化器

Selinger 优化器 [3] 利用动态规划解决连接优化问题: 先从计划树的最底层, 找到每个子问题的最优解. 子问题的最优解进行组合, 形成上层问题的最优解, 自底向上层层查找, 最终找到根问题的最优解. DB2 和 Postgres 都采用 Selinger 优化器作为基本结构. 然而, 其计划的搜索空间复杂度为 $n!C(n-1)$, $C(n)$ 为卡特兰 (Catalan) 数. 可见, 随着关系数量增多, 搜索空间爆炸式增长.

#### 2.1.2 启发式方法

为了减小 Selinger 优化方法的搜索空间, 研究人员提出了很多剪枝策略, 比如左深树剪枝、右深树剪枝与之字形树剪枝 (zig-zag [9]) 等. 这类方法通过限制解空间的形式达到剪枝的目的. 因此它们难以适应所有的代价模型. 当代价非线性时 (如关系大小超过内存限制), 此种固定的剪枝方式不再适用 [4], 造成错过潜在的最优计划. 同时限定搜索空间的动态规划算法, 也会导致复杂度与涉及关系数目成指数关系.

为了降低时间复杂度, 研究人员进一步提出了贪心算法、随机算法等. 贪心算法的中心思想是在树的每一级保持元组数目尽可能少的中间关系, 而随机算法包括遗传算法 [10]、QuickPick [5] 等. 它们随机地产生连接决策, 并依据代价进行更新.

#### 2.1.3 基于学习的方法

近年来, 研究人员也提出了基于学习的连接优化方法. Kipf 等 [11] 用 DNN 进行基数估计. Ortiz 等 [12] 通过预测中间关系的基数得到查询语句子计划的表示, 并应用到基于强化学习的连接优化上, 但是其奖励函数是中间关系的基数的倒数, 故而局限于优化逻辑计划, 无法优化物理计划. Marcus 等 [13] 使用强化学习制定查询计划, 然而其采用基于策略 (policy-based) 的方法, 收敛速度很慢. 随后基于价值 (value-based) 的 DQ [14] 被提出, 其训练速度、计划质量优于 Rejoin, 然而其用简单的 one-hot 编码 (查询涉及的列用 1 标出) 进行状态表征, 忽视了子计划的树形层次、中间关系大小, 并且单一的动

作选择策略可能导致错失最优计划. 而本文提出的 RLO 算法采用基于代价、运行时间的奖励, 可进行物理优化; 采用可扩展的树结构表征, 增强网络学习能力; 应用集束搜索提高计划质量, 实现了运行速度的进一步提升.

### 2.2 深度 $Q$- 学习

深度 $Q$- 学习 (deep $Q$-learning) [7] 属于基于价值 (value-based) 的强化学习算法, 它将决策与优化目标的改进 ($Q$ 值) 相关联. 深度 $Q$- 学习用神经网络来预测 $Q$ 值, 并通过不断更新神经网络从而学习到最优的行动路径. 在应用时, 智能体感知环境状态, 根据 $Q$ 值采取某一个动作作用于环境, 环境接受该动作后状态发生变化, 智能体根据当前状态再选择下一个动作, 直至问题结束.

## 3 RLO: 基于强化学习的连接优化器

### 3.1 问题定义

连接优化可以看作一个顺次做出连接关系决策的马尔可夫决策过程 (Markov decision process, MDP) [14]. 在 MDP 模型中, 智能体做出一系列动作 (记作 $a_i$), 目的是最大 (小) 化最终的目标值. 每个动作都取决于当前状态 (记作 $s_i$) 并转移到一个新状态 $s_{i+1}$. 在连接优化问题中, 我们将状态 $s$ 对应到连接进度, 动作 $a$ 对应到连接某两个关系的决策, 那么优化目标就是使得连接操作的总代价最小. 形式化的定义如下.

**定义1** (连接优化) 假设 $J$ 表示代价模型, $T$ 代表查询涉及的关系数, 那么连接优化在于找到一个执行动作序列 $a_1, a_2, \ldots, a_{T-1}$, 使总代价最小:

$$\min_{a_1,a_2,\ldots,a_{T-1}} \sum_{i=1}^{T-1} J(a_i), \tag{1}$$

$$\text{s.t. } s_{i+1} = a_i(s_i),$$

其中 $J(a_i)$ 是对动作 $a_i$ 的代价估算, 由数据库已有的代价模型进行估算 (如在 Postgres 中, 代价为 CPU 开销和 IO 开销的加权和). $a_i(s_i)$ 表示动作 $a_i$ 作用于状态 $s_i$ 得到后续状态 $s_{i+1}$.

### 3.2 RLO 算法框架

通过定义一, 我们将连接优化形式化地表示为一个马尔可夫决策过程. 基于此, 我们提出 RLO, 一个基于深度 $Q$- 学习的连接优化方法.

图 1 展示了 RLO 的算法流程. 给定一个 SQL 查询, RLO 的执行步骤如下:

(1) RLO 首先把输入的查询向量化为一个初始状态 $s_0$. $s_0$ 和此状态下的可能动作被同时传给神经网络.

(2) 将当前的状态和可能的动作作为输入, 通过神经网络计算得到对应的 $Q$ 值 (动作的长远成本).

(3) 智能体采用 $Q$ 值最低的一个或一组动作, 作用于环境, 得到一个新的状态.

(4) 在此状态下, 智能体继续选择动作, 并且得到一个新的状态, 重复步骤 (2) 直至达到最终过程.

(5) 最后优化器按传统方法 [3] 添加 Agg, Group 等计划节点, 便得到最终的物理计划, 交由执行器执行, 并返回查询结果.

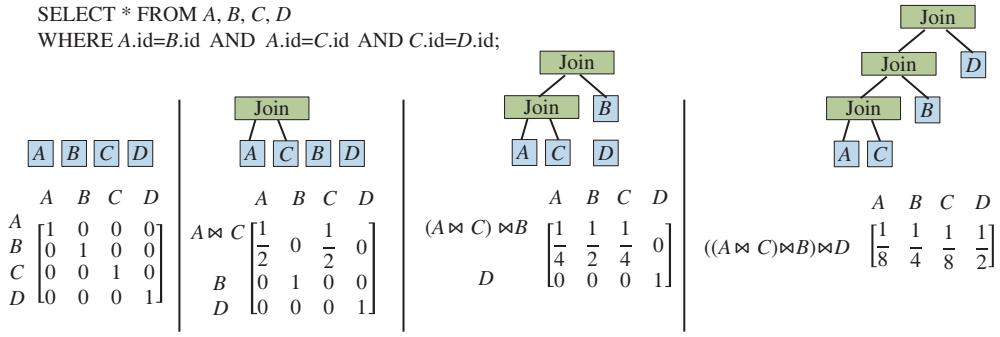



**图 2 (网络版彩图) 基于树结构的嵌入方法**

**Figure 2** (Color online) Tree-based embedding method for states

在这个过程中, 我们可以收集实际的执行数据, 对神经网络进行微调 (fine-tuning) [15], 实现从反馈中学习. 接下来, 我们按照以上的执行步骤依次介绍: 状态与动作的表征 (3.3 小节), $Q$ 值的计算 (3.4 小节), 搜索新动作 (3.5 小节) 以及网络微调 (3.6 小节).

### 3.3 状态与动作的表征

我们需要对模型中的每个状态进行向量化表示, 以便于神经网络的训练. 为了对状态中不同子计划进行区分, 我们采用一种基于树形结构的状态嵌入方法: 我们采用 $n$ 维行向量 $x$ 来编码每个子树, 其中 $n$ 是数据库中存在的关系数. 对于某一子树, 如果第 $i$ 个关系不在子树中, 则 $x_i$ 为 0; 反之, 则 $x_i$ 为 $\frac{1}{h(i,x)^2}$, $h(i,x)$ 为第 $i$ 个关系在子树中的高度.

图 2 展示了一个 SQL 查询在不同状态下的向量化表示的变化. 例如第 3 个状态的第 1 行对应着子树 $(A \bowtie C) \bowtie B$, 其第 1 行第 1 列值为 1/4, 是因为其关系 $A$ 在子树中的高度为 2; 其第 4 列为 0, 是因为关系 $D$ 不在子树中. 与 DQ [10] 相比, 此种表征方法嵌入了已有连接结构, 且具有较好的扩展性, 子树的表示可以进一步与基数估计值、选择性 (selectivity) 进行拼接.

另外, 我们需要对连接工作进行嵌入式表达. 连接动作的基本表征包含左右关系、物理操作、中间关系大小, 即 $f_a = L \oplus R \oplus ph \oplus c$. 其中左右关系 $L$ 与 $R$ 分别采用上文所述的 $n$ 维行向量表示. $ph$ 为物理操作, 我们采用 one-hot 编码. $c$ 为对连接产生的中间关系基数的估计, 可利用传统数据库的估计方法或基于学习的方法 [11] 得到.

### 3.4 $Q$ 值的计算

本文采用深度 $Q$- 学习 [7] 从样本数据中学习 $Q$ 值函数. 我们把执行连接动作的长远成本看作 $Q$ 值. 其定义如下:

$$Q(s,a) = J(a) + \min_{a'} Q(s',a'), \tag{2}$$

其中 $s'$ 是在状态 $s$ 下采用动作 $a$ 后得到的后续状态, $J(a)$ 是对动作 $a$ 的代价估算, 从数据库已有的代价模型获得, $a'$ 是实施动作 $a$ 后的下一步动作. 其中, 状态和动作采用的是 3.3 小节中的表征方法. $Q$ 值是后续决策选择最优动作 (连接) 的累积代价和. $Q$ 值越小, 该动作被选择的可能性越大. 每次都选择 $Q$ 值最小的动作, 则可实现优化目标即总代价最小 (参见 3.1 小节).

传统优化器的查询优化是一个由子问题组合解决父问题的动态规划过程. 我们可以利用传统优化器自下而上的规划过程, 获得 $\langle s, a, J(a), s' \rangle$ 的决策序列, 并递归得到在某一状态下采用的动作与 $Q$

值的对应关系. 我们建立一个参数模型 $Q_\theta$ 去模拟 $Q$ 值函数:

$$Q_\theta(f_s, f_a) = Q(s, a), \tag{3}$$

其中 $f_s$ 是对状态的向量化表示, $f_a$ 是对连接动作的向量化表示, $\theta$ 是模型参数. 对于每一个训练样本, 我们可以得到其 $Q$ 值的估计值 $q$:

$$q = Q_\theta(f_s, f_a). \tag{4}$$

我们将损失函数定义为估计的 $Q$ 值与实际 $Q$ 值的差距, 即均方误差:

$$L = \sum_t \|q_t - Q(s_t, a_t)\|_2^2. \tag{5}$$

本文选择用多层感知机 (multi-layer perceptron, MLP) 作为参数模型进行拟合, 模型通过随机梯度下降进行更新. 采用这种方法有以下两个好处: (1) 与传统优化器相比, 基于强化学习的优化算法具有贪心的时间复杂度, 搜索代价显著降低. (2) 如果状态与动作的向量化表示 $f_s$, $f_a$ 具有足够的表达性, 神经网络可以从历史查询中学习到一般经验, 而不高度依赖于相似性.

### 3.5 集束搜索

RLO 算法应用深度 $Q$- 学习解决连接优化问题. 与基于策略的强化学习不同, 深度 $Q$- 学习允许我们在训练过程中利用计划的最优子结构 [14], 以大大减少所需数据量并提高训练速度. 其次, 深度 $Q$- 学习并非仅仅选择一个动作, 而是为每个动作计算一个 $Q$ 值作为选择依据. 因此, 我们可以应用集束搜索 (beam search) 的思想, 在每步选择动作的时候不仅仅选择 $Q$ 值最小的那个动作, 而是保留多个动作. 即当存在动作的 $Q$ 值与最小 $Q$ 值很接近时 (分数差小于某一阈值 $\varphi$), 我们也会保留这些动作.

这样做有以下两个好处. 首先, 可以提高计划质量, 避免错过最优计划. 神经网络对 $Q$ 值的估计具有一定误差, 仅保留 $Q$ 值最小的动作有可能错过最好的执行计划. 其次, 此方法便于与现有数据库系统进行深度整合. 例如, 我们可以对不同的排序 (interesting order) [3] 保留动作.

在本方法中, 我们引入超参数 $\varphi$ 作为阈值, 分数差小于阈值 $\varphi$ 的动作将被保留. 保留更多的动作造成了搜索空间一定程度上增大, 进而搜索时间增加. 在集束搜索中, 我们利用了搜索时间与执行时间之间此消彼长 (trade-off) 的关系, 以搜索时间的增长为代价, 换取可能更优的执行计划, 即执行时间的减少, 进而实现总时间的减少. 另外, 集束搜索可以看作是基于深度 $Q$- 学习的方法与 Selinger 优化器的一个融合. 当每步保留全部的动作时, RLO 算法的时间复杂度与 Selinger 优化器一致.

### 3.6 网络微调

我们已经说明了 RLO 如何应用基于优化器的代价模型学习. 然而, 由于偏差的基数估计或使用不切实际的规则, 代价模型的估计可能无法与实际运行时间正相关 [4]. 为纠正基数估计与代价模型的偏差. 我们利用查询执行后的反馈, 采用深度学习中的微调 (fine-tuning) 方法 [15]: 网络先在基于代价模型收集的数据集上进行训练 (与实际运行时间相比, 基于代价模型收集数据成本低廉, 且可获得大量数据), 之后冻结隐藏层, "转移" 到基于运行时间的数据集进行少量训练.

## 4 实验

### 4.1 实验配置

实验中采用的基本配置如下:

**数据集.** 本文使用基于 IMDB (Internet Movie Database) 真实数据集 (3.6 GB, 21 个表) 的 JOB 进行测试. 与大多数现实世界的数据集一样, IMDB 数据集为非均匀分布, 因此比 TPC-H 或 TPC-DC 等合成数据集更具挑战性. 测试集包含 33 个模板和总共 113 个查询[1)], 模拟了真实情况下可能提出的查询, 诸如 "2000 年至 2005 年期间发布的电影中哪些演员的评分高于 8?". 一个查询涉及 4∼17 个关系, 每个查询平均有 8 个关系.

**基线系统.** 本文分别在 Apache Calcite 和 Postgres 上实现了各种优化算法并进行比较.

**衡量指标.** 对于不同的基线系统, 我们分别比较搜索时间 (制定执行计划的时间) 和执行时间 (执行该具体计划的时间).

**参数设置.** 实验均采用 4 重交叉检验, 以 80 个查询作训练集, 33 个查询作测试集, RLO 的集束搜索参数 $\varphi$ 设为 1. 在 RLO 中, 我们采用 MLP 近似 $Q$ 值函数. 实验中, 我们发现设置层数为 3, 优化方法为 Adagrad, dropout 参数为 0.8, 取得了最好的效果.

**部署代价.** 我们通过修改数据库查询优化部分的代码进行模型部署. 在 Apache Calcite 中, 我们用深度学习库 DL4J 部署神经网络模型, 在 Postgres 中, 我们采用 TF C API 调用的 TensorFlow 神经网络. 模型的训练数据由数据库现有优化器的查询优化过程获得, 对于 80 个查询的训练集, 数据收集时间需 2 h, 训练时间需 25 min.

### 4.2 基于 Apache Calcite 的实验结果

在基于 Apache Calcite 的系统中, 我们模拟了有内存限制的代价模型 [14], 并且实现了以下几种优化算法.

(1) RLO 算法.

(2) 启发式算法: 穷举 (exhaustive search, EX)、左深树剪枝 (left-deep, LD)、右深树剪枝 (right-deep, RD)、之字形树剪枝 (zig-zag, ZZ)、贪心算法 (greedy search, GD)、QuickPick 算法 (QuickPick, QP);

(3) 学习算法: DQ [14].

在 Apache Calcite 中, 我们只能模拟连接操作的执行时间, 该时间单位与搜索执行方案的时间单位不同, 因此无法进行端到端 (总时间) 的实验. 该实验将在 Postgres 上进一步探索. 本小节接下来分别就搜索时间和执行代价进行对比.

#### 4.2.1 搜索时间比较

图 3 展示了 Apace Calcite 上不同优化器算法在关系数目不同的情况下的搜索时间对比. 我们有以下几点发现.

首先, 随着关系数目的增加, 基于学习的算法比启发式算法的搜索效率更加稳定. 例如 RLO 和 DQ 算法在关系数目增加时, 搜索时间缓慢增加, 而穷举的方法 (EX) 和剪枝的启发式算法 (如 ZZ) 的搜索时间则指数级爆炸. 这是因为, RLO 具有贪心算法的时间复杂度, 且通过批处理优化, 而剪枝的启发式算法的复杂度与关系数目呈指数关系.

其次, RLO 算法的搜索效率在关系数目较多的时候占有较大的优势. 例如对于较多的关系数, RLO 和 DQ 算法实现了极大的提速: RLO 最多可以比 EX 快 $10^4$ 倍, 比 ZZ 快 100 倍, 比 LD, RD 快 10 倍以上. 平均来说, 当关系数大于 6 时, RLO 算法比 EX 快 2867 倍, 比 ZZ 快 56 倍, 比 LD, RD 快 5.6 倍. 原因与我们之前的分析相同, RLO 具有贪心算法的时间复杂度, 远快于动态规划. 另外,

1) http://www-db.in.tum.de/leis/go/job.tgz.

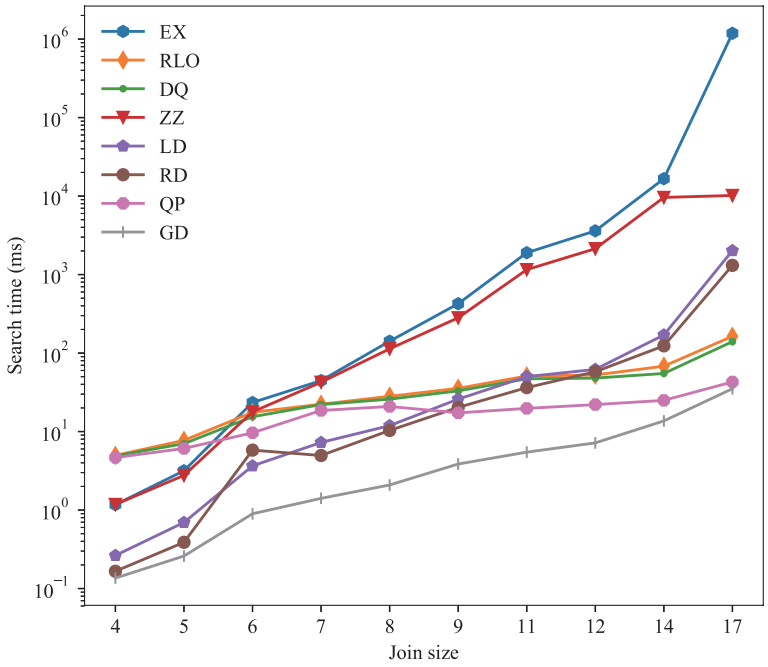



**图 3 (网络版彩图) Apache Calcite 上各个优化方法的搜索时间**

**Figure 3** (Color online) Time cost of searching the execution plan of different optimizers on Apache Calcite

GD, QP 一直保持着最小的优化时间, 这是因为它们剪枝的策略太严格, 忽略了大部分的解空间, 这会导致较差的执行效率 (参见 4.2.2 小节).

最后, RLO 的搜索效率略差于 DQ 算法. 例如 RLO 的平均搜索时间为 DQ 的 1.1 倍. 这是因为 RLO 采用了集束搜索策略, 即 RLO 在每步决策时保存多个动作. 但是带来的好处是 RLO 的执行效率更高 (参见 4.2.2 小节).

### 4.2.2 执行代价比较

图 4 展示了在设计不同关系数量时, 各个优化算法搜索得到的计划的相对代价的分布. 因为穷举 (EX) 得到的计划代价最小, 我们采用 EX 的执行代价对各个算法进行归一化. 我们有如下发现.

首先, RLO 的算法优于启发式算法. 例如 RLO 制定的查询计划优于代价最小的启发式算法 (ZZ) 80%. 固定剪枝的 LD, RD 错过了最优计划, ZZ 因搜索空间较大表现较好. GD 受限于局部最优性, 平均而言表现最差. 这也与我们之前的分析一致, 即基于启发式的算法难以搜索全量的解空间, 很难找到一个好的执行计划.

其次, RLO 优于已有的基于学习的算法 DQ. 平均来说 RLO 的执行效率优于 DQ 算法 38%. 这是因为与 DQ 相比, RLO 考虑了基于树形结构的嵌入表征, 提供了必要且更为丰富的状态信息, 并采用了集束搜索, 降低了错过较优计划的可能.

可见, RLO 的搜索效率略低于 DQ, 但是 RLO 的执行效率高于 DQ. 由于 Apache Calcite 的实验由模拟产生, 我们将在 Postgres 的实验中探索 RLO 与 DQ 端到端 (总时间) 的性能比较.

## 4.3 基于 Postgres 的实验结果

为了验证 RLO 算法是否适用于生产级别的系统, 我们将 RLO 集成到 Postgres 系统中 (采

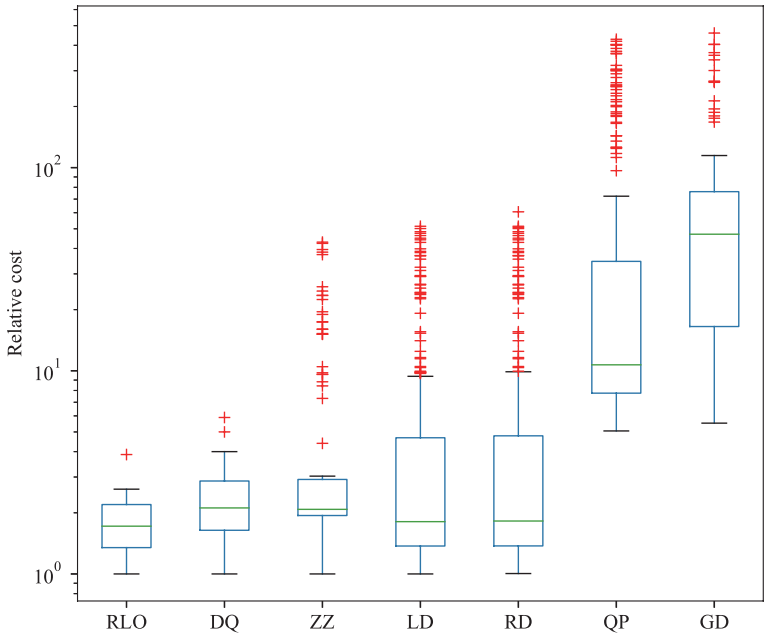



**图 4 (网络版彩图) Apache Calcite 上不同优化器相对执行代价**

**Figure 4** (Color online) Relative execution cost of different optimizers

用 TF C API 载入 MLP 近似 $Q$ 值函数). 我们接下来比较 RLO, DQ 以及 Postgres 默认的基于左深树的动态规划算法 (以下简称为 PG) 的搜索效率和执行效率.

### 4.3.1 搜索时间比较

图 5 展示了在不同关系数目的情况下, RLO, DQ 和 PG 的搜索效率的比较. 类似地, 我们有如下发现: 首先, PG 的搜索时间随着关系数目的增加爆炸地增长, 这是因为 PG 默认采用的是基于左深树的启发式方法, 随着关系数目的增加, 搜索空间指数级增加, 因此搜索代价也急速增加. 其次, DQ 和 RLO 的性能类似, 并且随着关系的数目增加没有显著的变化. 原因是 RLO 和 DQ 都是基于学习的方法, 它们以数据为导向对搜索空间进行剪枝. 另外, RLO 的搜索时间略长于 DQ, 这是因为 RLO 使用了集束搜索, 每一步保留的可能的动作可能不止一个. 最后, 从搜索效率上看, 当连接中关系的数目小于 8 时, PG 的搜索效率优于 RLO (差距小于 10 ms), 这是因为 RLO 载入神经网络需要一定的接口时间. 但是关系数目继续增加时, RLO 的搜索性能远超 PG. 平均而言, RLO 生成计划的速度是 PG 的 14.27 倍.

### 4.3.2 执行时间比较

图 6 展示了在不同关系数目的情况下, RLO, DQ 与 PG 的执行计划的时间开销. 我们可以看到, (1) RLO 的执行开销始终都小于 DQ 的执行开销, 这是因为我们采用了集束搜索, 保留了更多更优的搜索空间. (2) RLO 在大多数情况下都能搜到比较好的结果, 甚至在很多情况下能够显著超过 PG 搜到的执行计划. 例如在关系数目等于 4, 7, 9, 12, 14, 17 等时. 平均而言使用 RLO 可比 PG 的执行速度提升 12.6%, 比 DQ 提升 9.9%.

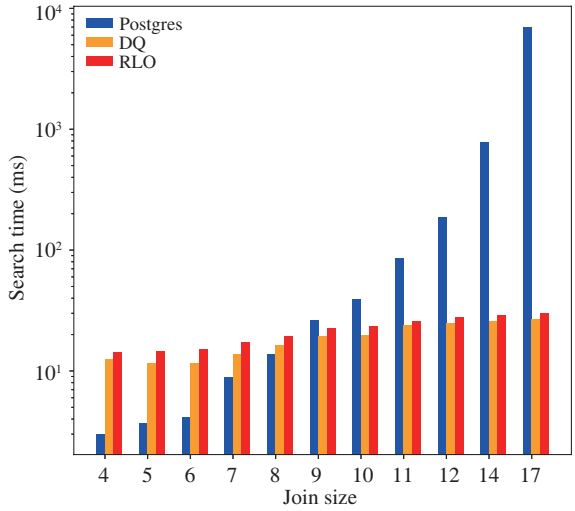



图 5 (网络版彩图) Postgres 上各个优化方法的搜索时间

**Figure 5** (Color online) Time cost of searching the execution plan of different optimizers on Postgres

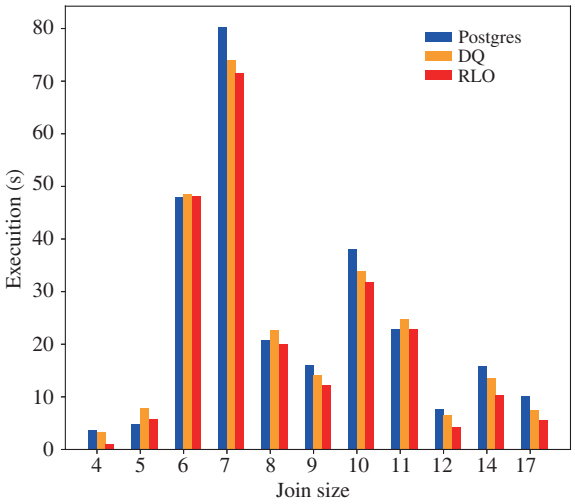



图 6 (网络版彩图) 不同优化器的执行时间

**Figure 6** (Color online) Execution cost of different optimizers

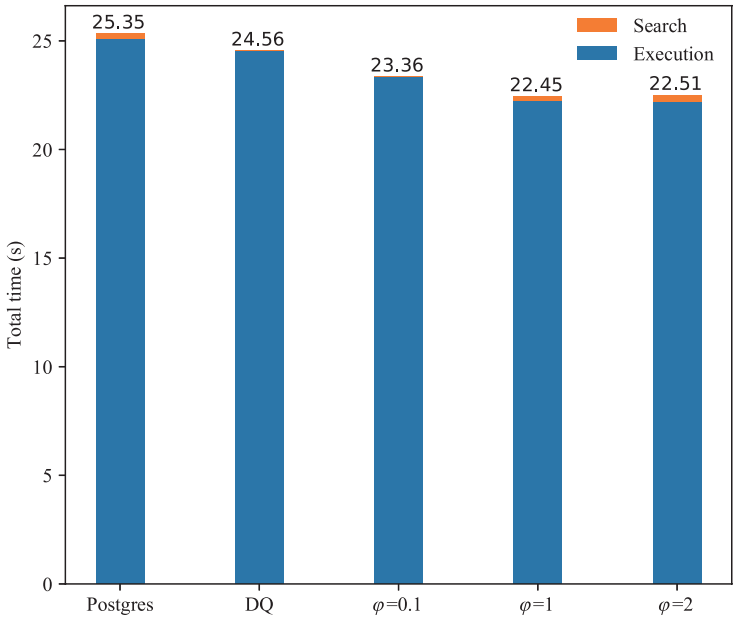



图 7 (网络版彩图) 优化时间与运行时间间的选择

**Figure 7** (Color online) Tradeoff between optimization and running time

### 4.3.3 端到端比较

接下来比较 RLO 与 PG, DQ 的端到端的性能.

对于连接优化问题, 存在搜索时间与执行时间的此消彼长 (trade off) 的关系. 搜索空间越大, 越能保证产生好的执行计划, 然而搜索时间会随之增长. 在 RLO 中, 我们可以通过超参数 $\varphi$ (定义参见 3.5 小节) 来控制搜索和执行的 trade off. 图 7 展示了 RLO, PG 和 DQ 在所有查询中端到端的平均执

行时间, 其中对于 RLO, 我们分别设置超参数 $\varphi$ 为 0.1, 1, 2.

我们可以发现, $\varphi = 1$ 时 RLO 端到端的效率最高, 为 22.45 s. 而 DQ 和 PG 分别需要 25.35 s 和 24.56 s, 即 RLO 算法比 Postgres 提升 12.9%, 比 DQ 提升 9.3%.

## 5 总结

本文进一步探索了将强化学习用于连接优化问题, 针对传统启发式算法的缺点, 结合深度 $Q$- 学习提出了算法 RLO. RLO 能够以数据为导向对搜索空间进行剪枝. 为了增强 RLO 的效果, 作者提出了基于树形结构的嵌入和集束搜索以及网络微调等策略. 在 Apache Calcite 和 Postgres 上的实验表明, 相比于现有的启发式算法和基于学习算法, RLO 能够获得更好的连接优化的性能.

## 参考文献


1 Babcock B, Chaudhuri S. Towards a robust query optimizer: a principled and practical approach. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, 2005. 119–130

2 Krishnamurthy R, Boral H, Zaniolo C. Optimization of nonrecursive queries. In: Proceedings of the 12th International Conference on Very Large Data Bases, 1986. 128–137

3 Selinger P G, Astrahan M M, Chamberlin D D, et al. Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, 1979. 23–34

4 Leis V, Gubichev A, Mirchev A, et al. How good are query optimizers, really? In: Proceedings of the VLDB Endowment, 2015. 204–215

5 Waas F, Pellenkoft A. Join order selection (good enough is easy). In: Proceedings of British National Conference on Databases, 2000. 51–67

6 Sutton R S, Barto A G. Reinforcement Learning: An Introduction. Cambridge: MIT Press, 2018. 34–152

7 Hester T, Vecerik M, Pietquin O, et al. Deep Q-learning from demonstrations. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018

8 Begoli E, Camacho-Rodríguez J, Hyde J, et al. Apache calcite: a foundational framework for optimized query processing over heterogeneous data sources. In: Proceedings of the 2018 International Conference on Management of Data, 2018. 221–230

9 Ziane M, Zaït M, Borla-Salamet P. Parallel query processing with zigzag trees. In: Proceedings of the International Journal on Very Large Data Bases, 1993. 277–302

10 Chande S V, Sinha M. Genetic optimization for the join ordering problem of database queries. In: Proceedings of 2011 Annual IEEE India Conference, 2011. 1–5

11 Kipf A, Kipf T, Radke B, et al. Learned cardinalities: estimating correlated joins with deep learning. 2018. ArXiv: 1809.00677

12 Ortiz J, Balazinska M, Gehrke J, et al. Learning state representations for query optimization with deep reinforcement learning. In: Proceedings of the 2nd Workshop on Data Management for End-To-End Machine Learning, 2018. 1–4

13 Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration. In: Proceedings of the 1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018. 3

14 Krishnan S, Yang Z, Goldberg K, et al. Learning to optimize join queries with deep reinforcement learning. 2018. ArXiv: 1808.03196

15 Yosinski J, Clune J, Bengio Y, et al. How transferable are features in deep neural networks? In: Proceedings of Advances in Neural Information Processing Systems, 2014. 3320–3328

# RLO: a reinforcement learning-based method for join optimization

Xinyi ZHANG[1], Zhipeng ZHANG[1], Tieying ZHANG[2], Bin CUI[1*] & Ju FAN[3]

1. *Key Laboratory of High Confidence Software Technologies* (*MOE*), *School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
2. *Database and Storage Laboratory, Damo Academy, Alibaba Group, Beijing* 100029, *China*;
3. *School of Information, Renmin University of China, Beijing* 100872, *China*
* Corresponding author. E-mail: bin.cui@pku.edu.cn

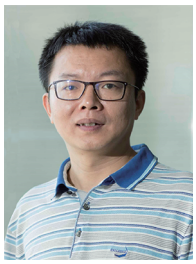
**Abstract** Join optimization is one of the most important research problems in database systems. Traditional join optimizers are usually proposed based on heuristics, which are expensive and often fail to generate the optimal execution plan. There are two reasons accounting for this. (1) The optimizers are based on heuristics and only explore a subset of the search space. (2) They do not use the history logs and cannot estimate the goodness of their generated plans on a specific join problem. To tackle these challenges, we propose RLO, a reinforcement learning-based optimizer for join optimization. We model the join optimization problem as a Markov decision process and use deep $Q$-learning to estimate the possible reward of a possible operation. To boost the effectiveness of RLO, we further propose a tree-based embedding method to represent the "state" and use a beam search to avoid missing the optimal plans. We implement RLO in Apache Calcite and Postgres. Extensive experiments demonstrate that: (1) Apache Calcite RLO is 10×–56× faster in finding the execution plan and 80% faster in executing the plan than the state-of-the-art heuristics. (2) Compared with the native Postgres implementation, RLO can be 14× faster in finding the execution plan and 12.9% faster in an end-to-end comparison.

**Keywords** join optimization, reinforcement learning, embedding method, beam search

**Xinyi ZHANG** is an M.S. candidate at Peking University. Her research interest focuses on intelligent databases.

**Zhipeng ZHANG** is a Ph.D. candidate at Peking University. His research interests include distributed machine learning and big data analytics.

**Tieying ZHANG** is a research scientist at Database and Storage Laboratory, Damo Academy, Alibaba Group. His research interests focus on intelligent databases and distributed systems.



**Bin CUI** is a professor and deputy head of the Department of Computer Science at Peking University. He obtained his Ph.D. degree from National University of Singapore in 2004. His research interests include database system architectures, query and index techniques, big data management, and distributed machine learning.