RapidMatch: A Holistic Approach to Subgraph Query Processing

摘要

子图查询可搜索数据图中与查询图相同的所有嵌入。目前已开发出的两种算法，一种是基于图探索的算法，另一种是基于连接的算法。由于算法和实现上的差异，基于连接的系统可以高效处理几个顶点的查询图，而基于探索的方法通常处理查询图中多达几十个顶点。本文提出了 RapidMatch，一个集成了这两种方法的整体子图查询处理框架。具体来说，RapidMatch 不仅能运行选择和连接等关系运算符，还能利用图探索中的图结构信息进行过滤和连接计划生成。因此，RapidMatch 在各种查询工作负载上的表现都优于这两种方法。

关键词：子图查询；索引枚举框架

1 引言

近年来，图数据结构在学术界和工业界的应用越来越广泛，包括社交网络分析、道路分析、化学分子合成、生物蛋白质网络分析、金融欺诈检测等等。子图匹配 (Subgraph Matching) 是图分析领域研究的一个重要课题，其旨在一个大的数据图上匹配一个给定的查询子图，获得这个子图的所有同构嵌入 (embedding)。由于子图匹配是一个 NP-hard 的问题，因此如何在一个大的数据图上有效的时间内枚举所有的查询结果是子图匹配问题研究的重中之重。

2 相关工作

对比了两种主流求解子图查询问题的算法，基于探索的算法和基于连接的算法，分析了两种算法的优劣势，并结合分析提出了本文中所提出的 Rapidmatch 算法 [4]，本质是利用了两种算法的长处综合得出的一种算法，并对开源项目中的一些数据集进行了测试。

2.1 基于探索的算法

基于探索的方法采用的是 Ullmann 提出的回溯模型 [5]，通过某一个查询顶点匹配顺序迭代地将查询顶点映射到数据顶点以扩展中间结果。即按照匹配顺序进行深度优先搜索，如果下一个查询顶点没有有效的匹配或者到了最后的深度，那么就会递归回溯。根据执行过程，基于探索的方法可以分为三类。第一类算法，如 QuickSI [3]，遵循直接枚举框架，直接探索数据图以枚举匹配项。第二类算法，如 GADDI [6]、SPath 和 SGMatch，利用索引枚举框

架，在索引的帮助下评估所有查询。第三组算法如 TurboIso、CFLMatchQuickSI [1]、CECI 和 DP-iso [2] 采用了预处理计算框架。先前的性能研究表明：由于索引构造，索引枚举方法存在严重的可扩展性问题；预处理计算方法通常在其中表现最好。算法思想如下算法 2 简要介绍了大多数基于探索的算法。第 1 行首先对于每个 Q 中的顶点，构造候选顶点集 C(u) 然后修剪 C(u) 。之后，它构建了一个辅助结构 A，用于维护候选顶点集之间的数据边。接下来，第 2 行生成匹配顺序，它是查询顶点的序列。基于探索的方法通常采用贪心算法，即首先选择一个起始顶点，迭代添加对 A 中嵌入数量的基数估计。最后，Enumerate 过程递归地查找所有结果。并记录查询顶点和数据顶点之间的映射。第 6 行计算局部候选顶点集。

Algorithm 2: Exploration-Based Method

Input: a query graph Q and a data graph G;
Output: all subgraph isomorphisms from Q to G;
1 $C, \mathcal{A} \leftarrow$ build candidate vertex sets and auxiliary structures;
2 $\varphi \leftarrow$ generate a matching order;
3 Enumerate$(C, \mathcal{A}, \varphi, \{\}, 1)$;
4 **Procedure** Enumerate$(C, \mathcal{A}, \varphi, M, i)$
5 **if** $i = |\varphi| + 1$ **then** Output M, **return**;
6 $u \leftarrow \varphi[i], C_M(u) \leftarrow \bigcap_{u' \in N_+^{\varphi}(u)} \mathcal{A}_u^{u'}(M[u'])$;
7 **foreach** $v \in C_M(u)$ **do**
/* Remove if to find subgraph homomorphisms. */
8 **if** $v \notin M$ **then**
9 Add (u, v) to M;
10 Enumerate$(C, \mathcal{A}, \varphi, M, i + 1)$;
11 Remove (u, v) from M;

图 1. 基于探索的方法

2.2 基于连接的算法

基于连接的方法则不同，它们会将查询子图建模为一个关系查询，然后通过一系列的关系操作去执行这个查询，例如：选择操作（selection）和连接操作（join）。即它们是从关系（也就是边）的角度去匹配的。一些传统的关系数据库例如 MonetDB 和 PostgreSQL 会将子图匹配问题实现为一系列的二元连接，即每一步通过匹配一条边（关系）来扩展中间结果。一些最近的工作 [3, 6] 通过最坏情况下最优连接（WCOJ）的方式在每一步匹配连接到某一个查询顶点的多条边（关系）来扩展中间结果，获得了更好的性能。这种基于连接的方法在处理子图查询问题时，主要思想集中在优化连接操作，以最小化运行时间和提高查询效率为目标。WCOJ 类算法：WCOJ（Worst-case optimal join）类算法旨在将运行时间控制在查询图的某种上界，以应对最坏情况的运行时长。其中，AGM（基于 fractional edge cover 的理论）提供了对运行时间上界的精确限制，通过 fractional edge cover 数量的最小值来限定运行时间，这种方法考虑了查询图的边覆盖度。通过构建连接查询 Q，其中每个查询边的关系都与图 G 中对应边的标签和顶点标签匹配，WCOJ 算法可以通过匹配顶点和边来进行高效的连接操作。LFTJ

(Leapfrog Triejoin)：LFTJ 是 WCOJ 的一个具体实例，它使用一种称为 GenericJoin 的算法抽象。LFTJ 通过设定匹配顺序，递归地将查询顶点与数据顶点绑定来评估查询图。在这个过程中，它利用了 set intersections 来高效计算查询结果。连接优化: RapidMatch 和 Graphflow 等系统使用了 WCOJ 类算法来处理子图查询。这些系统都致力于优化连接计划和连接操作，以尽量减少计算成本和运行时间。Graphflow 采用了贪心算法来生成连接计划，优化了连接操作的顺序以最小化运行时间。此外，Graphflow 还支持边标签和有向图的查询，在考虑了各种指标后，根据成本模型来优化连接计划。一种基于连接的算法（LFTJ）如下图

Algorithm 3: Leapfrog Triejoin (LFTJ)

Procedure LFTJ$(Q = (V, E), \varphi, M, i)$
  if $i = |\varphi| + 1$ **then** Output M, **return**;
  $I \leftarrow \{u \leftarrow \varphi[i]\}, X_M(I) \leftarrow \bowtie_{e \in E_I} \pi_I R_e, J \leftarrow V - I$;
  foreach $v \in X_M(I)$ **do**
    Bind u to v in M;
    $Q' \leftarrow \bowtie_{e \in E_J} \pi_J(R_e \ltimes R(u))$ where $R(u) \leftarrow \{v\}$;
    LFTJ $(Q', \varphi, M, i + 1)$;
    Unbind u from v in M;

图 2. 基于连接的方法 (LFTJ)

2.3 两种方法的对比

对于基于探索的算法，其优点在于

1、算法基于候选顶点集合进行结果枚举，有效地利用了候选集合的特性；

2、通过沿着查询图的生成树剪枝候选顶点集合，有效地减少了搜索空间，提高了效率；

3、对于稳态的候选顶点集合，在每一步都能够产生与基于连接算法相同的中间结果；

但是对于大型图或复杂查询图，可能会受到搜索空间过大的影响，导致计算复杂度增加。

对于基于连接的算法，其优点在于

1、算法通过计算关系的连接来枚举结果，能够更加高效地利用关系之间的匹配关系；

2、在处理大规模图数据时，基于连接的算法往往能够更快地收敛到最终结果；

但是当查询图或数据规模较大时，可能会受到连接操作复杂度的影响，算法可能会有较高的时间复杂度。对比二者的优劣势可以得出，基于探索的算法通过剪枝和优化搜索空间，能够在某些情况下更高效地找到结果。然而，对于大型图数据，可能会受到搜索空间限制。基于连接的算法则更加适合于处理大规模的图数据，能够更有效地利用关系连接来加速结果的计算过程，但在某些情况下，其计算复杂度可能会较高

3 本文方法

3.1 本文方法概述

本文应用了 CFL 算法，但在此基础上做出了改进，相较于先前的 core-forest-leaf 分解，首先进行核分解（nucleus decomposition），不同于 CFL 算法的仅有的 2-core 分解，本文中的分解对核进行了进一步的分解。得到查询图的核结构、森林和核森林 (T1,2、T2,3 和 T3,4)。这个过程能够识别查询图中的密集子图以及它们的层级关系，为后续的处理提供了关键信息。采用 T1,2、T2,3 和 T3,4 的原因在于它们包含所有的查询顶点，并且之前的研究表明在设置 r=3 和 s=4 时能够高效地获取密集子图。算法接着进行子图查询处理。RelationFilter 根据顶点标签为每个查询边构建关系，然后将查询图分解为一组基于树结构的子查询。JoinPlanGenerator 根据 RelationFilter 获取的关系统计信息和核结构 T1,2、T2,3 和 T3,4 生成连接计划。连接计划的执行顺序优先对 QC 进行 LFTJ（Leapfrog Triejoin）评估，然后对后续的查询执行一系列哈希连接操作，以找到最终的结果。在整个流程中，对连接计划、数据布局和关系进行多种优化，例如 RelationEncoder 在运行时优化数据布局、构建哈希索引、高级的集合交集方法、交集缓存和失败集合剪枝等，以提高查询效率。本文方法的流程图如下

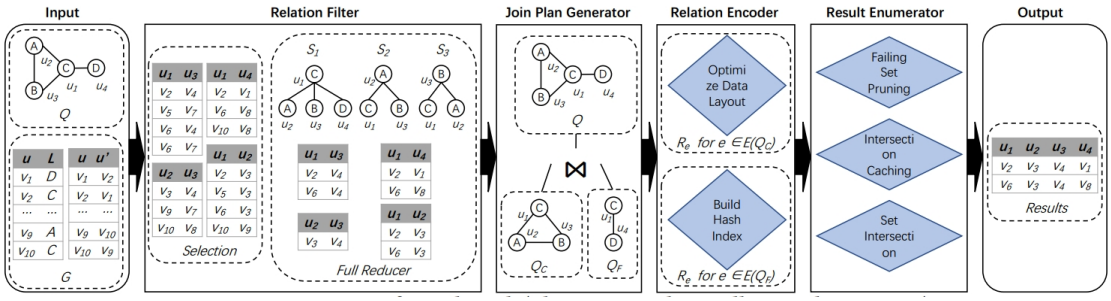

图 3. 本文方法概览流程图

3.2 关系过滤层

关系过滤层的主要作用是利用图结构特征来消除查询图中存在的“冗余元组”（dangling tuples），冗余元组指的是在关系中存在的数据边，但这些边在最终的查询结果中并不会被使用，它的存在会导致无效的计算和性能下降。简单的基于标签的方法会生成关系 R(u, u)，但它忽略了底层的图结构。这导致了生成的关系中存在大量的冗余元组，即数据边在最终结果中并不会被利用到，这直接降低了连接操作的性能。删除所有查询图中的冗余元组是一个 NP-hard 问题，因为这涉及到子图同态（或同构）问题的困难性，需要在保持查询图结构的同时消除不必要的边。

为了应对这个问题，本文提出了一种启发式算法，利用图结构特征来消除冗余元组。算法将查询图 Q 分解为一组树结构的子查询 Q 。对每个子查询 Q 应用 full reducer，以消除冗余元组。在执行完全减少器后，每个关系 Re 中的数据边（其中 e E(Q)）都会出现在从 Q 到 G 的子图同态中。通过利用图结构的树形特征，这种方法能够更有效地消除那些不会出现在最终结果中的数据边，从而提高了查询操作的效率。虽然删除所有冗余元组是一个 NP-hard 问题，但这种启发式方法利用了图的特定结构，使得计算上更加高效，并且在减少无效计算和优化连接计划方面具有显著作用。

3.3 加入计划生成层

RapidMatch 中的连接计划（join plan）是一个匹配顺序 F，它将核心顶点 V（Q_C）放置在非核心顶点之前（即，V(Q) − V (Q_C)）。并且是 F 连通的，即对于任意的 $1 \leqslant i \leqslant |\phi|$，Q[F[1:i]] 是一个连通图。非核心顶点在 F 中有且只有一个向后的邻居。RapidMatch 首先使用 LFTJ（Left-Deep Full Theta Join）执行 $Q_C = \bowtie_{e \in E(Q_C)} R_e$，以 F[1 : |V (QC)|] 作为匹配顺序，然后通过一系列哈希连接（hash joins）执行 $Q_c \bowtie Q_F$ ，在实践中，RapidMatch 以流水线的方式执行所有连接，即一个连接生成的结果立即传递给后续的连接，以避免生成中间结果。现有的方法通常基于代价模型 cost(F) 进行连接计划的优化，这个代价模型考虑了枚举过程中产生的中间结果的总数。然而，当查询规模较大时，对于查询子查询输出大小的基数估计非常困难，估计的偏差可能导致连接计划非常低效。因此，RapidMatch 从图结构的角度对连接顺序 F 进行了优化，避免了无效的搜索路径。通过计算某个中间结果不能进一步扩展的顶点，对 F 进行了优化。这个连接顺序会将具有更多向后邻居的查询顶点放置在前面，将顶点优先考虑放置在 Q 的密集区域，然后使用核心分解（nucleus decomposition）来生成 F，因为它能够有效地找到具有详细层次结构的高质量密集区域。对于哈希连接的顺序，会优先选择具有较小基数的关系进行连接。

3.4 关系编码器

LFTJ（Left-Deep Full Theta Join）算法使用两级前缀树（trie）来存储关系 $R(u, u')$。第一级存储了 u 的值，第二级记录了已排序的邻居。优化的关系数据布局通过将顶点进行编码，称之为编码前缀树（encoded trie）。具体而言，对于每个 $u \in V(Q_C)$，首先生成一个包含 u 的候选数据顶点的关系 R(u)，并将其存储为一个数组。这里需要注意的是，对于给定的 $u \in V(Q_C)$，所有包含属性 u 的关系共享同一个 R(u)。pos(v) 表示 v 在 R(u) 中的位置。对于每个边 $(u, u') \in E(Q_C)$, 其中 $u \in \mathcal{N}_+^{\phi}(u')$ ，根据 $(v, v') \in R(u, u')$ 进行遍历：如果 $v \in R(u)$ 且 $v' \in R(u')$，则将 $(\text{pos}(v), \text{pos}(v'))$ 添加到 $R'(u, u')$ 中；否则跳过 (v, v')。此外，对于 $v \in R(u) - \pi_{u'} R(u, u')$，将 $(\text{pos}(v), \emptyset)$ 添加到 $R'(u, u')$ 中以指示 v 在 $R'(u, u')$ 中没有邻居。然后将 $R'(u, u')$ 存储为一棵前缀树。对于每个边 $(u, u') \in E(Q_F)$，构建 $R(u, u')$ 的哈希索引以便进行哈希连接。

3.5 结果枚举

在结果枚举过程中采用了三种优化方法以提高效率。首先，在 LFTJ 中采用了两种集合交集（SI）方法。对于邻居集较大的小型查询，使用了 QFilter，该方法将集合编码为紧凑的布局。而对于邻居集经过过滤后较小的大型查询，则采用基于整数数组的混合方法。混合 SI 方法采用了融合了 Merge 方法和 Galloping 算法的方式来处理集合的基数偏差。同时，为了加速 Merge 和 Hybrid，采用了 AVX2 指令集（256 位宽度），分别标记为 M+AVX2 和 H+AVX2。QFilter 使用了 SSE 指令集（128 位宽度）。其次，对于小型查询，采用了交集缓存。最后，利用了失败集合剪枝来加速大型查询。

4 复现细节

4.1 与已有开源代码对比

本文复现了 Rapidmatch 算法实现子图查询的实验，本文在此基础上添加了一个脚本以实现自动输入不同大小的 Query graph 来进行循环测试，并将输出的结果存入 csv 文件。根据搜索结果记录所需的时间和搜索到的样例数，并根据所得数据绘制了表格。在实验的过程中，想到了可以在此基础上修改边上的权值以实现搜索带权无向图的搜索，此外，还有修改边重为 0 和 1 以标记正向和反向，实现有向无权图的搜索。但是修改了边上的权值后算法部分的代码需要进行大量调整，本文未能实现这部分的代码修改，但是这是以后可以改进创新的方向。

GitHub 地址如下 https://github.com/RapidsAtHKUST/RapidMatch

4.2 实验环境搭建

本项目在云服务器上搭建，主要使用 C++ 实现，根据 GitHub 开源地址，找到项目 readme 文件，按指引搭建。

4.3 界面分析与使用说明

代码文件说明：

· configuration/: 配置文件
· dataset/: 数据集
· graph/: 处理图数据代码
· matching/: 匹配算法
· preprocessing/: 预处理
· cmakelist/: 所需的主要 package

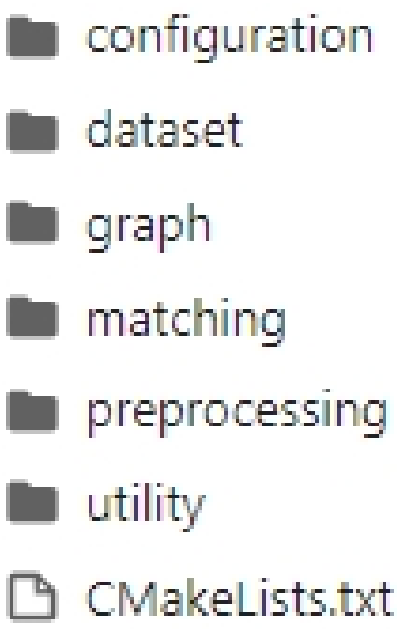

图 4. 所需文件

4.4 创新点

在复现子图查询代码时，每次只能查询测试输入一个 query graph 和一个 data graph，然后记录查询结果，其中包含的主要信息有查询到的同构嵌入数量和查询时间，鉴于一个个手动查询的效率太慢，于是写了一个脚本以实现循环输入数据库中的大图与小图。并将输出结果保存为一个 txt 文档，然后利用 python 脚本将 txt 文档中数据提取为 csv 文件，然后制图。

5 实验结果分析

实验测试了样本数据中的各种数据集，主要测试了其中 query-graph 中的较小的节点，测试结果如下图，根据图 5 可以读出，在 YouTube 数据集中，对于 8 节点的 query，搜索时间基本在 0.2s 以下，其他搜索时间长是由于同构嵌入较多或者节点频率在 data-graph 中太少，对于 4 节点 query，搜索时间在 0.1s 一下。除此之外，还测试了 human 数据集和 dblp 数据集 4 节点搜索时间。由于测试数据集过多，仅测试了 query 数据集较少的样例。

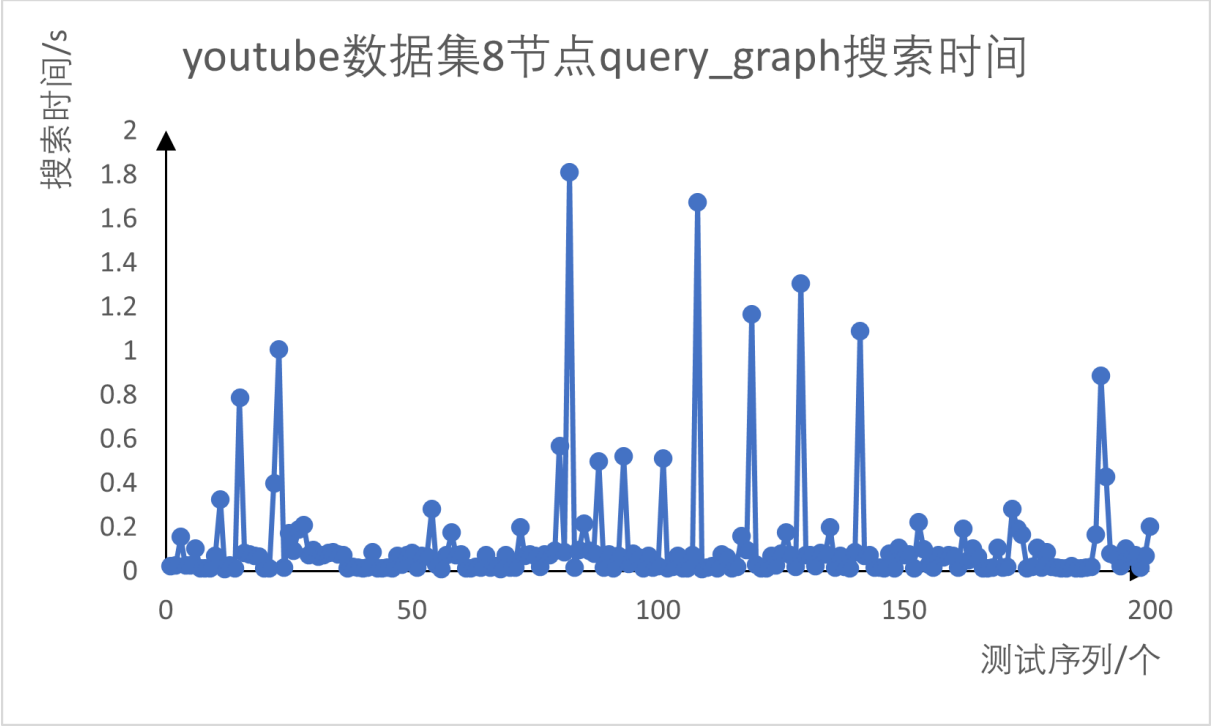

图 5. youtube 数据集 8 节点 query$_g raph$

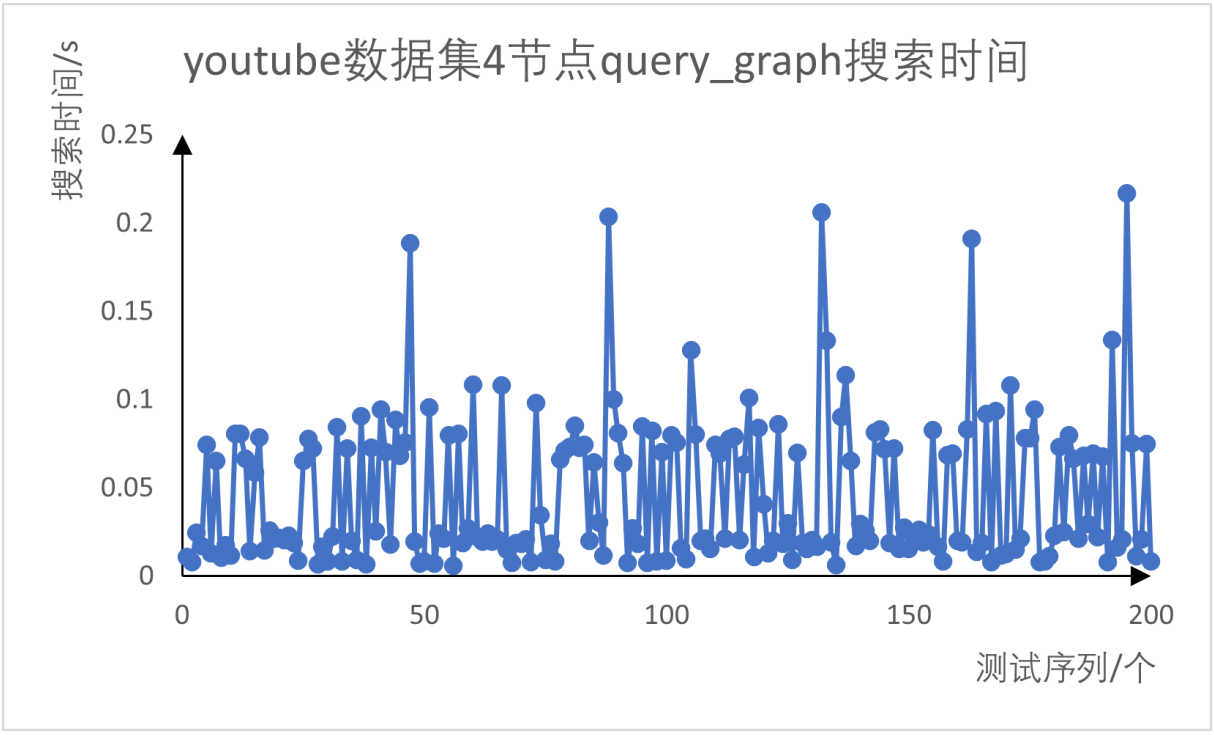

图 6. youtube 数据集 4 节点 query$_g raph$

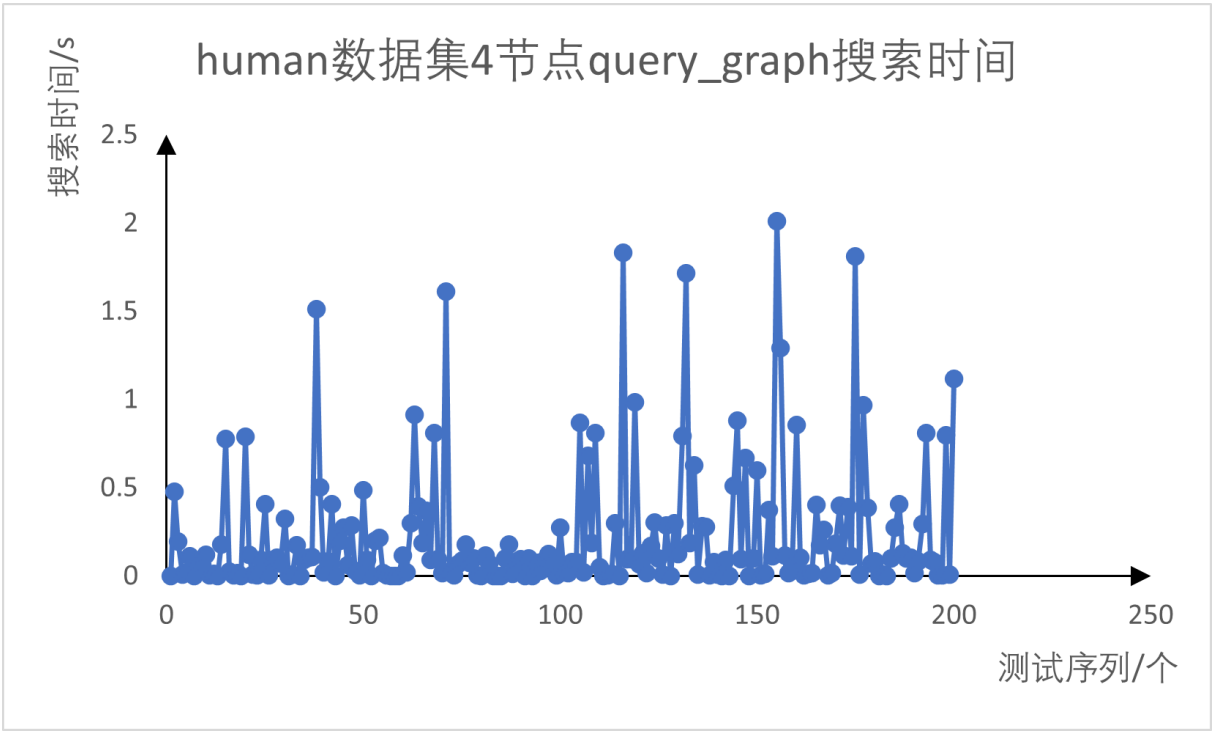

图 7. human 数据集 4 节点 query$_g raph$

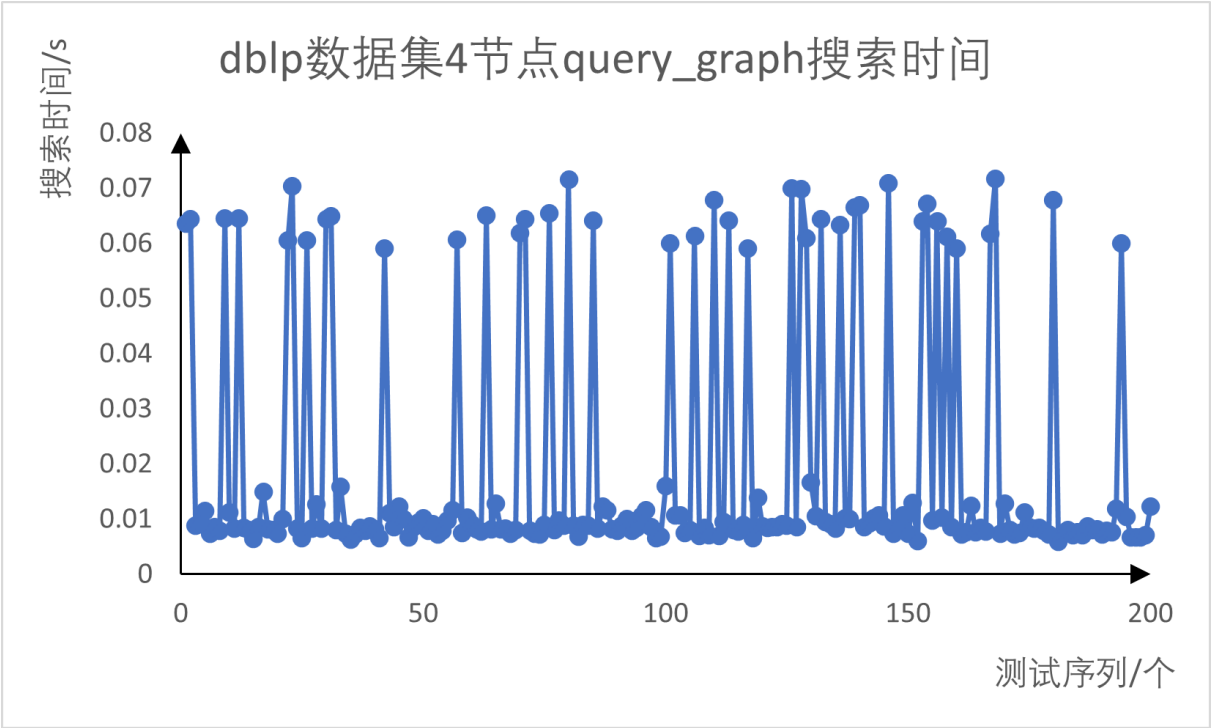

图 8. dblp 数据集 4 节点 query$_graph$

原论文还对各种相似的算法比较，结果如图 9，图中比较了不同算法的平均查询时间，在 wn 和 yt 数据集上，RM 的运行速度比竞争算法快了一个数量级以上，而在 db 数据集的查询上，DP 和 RM 的性能相近。

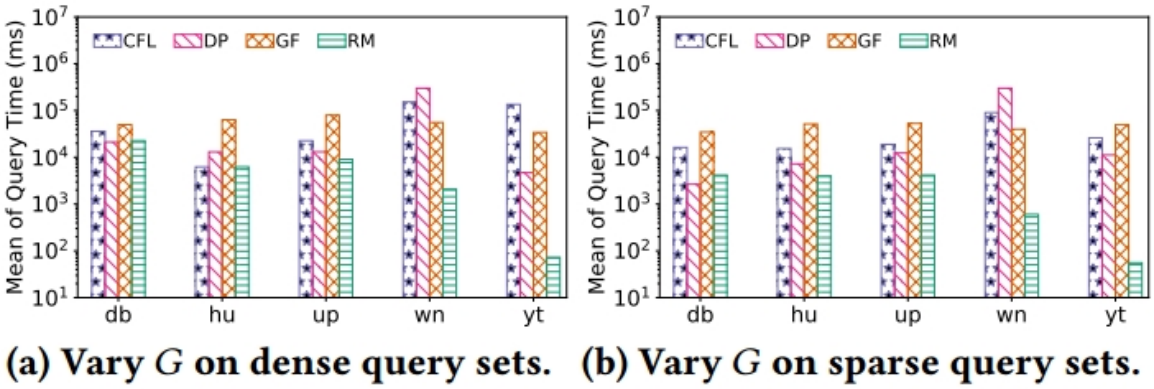

图 9. 不同密度 query 查询对比图

图 10 展现了查询集合中查询的预处理时间（虚线）和查询时间（实线）的累积分布函数。针对运行时间短的查询（即查询时间短的查询），预处理时间占主导地位。由于 RM 在 db 上的预处理时间比 CFL 长，因此在一些查询上，RM 的查询时间比 CFL 长。

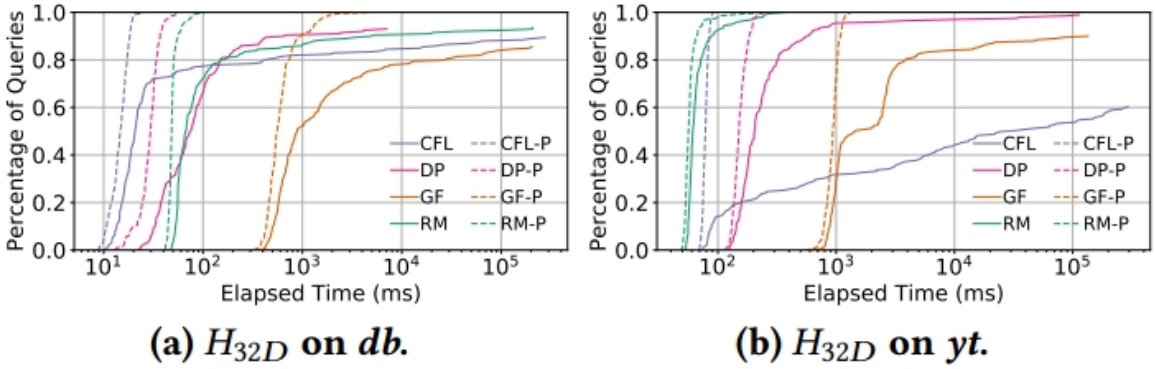

图 10. 个体查询性能分析

6 总结与展望

本文基于 RapidMatch 的算法做复现，它是一种综合基于连接的子图查询处理算法，该算法在过滤、匹配顺序生成和枚举方面进行了优化。研究中涵盖了基于探索和基于连接的子图查询处理算法，强调了 RapidMatch 的优越性能。该方法以关系运算符为基础，利用图结构信息来优化关系过滤和连接计划。通过广泛的实验证明，RapidMatch 在各种工作负载下均优于目前领先的基于连接和基于探索的方法。

在复现的过程中，根据其原理的解释，想到了两个可以改进的方向。

1、改变图数据中边重权值，在此基础上修改为查询无向带权图的子图查询 2、改变边的方向，在此基础上修改为查询有向无权图的子图查询

参考文献

[1] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1199–1214, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1429–1446, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. *Proc. VLDB Endow.*, 1(1):364–375, aug 2008.

[4] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. Rapidmatch: A holistic approach to subgraph query processing. *Proc. VLDB Endow.*, 14(2):176–188, oct 2020.

[5] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, jan 1976.

[6] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: Distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, page 192–203, New York, NY, USA, 2009. Association for Computing Machinery.