文章编号：2095-2163(2021)09-0119-04 中图分类号：TP392 文献标志码：A

大规模知识图谱的多查询优化问题研究

郭欣彤，高 宏
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要： 多查询优化问题是从一组查询中找出公共子结构，将其结果缓存起来，每个查询可以利用缓存结果构建自己的结果。由于知识图谱上的多查询优化是 NP-hard 问题，现有方法无法在大量查询同时到达时高效地查找公共子结构，也无法保证优化后查询时间一定减少。因此，本文提出了一个新的分布式，基于内存的 RDF 查询引擎 Leon 来处理多查询优化问题。Leon 使用了基于特征集合的索引和划分方法，具有简单高效、空间占用小的特点。针对现有检测查询之间公共子结构检测算法时间复杂度高的特点，本文提出了一个新颖的多查询优化算法：利用特征集合快速过滤没必要优化的查询，在剩下来的查询中精确、高效地查找公共子结构。实验结果证明：引入多查询优化情形下，时间是基准方法的 1/10。

关键词： 知识图谱；多查询优化；公共子结构检测

Research on multi-query optimization problem in knowledge graph

GUO Xintong, GAO Hong
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】 Multi-query optimization (MQO) aims to identify common sub-expressions of a set of queries, cache their results, and assemble queries' final results to avoid redundant computation. Due to the NP-hardness of MQO in knowledge graph, existing methods could not detect common subquery gracefully and cannot assure the runtime reduction after optimization, either. This paper presents Leon, a distributed in-memory knowledge graph query engine, to address the MQO problem. Leon applies a characteristic-set-based indexing and partitioning scheme, which is simple but time-saving. This paper proposes a novel MQO algorithm targeting the high complexity of common sub-expression detection. It uses characteristic sets to filter out the non-promising queries and then discover the high-quality common subquery in the remaining ones. The extensive experiments prove that Leon outperforms 10x faster over the baseline method.

【Key words】 knowledge graph; multi-query optimization; maximal common substructure detection

0 引 言

知识图谱将客观世界中的海量信息以结构化的形式描述，提供了强大的组织、管理和理解互联网海量信息的能力。知识图谱不仅给搜索引擎带来了新的活力，同时也在智能决策中显示出强大威力，是人工智能的重要基石。随着知识图谱被广泛应用于各个领域，在真实 SPARQL 查询历史中发现很多相似查询。如果一定时间段内的相似查询可以一起处理，就能减少冗余计算，从而大大降低查询响应时间，提高吞吐量。因此，本文研究在知识图谱上的多查询优化问题。

关系数据库上的多查询优化问题是 NP-hard，由于 SQL 与 SPARQL 二者之间有等价关系，因此 RDF/SPARQL 上的多查询优化问题也是 NP-hard。MQO 是关系数据和半结构化数据上的一个经典问题，有很多方法取得了很好的效果。一个很直观的想法是把已有方法推广到 RDF 数据上，但现有的方法并不能无缝的集成在已有的 RDF 查询引擎上，主要原因有：

(1)关系数据上的 SQL 查询通常可以转换成一棵抽象语法树，在树上检测同构的子树较为简单，而 SPARQL 查询通常表示成图结构，因此公共子结构检测更难；

(2)RDF 的物理存储和索引并没有固定的模式，每个系统选用的方法都不同，例如 RDF-3X 中使用全索引结构[1]，Jena 使用属性表[2]，还有最近很多系统采用的垂直划分方法，这些多样的存储模式和索引选择，使得代价估算不准确，某些情形下优化后的响应时间并不总是小于未优化的；

(3)真实世界中的 SPARQL 查询远比关系数据库上的 SQL 语句复杂，连接更多，这可以通过比较 TPC 测试集和一些 RDF 测试集得出结论。随着连接个数的增加，代价估计的准确性大幅下降。在查

作者简介：郭欣彤(1991-)，女，博士研究生，主要研究方向：知识图谱、图数据挖掘；高 宏(1966-)，女，博士，教授，博士生导师，主要研究方向：数据质量、传感网、数据挖掘。

收稿日期：2021-03-24

询处理时,代价估计是非常重要的,错误的代价估计有时反而会使查询效率下降[3]。

从图的角度也有很多针对多查询优化的解法,大多数都需要找出最大公共子图(Maximum Common Subgraph, MCS)。由于 MCS 检索是 NP-complete 问题,很多查询同时到达时,时间开销难以承受。

除了多查询优化问题本身的复杂度,随着知识图谱数据集规模的不断增大, RDF 查询引擎的可扩展性变得尤为重要,需要一个分布式的 RDF 查询引擎提高查询处理时的并行性,从而减少响应时间,此时数据划分变得尤为重要。众所周知,图划分问题是 NP-complete 的,现有系统大多都基于启发式方法,一些系统使用简单的哈希方法,但这种方法导致很少查询能够无通信的并行计算;一些系统使用了较为复杂的划分算法,这类方法局限于很高的预处理代价和较高的复制系数。

为了解决现有方法不足,本文提出了一个新的分布式,基于内存的 RDF 查询引擎 Leon,能够解决多查询优化问题。Leon 使用一个基于特征集合的平衡划分算法,具有时间复杂度低,有利于星型查询等特点。对之后的多查询优化也有很多好处,本文提出了一个新颖的多查询优化算法。首先,利用特征集合快速过滤没有优化前景的查询;其次,利用 triplet 进一步过滤,只保留十分相似的查询;最后,在剩下来的查询中精确、高效地查找公共子结构。实验结果表明,在多查询负载下,时间是无优化时的 1/10。

1 算法框架

1.1 特征集合

RDF 图中的主体都有一系列谓词,同一实体类型的主体拥有的谓词也十分相似。真实世界中的 RDF 数据集即使包含的三元组个数很多,谓词集合也很少。例 如,对于一个拥有超过 8 亿三元组的 UniProt 数据集来说,谓词集合只有 615 个;DBLP 数据集包含 176.63M 条元组,谓词集合只有 95 个。许多其它数据集也观察到这种现象,例如:YAGO、LibraryThings、Barton 等。Neumann 和 Moerkott 由此提出了特征集合的概念[4]。

给定一个 RDF 数据图 $G(V,E,L)$,s 是一个主体,s 的特征集合 $S(s)$ 定义为式(1):

$$S(s) = \{p \mid \exists o: < s,p,o > \in G\} \quad (1)$$

图 G 中所有特征集合可以表示为式(2):

$$S(G) = \{S(s) \mid \exists p,o: < s,p,o > \in G\} \quad (2)$$

基于以上两个定义,定义特征集合块,特征集合为 S_i 的主体对应的所有三元组,形成了一个特征集合块 B_i ,式(3):

$$B_i = \{ < s,p,o > \mid S(s) = S_i\} \quad (3)$$

图中所有特征集合块可以表示为式(4):

$$B(G) = \{B_i \mid S_i \in S(G)\} \quad (4)$$

显然, $B(G)$ 是 RDF 数据图以主体为中心的一个划分,特征集合S_i 和特征集合块 B_i 之间是一一对应的。

将 RDF 图中的三元组根据特征集合编码,使用基于特征集合的平衡划分算法将特征集合块发送到从节点中。为了使各个节点上的工作量较为平均,保持每台机器上的三元组个数接近。假设划分计划表示为 $P = \{P_1,P_2,\cdots,P_k\}$。将 $|E|$ 条三元组平均分配到 k 个节点上,那么每台机器上的数量应为 $|E|/k$ 个。根据每个特征集合块内三元组数量从大到小排序,将排序后的特征集合块贪心地放入剩余容量最多的节点。

1.2 公共子结构检测

多查询优化算法的核心是公共子结构检测,主要分为 3 个步骤:

(1) 将 SPARQL 查询根据特征集合进行分解,根据包含的特征集合,将查询划分成几个粗粒度的聚类;

(2)使用 triplet 进一步精细聚类,只保留非常相似的查询进行优化;

(3) 找出查询间准确的公共子结构。

到达的 SPARQL 查询会被分解成一系列不相交的星型的子查询,称为 fork。fork 中的三元组模式有相同的主体连接变量,根据谓词每个 fork 都能映射到 0 个或多个特征集合,相应的特征集合块一定含有这个 fork 的结果。

要筛选出哪些查询可以一起优化,因此使用 k-means 聚类算法粗粒度地对查询集合聚类。k-means 聚类算法使用基于特征集合的 Jaccard 系数作为度量,使用特征集合对查询粗聚类是高效且准确的,特征集合中不仅包含了查询的谓词信息,也反映了结构信息。如果两个查询包含的特征集合都相同,那么其有很大可能结构也相似,相比已有方法只使用谓词进行聚类更加准确。

粗聚类后,在每个簇上进一步筛选可以共同优化的查询。将每个查询 q 分解成一系列 triplet,所谓 triplet 就是相连的两条边组成的三元组。表 1 就是图 1 的 triplet 分解结果。

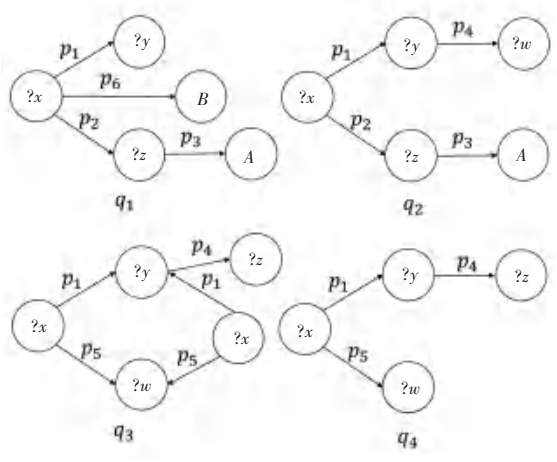

图 1　一组查询

Fig. 1　A set of queries

表 1　Triplets

Tab. 1　Triplets

查询	triplet, 实例个数
q_1	$tp_1: ?x \xrightarrow{p_2} ?z \xrightarrow{p_3} A, 1$ $tp_2: ?z \xleftarrow{p_2} ?x \xrightarrow{p_1} ?y, 1$ $tp_3: B \xleftarrow{p_6} ?x \xrightarrow{p_5} ?y, 1$ $tp_4: B \xleftarrow{p_6} ?x \xrightarrow{p_4} ?z, 1$
q_2	$tp_1: ?x \xrightarrow{p_2} ?z \xrightarrow{p_3} A, 1$ $tp_2: ?z \xleftarrow{p_2} ?x \xrightarrow{p_1} ?y, 1$ $tp_5: ?x \xrightarrow{p_4} ?y \xrightarrow{p_4} ?w, 1$
q_3	$tp_5: ?x \xrightarrow{p_1} ?y \xrightarrow{p_4} ?z, 2$ $tp_6: ?y \xleftarrow{p_1} ?x \xrightarrow{p_5} ?w, 2$ $tp_7: ?x \xrightarrow{p_1} ?y \xleftarrow{p_1} ?m, 1$ $tp_8: ?x \xrightarrow{p_5} ?w \xleftarrow{p_5} ?m, 1$
q_4	$tp_5: ?x \xrightarrow{p_2} ?y \xrightarrow{p_4} ?w, 1$ $tp_6: ?y \xleftarrow{p_1} ?x \xrightarrow{p_5} ?w, 1$

如果两个查询有很多公共的 triplet，那么其很可能有很大的公共子结构，计算两两查询的亲密度，式(5)：

$$AF(q_i, q_j) = \frac{|\ CTL(q_i, q_j)\ |}{\min\{|TL(q_i)|, |\ TL(q_j)\ |\}} \quad (5)$$

亲密度越高，说明两个查询公共子结构越大。只有当两个查询之间的亲密度高于一个阈值 β 时，才会被考虑一起优化。将不相似查询过滤出去后，在剩下的查询上运行最大公共子结构发现算法，找到的公共子结构如图 2 所示。

1.3　查询重写

给定一组查询 $Q = \{q_1, q_2, \cdots, q_n\}$，假设 q_c 是其公共子结构，将每个查询改写如式(6)和式(7)：

$$q_i' = q_i \setminus q_c \quad (6)$$

$$Q_{OPT} = \{q_c, q_1', q_i' \times q_c, \cdots\} \quad (7)$$

为 Q_{OPT} 生成查询计划，进行查询处理。

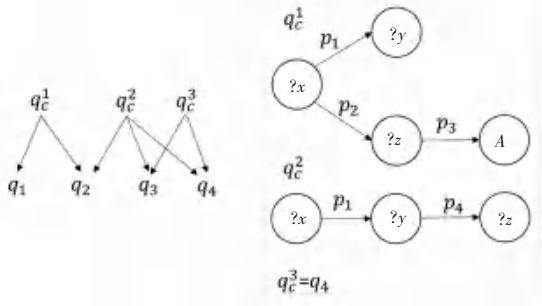

图 2　查询的公共子结构

Fig. 2　Maximal common substructure

2　实验结果

使用两个数据集进行实验。LUBM 是一个广为人知的 RDF 测试基准，其优点是数据集中的度量值随着数据集大小线性增长，结构简单的数据集，只含有 11 个特征集合，本文使用 LUBM 数据生成器生成 LUBM1k 数据集；WatDiv 也是一个人工数据集，其可以测试不同查询负载情况下 RDF 系统的性能，本文生成了一个超过 10 亿条三元组的 WatDiv1B 数据集。将 Leon 与目前最先进的 AdPart[5] 系统进行对比，AdPart 使用主体上哈希的划分算法，并能根据负载进行数据复制。

为 LUBM1k 和 WatDiv1B 数据集分别生成包含 $10k$ 个查询的查询负载，比较 4 个算法 AdPart-random、AdPart-seq、Leon 和 Leon-non(无优化)，其中“AdPart-seq”表明相似的查询同时到达，“AdPart-random”代表相似查询随机到达，做这样的区分是因为 AdPart 是通过一段时间内重复出现的查询进行复制的。

图 3 给出了 LUBM1k 负载上的查询表现，横轴代表查询个数，纵轴代表累积处理时间。对于 AdPart-random，由于相似查询随机到达，并没有检测到很多公共子结构，因此累积处理时间迅速增长；而 AdPart-seq 识别到了连续到来的相似查询，并将公共子查询的结果在集群中复制，因此时间远远小于 AdPart-random。相比之下，Leon 将所有查询看成一个整体，对不同的查询聚簇分别优化，运行时间比 AdPart-seq 减少了大约 40%，这是因为最大程度的共享了公共部分的计算，而且选择了正确的公共子结构进行重写；Leon-non 比 AdPart-random 花费时间少大约 32%，这是因为基于特征集合的划分方法减少了查询处理时间。Leon 的运行时间大约是 Leon-non 的 1/10，说明引入多查询优化可以大大提高查询响应时间。

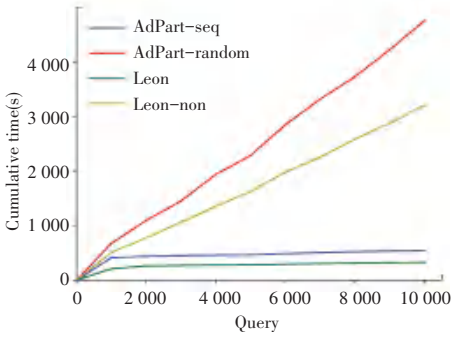

图 3 LUBM 负载表现

Fig. 3 LUBM workload performance

图 4 表明在 WatDiv 负载上各个系统的查询表现也有相同的变化趋势。由于 WatDiv 的查询模板更复杂,因此特征集合的剪枝能力发挥了巨大的作用。此外,基于特征集合的基数估计更准确,使得生成的查询计划更优。Leon 比 AdPart-seq 节约了 25% 的时间,Leon 比 Leon-non 节约了 91%的时间。

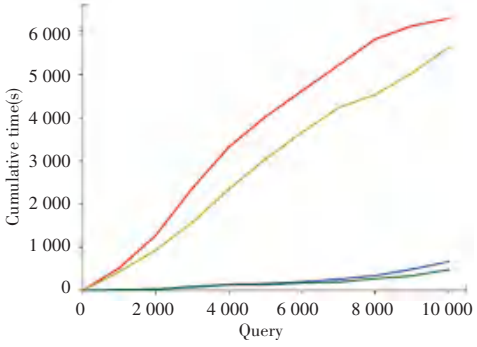

图 4 WatDiv1B 负载表现

Fig. 4 WatDiv1B workload performance

3 结束语

本文探讨大规模知识图谱上的多查询优化问题,提出了一个基于内存的,分布式 RDF 处理引擎 Leon。Leon 基于特征集合对 RDF 数据进行划分,大大减少了连接个数,从而减少了计算时间。同时本文也提出了一个高效的提取公共子结构的算法,利用特征集合和 triplet 两次筛选,逐渐缩小搜索空间。实验证明,Leon 在单查询性能上与目前最好的 RDF 查询引擎相当;在多查询优化上,响应时间是无优化的版本的 1/10。

参考文献

[1] Neumann, Thomas and G. Weikum. RDF-3X: a RISC-style engine for RDF[C] //Proc. VLDB Endow. Auckland, New Zealand: 2008: 647-659.

[2] CARROLL J, et al. Jena: implementing the semantic web recommendations[C] //WWW, New York,2004.

[3] Leis, Viktor, et al. How Good Are Query Optimizers, Really? [C] // Proc. VLDB Endow, Kohala Coast, Hawaii: 2015: 204-215.

[4] Neumann, Thomas and G.Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins[C] // 2011 IEEE 27th International Conference on Data Engineering, Hannover, Germany:2011: 984-994.

[5] AL-HARBI R. et al. Accelerating SPARQL queries by exploiting hash-based locality and adaptive partitioning[J]. The VLDB Journal, 2016, 25: 355-380.

(上接第 118 页)

[8] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[EB/OL]. http://arxiv.org/abs/1409.1556, 2014.

[9] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[10] Sandler M, Howard A, Zhu M, et al. Mobilenetv2: Inverted residuals and linear bottlenecks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 4510-4520.

[11] TAN C, SUN F, KONG T, et al. A survey on deep transfer learning [C]//International conference on artificial neural networks. Springer, Cham, 2018: 270-279.

[12] MUSAVI M T. Receptive field estimation for Gaussian-based neural networks [C]//Proceedings of the IEEE International Conference on Neural Networks, 1993: 1343-1347.

[13] SI J, HARRIS S L, YFANTIS E. A dynamic ReLU on neural network [C]//2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS). IEEE, 2018: 1-6.

[14] HOWARD A G, ZHU M, CHEN B, et al.Mobilenets: Efficient convolutional neural networks for mobile vision applications[EB/OL].http://arxiv.org/abs/1704.04861, 2017.

[15] ZHUANG F, QI Z, DUAN K, et al. A comprehensive survey on transfer learning[J]. Proceedings of the IEEE, 2020.

[16] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, et al. Dropout: a simple way to prevent neural networks from overfitting [J]. The journal of machine learning research, 2014, 15(1): 1929-1958.

[17] MARCEL S, RODRIGUEZ Y. Torchvision the machine-vision package of torch[C]//Proceedings of the 18th ACM international conference on Multimedia. 2010: 1485-1488.

[18] WIEDEMANN S,MARBAN A, MÜLLER K R, et al. Entropy-constrained training of deep neural networks [C]//2019 International Joint Conference on Neural Networks (IJCNN). IEEE, 2019: 1-8.

[19] KINGMA D P, BA J. Adam: A method for stochastic optimization[EB/OL].http://arxiv.org/abs/1412.6980, 2014.