软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(5):2235−2256 [doi: 10.13328/j.cnki.jos.007493] [CSTR: 32375.14.jos.007493]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

大图数据的统一查询处理机制*

陈 迪[1], 袁 野[2], 潘雅妮[1], 王国仁[2]

[1](东北大学 计算机科学与工程学院, 辽宁 沈阳 110169)

[2](北京理工大学 计算机学院, 北京 100081)

通信作者: 袁野, E-mail: yuan-ye@bit.edu.cn

摘 要: 现实世界中许多应用场景都可以用图数据表示, 图上的查询也具有广泛的应用, 如可达、最短路径、关键字、图匹配、PageRank、SimRank、*k*-core、*k*-truss 和 Clique 等. 针对特定的查询问题, 目前的研究方法可概括为: 提出相应的查询处理算法, 并构建索引结构来加速查询. 然而, 现实应用中需求的多样化以及图数据规模爆炸式的增长为该研究方法带来了两方面挑战: 第一, 同一个图数据在应用中会涉及多种查询, 但针对不同查询问题的处理机制和索引结构均不相同, 因此在设计图数据库时需构建多个索引和相应的查询算法; 第二, 索引的规模通常比原图数据的规模大, 多个索引同时存在会占用大量的系统空间, 导致图数据库的性能急剧下降, 不能被真正地应用. 为应对上述挑战, 提出一种统一的查询处理机制, 即为大图数据构建统一且高效的索引结构, 并基于统一索引结构设计可达、最短路径、关键字和图匹配这 4 种查询处理算法. 为构建统一索引结构, 对大图数据进行划分, 并根据可达、最短路径、关键字和图匹配这 4 种查询的特点提取出图数据中的重要顶点, 该统一索引结构规模比图数据规模小, 并且能高效地支持上述 4 种查询. 最后, 通过在 4 组真实数据上的实验验证了统一索引结构和 4 种查询处理算法的高效性和扩展性.

关键词: 统一索引; 可达查询; 最短路径查询; 关键字查询; 图匹配查询

中图法分类号: TP311

中文引用格式: 陈迪, 袁野, 潘雅妮, 王国仁. 大图数据的统一查询处理机制. 软件学报, 2026, 37(5): 2235–2256. http://www.jos.org.cn/1000-9825/7493.htm

英文引用格式: Chen D, Yuan Y, Pan YN, Wang GR. Unified Query Processing Mechanism over Large-scale Graph Data. Ruan Jian Xue Bao/Journal of Software, 2026, 37(5): 2235–2256 (in Chinese). http://www.jos.org.cn/1000-9825/7493.htm

Unified Query Processing Mechanism over Large-scale Graph Data

CHEN Di[1], YUAN Ye[2], PAN Ya-Ni[1], WANG Guo-Ren[2]

[1](School of Computer Science and Engineering, Northeastern University, Shenyang 110169, China)

[2](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Graph data can represent a wide range of real-world application scenarios, and query processing over graphs plays a crucial role in various tasks, such as reachability, shortest path, keyword search, graph pattern matching, PageRank, SimRank, *k*-core, *k*-truss, and Clique. For specific query problems, existing approaches typically propose corresponding query processing algorithms and build index structures to speed up the query. However, the diversification of application demands and the explosive growth in graph data scale present two major challenges to this methodology. First, a single graph dataset may involve multiple types of queries in practice, yet each query type often requires distinct processing mechanisms and index structures. Consequently, multiple indexes and corresponding query algorithms need to be constructed when designing a graph database. Second, index structures are often larger than the original graph data, and maintaining multiple indexes simultaneously can lead to significant space overhead, resulting in sharp performance degradation and limited practical applicability. To address these challenges, this study proposes a unified query processing mechanism. A unified and

* 基金项目: 国家重点研发计划 (2022YFB2702100); 国家自然科学基金 (62225203, U21A20516)
收稿时间: 2025-01-15; 修改时间: 2025-04-11; 采用时间: 2025-06-13; jos 在线出版时间: 2025-12-17
CNKI 网络首发时间: 2025-12-25

efficient index structure is constructed for large-scale graph data, upon which four query processing algorithms are designed, supporting reachability, shortest path, keyword search, and graph pattern matching. To build the unified index structure, the graph data is partitioned, and important vertices are extracted based on the characteristics of the four queries. The resulting unified index is smaller in size than the original graph and efficiently supports all four queries. Finally, the effectiveness and scalability of the unified index and the proposed algorithms are validated through experiments on four real-world datasets.

Key words: unified index; reachable query; shortest path query; keyword search; graph matching query

近年来, 随着语义网络、社交网络、生物网络等新型领域的飞速发展, 数据的结构越来越复杂, 图作为一种特殊的数据存储模型, 更具有一般性表示能力[1]. 现实世界中的许多应用场景都需要用图数据表示, 比如, 传统应用中的最优运输路线的确定、科技文献的引用关系等; 新兴应用中的社交网络分析、人脑网络等都可以看作是大图数据的应用. 大图上的查询处理也有着广泛的应用, 人们为及时准确地从大图中查询信息并进行有效的分析, 提出了许多不同类型的查询问题. 例如可达、最短路径、关键字、图匹配、PageRank、SimRank、*k*-core、*k*-truss 和 Clique 等查询问题[2−6].

可达、最短路径、关键字和图匹配是最基础的查询, 覆盖了图查询中最核心的 4 大典型任务, 分别对应内容检索、结构匹配、连通性判定与路径优化. 这些查询类型广泛应用于知识图谱、社交网络、交通路径规划等实际场景, 具有良好的代表性和通用性, 现已存在很多算法用以解决这 4 种查询问题. Wu 等人[7]总结了 SILC 和 CH 等 4 种可达和最短路径查询算法, Zhang 等人[8]提出了构建索引 SPTI 加速最短路径查询. Bhalotia 等人[9]提出了可解决关键字查询的后向搜索算法, 并且可通过构建双层索引并采用双向搜索加速关键字查询.

He 等人[10]和 Cheng 等人[11]分别提出了 R-join 和 GraphQL 算法并构建相应的索引完成图匹配查询, Gao 等人[12]提出将模式图划分成新框架以加速图匹配查询. Moayed 等人[13]提出一种特征分解剪枝策略以加速图匹配查询.

由现有研究成果可以看出, 针对特定的查询问题, 人们倾向于提出相应的查询处理算法, 并构建不同的索引结构来加速查询, 是非统一的查询处理机制, 其框架如图 1 所示. 然而, 现实应用中需求的多样化以及图数据规模爆炸式的增长使得现有研究方法存在两方面挑战. 第一, 同一个图数据在应用中会涉及多种查询, 不同查询问题的处理机制和索引结构均不相同, 因此在设计图数据库时需构建多个索引结构和相应的查询算法; 第二, 索引的规模通常比原图数据的规模大, 多个索引同时存在会占用大量的系统空间, 导致图数据库的性能急剧下降, 不能被真正应用.

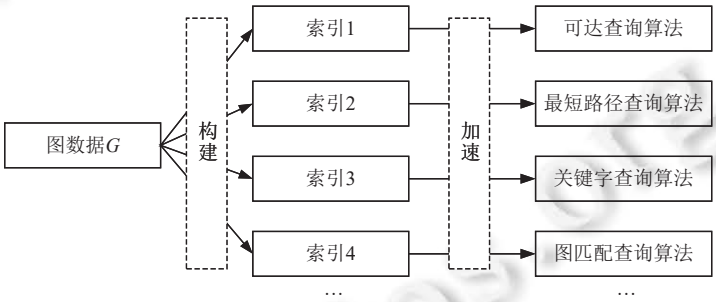


图 1 非统一查询处理机制框架

为解决上述挑战, 本文提出了一种统一的大图数据查询处理机制, 其框架如图 2 所示, 即为大图构建统一且高效的索引结构, 并基于统一索引结构设计了可达、最短路径、关键字和图匹配这 4 种查询算法.

但现有的索引和查询算法都是针对某一查询问题提出的, 它们在结构和处理机制上均不相同, 不能直接应用在统一的查询处理机制中. 因此, 本文首先基于图数据的疏密性对大图进行划分, 并根据可达、最短路径、关键字和图匹配这 4 种查询的特点提取出各划分区域的重要顶点来构建统一索引结构. 为保证该统一索引结构的空间复杂度小, 并可高效支持上述 4 种查询, 其由中心索引、临界顶点集合, 以及关键字的倒排索引这 3 部分构成. 除此之外, 本文基于统一索引结构设计了上述 4 种查询相应的处理算法, 分别加速了基于宽度优先遍历搜索的可达和最短路径查询算法、基于双向搜索思想的关键字查询算法以及先匹配模式子图再连接取并集的图匹配算法.

本文的主要贡献如下.

(1) 提出了统一的大图数据查询处理机制, 即基于统一索引结构便可解决大图上的多种查询问题.

(2) 构建了统一的索引结构, 其规模比原图数据要小, 且能支持可达、最短路径、关键字和图匹配这 4 种查询.

(3) 基于统一索引结构, 设计了可达、最短路径、关键字和图匹配这 4 种查询算法.

(4) 通过在 4 组真实数据集上的实验验证了统一索引结构和查询处理算法的高效性与扩展性.

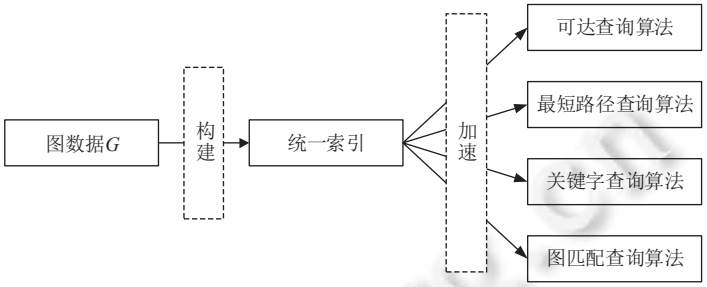


图 2 统一查询处理机制框架

本文第 1 节介绍相关工作. 第 2 节介绍大图和本文重点研究的 4 种查询问题的基础知识, 并给出本文研究内容的问题定义. 第 3 节介绍大图划分和统一索引结构的构建. 第 4 节介绍基于统一索引结构设计的可达、最短路径、关键字和图匹配这 4 种查询算法. 第 5 节介绍在 4 组不同数据集上进行实验并分析结果. 第 6 节进行工作总结.

1 相关工作

由于最短路径查询算法普遍也可解决可达查询, 因此本节将对现有的大图压缩技术以及最短路径、关键字和图匹配这 3 种查询算法做简要介绍与对比.

1.1 大图压缩技术

目前人们倾向于存储预处理信息或者建立索引来压缩大图数据. 大图的存储压缩主要包括对邻接链表和邻接矩阵的压缩, 或者结合图的特征进行压缩.

Claude 等人[14]提出了 Re-Pair 算法完成对邻接链表的压缩, 该算法将邻接链表存储为一个字符序列, 重复在这个字符序列中寻找最频繁的字符对, 用新字符替换频繁字符对, 并将新字符和频繁字符对的映射关系存储到字典中, 直到字符序列中所有字符仅出现一次停止替换. k^2 树算法[15]可对邻接矩阵进行压缩, 该算法将邻接矩阵等分成 k^2 个正方形, 每个正方形中都含有叶顶点和内顶点, 每个顶点都用 1 bit 来表示, 使得 k^2 树中存在很多数值为 0 的内顶点, 从而会减少内存空间. 压缩邻接矩阵或者邻接链表这些方法可以减少对内存的消耗, 但这些方法并没有利用到图数据本身的特点, 很难做到有效地提高查询效率.

结合图特征压缩后形成的预处理信息或是索引结构均可明显加快算法的执行速度, 越来越多的研究人员开始重视这一领域的研究内容. 现有 3 种结合图特征的压缩方法: 其一是基于聚合进行压缩, 其使用超级顶点和超级边来构建索引, 有助于理解和可视化复杂图数据, 并可以减少存储内存; 其二是基于属性进行压缩, 利用拓扑结构和大图中点边之间的相关属性形成小规模图, Rossi 等人[16]提到了一种将大图分解成多个团以减少大图中边的数量的方法; 其三是采用编码技术进行压缩. 其中最适合我们所研究的带有标签的有向加权无环图的解决方法是上述压缩方法中的第 2 种. Yu 等人[17]提出的 BL 图索引技术只需要线性的时间和空间. 使用 BL 可以在不到 16 ms 的时间内回答平均有 160 万个节点的图数据的可达性查询. 但是其压缩后的小规模图仅适合进行可达查询, 并不适合于最短路径、关键字以及图匹配这 3 种查询. 本文提出的统一大图索引技术所压缩后的小规模图将同时适用于以上 4 种图查询.

1.2 最短路径查询算法

现已存在很多经典的最短路径查询算法, 比如对于无权图数据而言, 即边的权重始终为 1, 则可以采用广度优先遍历的方法进行最短路径查询; 再如对于有权图数据而言, Dijkstra 算法和 Floyd 算法均被广泛用于解决最短路

径查询问题. 然而, 这些经典的最短路径查询算法都需在内存中得以实现, 当图规模很大时, 这些算法则不再适用, 它们无法支持大规模图的在线查询.

为解决上述问题, 研究者们提出了一个重要思路, 即先离线预处理存储图中部分最短路径信息, 待在线查询时便基于预处理信息给出最终的查询结果. 例如, Agrawal 等人[18]对存储了大图中所有核心顶点间的最短路径进行预处理. 除此之外, Yang 等人[19]还提出了一种基于顶点优先排列的方法来存储顶点间的最短路径. Wang 等人[20]通过草图查询离线标记 (即预先计算的标签) 高效地引导在线搜索. 然而, 存储大量的预处理信息会消耗过多的内存空间从而加重存储代价, 因此近年来人们一直在研究以减少预处理信息为目标的最短路径查询算法. 例如, TEDI[21]则提出将图分解为树结构, 树上每个顶点代表图顶点的一个子集, 查询算法会并行预处理存储每个树顶点中各图顶点到彼此的最短路径. Wu 等人[7]总结了路网中用于解决最短路径的查询算法, 并将其分为两类, 其一是基于空间相关性的算法, 如 SILC 算法和 PCPD 算法; 其二是基于顶点重要性的算法, 如 CH 算法和 TNR 算法. 其中基于顶点重要性的算法不仅会选取出较为重要的地标顶点, 也会忽略一些并不重要的顶点. 除此之外, Zhang 等人[8]提出了最短路径树桩 SPT 和最短路径树桩索引 SPTI 来加快最短路径查询.

总之, 现有的高效最短路径查询算法都是基于预处理信息来加速的. 人们仍在进一步研究如何减少预处理信息的存储代价, 以及如何更多地利用最短路径本身的一些固有结构特征.

1.3 关键字查询算法

最为基础的关键字查询算法便是基于 Dijkstra 算法进行后向搜索查询, 如 Bhalotia 等人[9]提出了 BANKS 算法, 其主要思想是: 首先定义了顶点集合 E_i, 其存放可达关键字 k_i 的顶点, 初始化后的 E_i 仅存放含有关键字 k_i 的顶点; 然后在后向搜索的过程中更新顶点集合 E_i, 在每一轮的搜索过程中, 选取集合 E_i 中到关键字 k_i 距离最近且未被访问过的顶点 v, 将顶点 v 后向广度优先搜索一步访问到的新顶点添加到 E_i 中; 最后的返回结果是所有顶点集合中共同含有的顶点.

后向搜索算法虽然易于理解, 但难扩展到大规模图上, 双向搜索算法 BLINKS[22]可加速关键字匹配, 其主要的思想是: 首先从含有关键字 k_i 的顶点或可达关键字 k_i 的顶点后向搜索一轮访问新顶点; 然后每访问到一个新顶点 u 时, 便前向搜索查询该顶点到其他关键字的最短距离 $dist_j$ ($1\leqslant j\leqslant m$, $i\neq j$), 其中 m 为关键字的个数, 并计算以 u 为根顶点的评价函数, 若此时根顶点 u 的评价函数小于当前最优结果的评价函数, 则把该顶点记作当前最优结果; 最后待达到搜索的停止条件时, 返回当前的最优结果. He 等人[10]还提出了划分大图数据并构建双层索引结构加速双向搜索查询. 除此之外, 为加快关键字查询, Jiang 等人[23]提出了一种通用本体索引框架 BiG-index, 通过迭代式构建多层次的摘要图 (summary graph) 索引, 结合本体引导的查询重写与早期剪枝策略, 在保证查询语义准确性的同时, 有效缩小了搜索空间, 大幅提升了关键词查询的效率, 实验结果显示对现有方法如 BLINKS 具有显著加速效果. Ghanbarpour 等人[24]提出一种最小覆盖 r-clique 对现有模型进行扩展, 解决了现有算法中存在冗余节点的问题并能够以分布式方式执行.

1.4 图匹配查询算法

图匹配查询问题根据查询语义的不同, 可分为图同构匹配查询和图模拟匹配查询. 图匹配问题可分为图模式匹配和图模拟匹配. 图模式匹配是在图数据 G 中找到与模式图 P 同构的所有子图, 是 P 中边与 G 中边匹配的问题, 其为 NP 难问题. 图模拟匹配试图找到语义或概念上与模式图相似的子图, 是 P 中边与 G 中路径匹配的问题, 模拟匹配降低了整体匹配的复杂度, 可在多项式时间内解决[25], Bouhenni 等人[26]给出了不同的图模拟匹配算法.

本文将重点研究同构匹配查询, 现已存在很多优秀的模式匹配算法. 最为经典的是 1976 年提出的 Ullmann 算法[27], 该算法采用深度优先遍历的方法, 逐一枚举出与模式图同构的子图, 在遍历的过程中通过检查匹配点对的邻接顶点来进行剪枝, 尽早识别出不可匹配的顶点. 接下来 VF2 算法[28]在 Ullmann 算法的基础上加以改进, 也可用于解决精确图匹配问题. 还有许多查询算法通过建立索引来加速匹配过程, 比如 GraphGrep 算法[29]将带有语义信息的顶点编码后建立索引, 还可以基于简单的路径、树以及其他简单的结构对图数据中的每个节点进行编码描述来建立索引. Spath 算法[30]为原数据图和模式图中的每一个顶点建立到该点最短距离小于 k 的邻居签名, 从而来

描述每个顶点的局部信息, 根据图数据和模式图中的邻居签名可有效地对候选点进行裁剪, 从而加快匹配速度.

除此之外, Han 等人[31]提出了 Turbo 算法用于加快精确图匹配查询, 该算法涉及候选匹配区域探索和结果合并两个过程, 其首先对匹配区域进行有效排序, 然后基于邻居等价集合来选择匹配的顶点, 一定程度上可保证尽早进行剪枝, 从而加快查询效率. Carletti 等人[32]近些年又提出了 VF3 算法, 可加快巨大并稠密的大图数据中的精确图匹配查询. 还可采用子图结构来建立索引, 通过数据挖掘技术提取出图中的频繁子图从而建立索引, 能够很大程度上降低索引结构的规模. Sun 等人[33]提出了结合候选顶点及其邻居构建 BI (bigraph index) 索引, 基于此索引提出高效算法 VC, 在子图匹配过程中高效支持邻居检索与剪枝, 显著提升了子图同构查询的性能.

上述算法都是针对相应的查询问题提出的, 其中涉及的图划分策略和索引的构建策略均不相同, 无法直接应用到统一的大图查询处理机制中.

2 问题定义

本节首先给出图的定义, 随后给出可达、最短路径、关键字和图匹配这 4 种查询问题的定义, 最后给出本文研究内容统一查询处理机制的问题定义.

定义 1. 图数据. 在图论中, 带有标签的有向图可用 $G=(V, E, L, \Sigma)$ 表示. V 表示图中顶点的集合. E 表示图中边的集合, 对于图中任意边 $e \in E$, 可表示为 (u, v), 其中顶点 $u, v \in V$, 即顶点 u 可达顶点 v. L 为作用于顶点集合 V 上的标签函数, 对于顶点集 V 中的任意顶点, $L(v)$ 即为顶点 v 的标签, Σ 为标签集合. 顶点上的标签可呈现该顶点的性质, 如关键字、等级和社会角色等, 并且每个顶点上的标签不唯一, 其拥有一个标签集合. 例如, 图 3 所示, 顶点 8 含有标签 $\{e, f, g\}$.

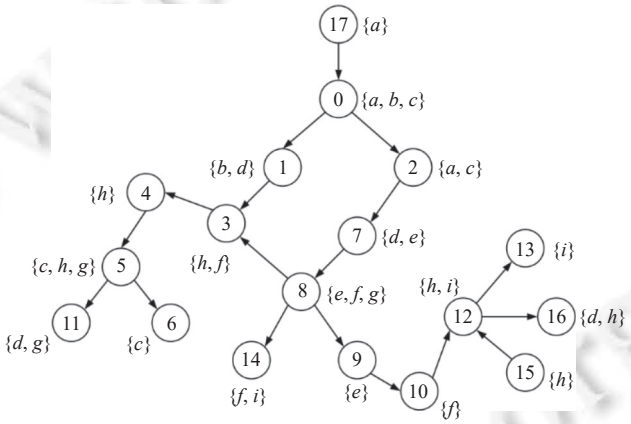


图 3 有向图 G

定义 2. 可达查询. 给定一个有向图 G 和图中的两个顶点 u 和 v, 查询 u 能否到达另一个顶点 v, 即是否存在一条以 u 为起始顶点, 以 v 为终止顶点的路径, 是一种布尔类型的查询.

例如, 图 3 中顶点 1 和顶点 5 的可达查询返回 true, 顶点 1 和顶点 9 的可达查询返回 false.

定义 3. 最短路径查询. 给定一个有向图 G 和图中的两顶点 u 和 v, 最短路径查询需在顶点 u 和顶点 v 的所有可达路径中找到距离和最小的那条路径, $D_s(u, v)$ 记为最短路径的路径长度.

如果查询的两顶点间仅有一条可达路径, 那该条路径即为最短路径. 例如, 图 3 中假设每一条边距离为 1, 则 $D_s(8, 5) = 3$.

定义 4. 关键字查询. 给定一个有向图 G 和一组关键字 $q_k = (w_1, w_2, \ldots, w_m)$ 后, 关键字查询需在 G 中查询可达这组关键字的顶点并返回 $t =< r, (n_1, n_2, \ldots, n_m) >$, 其中 r 和 n_i 是图 G 中的顶点, 这些顶点需要满足如下性质:

(1) 覆盖性: 对于每一个 i ($1 \leqslant i \leqslant m$) 来说, 顶点 n_i 的标签集合中含有关键字 w_i.

(2) 连接性: 对于每一个 i ($1 \leqslant i \leqslant m$) 来说, 顶点 r 在图 G 中可达顶点 n_i.

其中, r 为查询结果的根, n_i 是查询结果对 q_k 中每个关键字 w_i 的匹配. 由上述的连接性可知查询结果是一棵子树, 其根顶点 r 可达的顶点包含了待查询的所有关键字. 例如, 我们在图 3 所示的 G 中查找关于关键字组 q_k=(b, f) 时, 可得到 t_1 =< 1,(1,3) > 和 t_2 =< 0,(0,8) > 等多种查询结果.

关键字匹配查询往往会返回多个查询结果 t_i, 而一般情况仅需最优或 top-k 结果, 因此需对返回的 t_i 进行评价, 评价函数 $S(t)=\sum_{i=1}^{m} D_s(r,n_i)$, 其中 t =< r,(n_1,n_2,…,n_m) >, $D_s(r, n_i)$ 为在图 G 中根顶点 r 到含有关键字 w_i 顶点 n_i 的最短路径的距离, 评价函数值越小, 其结果越优. 例如, 对于上文中 t_1 =< 1,(1,3) > 和 t_2 =< 0,(0,8) > 这两种查询结果而言, $S(t_1)$ = 1, $S(t_2)$=3, 则可说 t_1 的结果优于 t_2.

定义 5. 图匹配查询. 给定图 G 和模式图 q_m=(V_q, E_q, L_q), 返回的子图 g 与模式图 q_m 图同构. 若子图 g=(V_g, E_g, L_g) 同构于模式图 q_m, 则存在一种映射关系 f, 使得模式图 q_m 中的顶点可一一映射到子图 g 中的顶点, 即 $f: V_q \to V_g$, 且满足如下两个条件.

(1) 顶点映射: 对于模式图 q_m 中任意顶点, 其标签与映射顶点的标签相同, 即 $\forall v \in V_q$, $L_q(v) = L(f(v))$.

(2) 边映射: 对于模式图 q_m 中的每一条边 (u, v), 顶点 u 和顶点 v 映射的顶点在子图 g 中也存在一条边, 即 $\forall e=(u,v) \in E_q$, $(f(u), f(v)) \in E_g$.

定义 6. 统一的大图查询处理机制. 当给定图数据 G 后, 构建统一索引结构, 并基于统一索引结构实现图 G 中可达、最短路径、关键字和图匹配这 4 种查询问题.

3 统一索引的构建

本节将介绍统一索引结构的构建, 主要思想是: 首先基于疏密性对图数据进行划分; 然后根据可达、最短路径、关键字和图匹配这 4 种查询的特点提取出各划分区域的重要顶点; 最后基于划分和提取出的重要顶点构建统一的索引结构.

3.1 图划分算法

映射现实世界的大图数据全局来看是稀疏的, 而局部来看却是稠密的[10], 因此本文基于疏密性对图数据进行划分. 基于疏密性划分后, 每一划分区域中存在一些重要顶点, 它们决定着该区域的拓扑结构[8], 本文在划分后还需记录这些重要顶点, 供统一索引结构构建时使用.

每一划分区域是由未被划分的顶点 u, 以及与顶点 u 距离小于等于 ε 且未被划分的顶点集合构成. 其中, 顶点 u 为该划分区域的中心顶点; 该划分区域中入度大于 0 且存在顶点来源于其他划分区域的顶点为入边界顶点; 该划分区域中出度大于 0 且存在顶点指向其他划分区域的顶点为出边界顶点.

当所有顶点都有所属的划分区域后则停止划分, 得到划分覆盖 $P=(P_1,P_2,\ldots,P_k)$, k 为划分区域的个数, 并且满足 $\bigcup P_i = V$, 当 $i \neq j$ 时, $P_i \cap P_j = \varnothing$. 为得到最小划分覆盖, 本文的划分策略是: 总是选取当前不属于任何划分区域且度最大的顶点作为新划分区域的中心顶点, 这里的度是指出度与入度的和, 并从该顶点开始 ε 步的双向广度优先遍历得到新的划分区域. 因此在划分前, 需将图 G 中的顶点按照度的大小进行排序, 以加速启发式地生成最小划分覆盖.

图划分算法 *GD* (graph division) 的流程如算法 1 所示. 第 1 行对结果初始化. 第 2 行将图 G 中的顶点按照度排序并存放在队列 Q 中. 第 3–12 行生成以 u 为中心顶点的划分区域 P_i. 第 5 行将顶点 u 添加到中心顶点集合 C 和当前划分区域 P_i 中. 第 6–10 行将从 u 双向广度优先遍历并找到距离小于 ε 的新顶点添加到 P_i 中. 第 11 行将新得到的划分区域 P_i 压入近优划分覆盖 P 中; 直到 Q 为空停止循环, 得到近优的划分覆盖 P. 第 13 行在将遍历图 G 并获得各划分区域的入边界顶点集合和出边界顶点集合. 图划分算法 *GD* 结束后, 便直接返回近优划分覆盖 P、中心顶点集合 C、出边界顶点集合 *OUT* 和入边界顶点集合 *IN* 这 4 部分内容.

算法 1. 图划分算法 *GD*.

输入: 图数据 G, 非负整数 ε;
输出: 近优划分覆盖 P, 中心顶点集合 C, 入边界顶点集合 *IN*, 以及出边界顶点集合 *OUT*.

1. $P \leftarrow \varnothing$, $C \leftarrow \varnothing$, $IN \leftarrow \varnothing$, $OUT \leftarrow \varnothing$;
2. $Q \leftarrow$ all nodes in G sorted by degree;
3. WHILE Q is not empty DO
4. $u \leftarrow Q.pop()$;
5. $C.push(u)$, $P_i.push(u)$;
6. FOR each v that u can visit in ε distance DO
7. IF $v \in Q$ THEN
8. $P_i.push(v)$, $Q.remove(v)$;
9. END IF
10. END FOR
11. $P.push(P_i)$;
12. END WHILE
13. OUT, $IN \leftarrow$ get important points of every parts;
14. RETURN P, C, IN, OUT.

算法 1 中, 第 13 行遍历图 G 并提取出重要顶点的过程为: 得到近优划分覆盖 P 后, 逐一遍历每一划分区域 P_i 中的顶点 v, 根据当前顶点 v 的入度和出度以及相应的边, 判断顶点 v 是否属于出边界顶点集合或入边界顶点集合. 若顶点 v 入度大于 0 且存在属于其他划分区域的入顶点, 则将顶点 v 添加到该划分区域 P_i 的入边界顶点集合 IN_i 中, 同理若顶点 v 出度大于 0 且存在属于其他划分区域的出顶点, 则将顶点 v 添加到该划分区域 P_i 出边界顶点集合 OUT_i 中. 待所有划分区域都被遍历过后, 便提取出所有的重要顶点, 即 OUT 为 OUT_i 的集合, IN 为 IN_i 的集合.

例如, 给定如图 3 所示的图 G, 并给定 $\varepsilon = 4$, 为得到近优划分, 首先将图中顶点按照度排序得到{8, 12, 0, 3, 5, 1, 2, 4, 7, 9, 10, 6, 11, 13, 14, 15, 16, 17}, 从顶点 8 开始前向遍历并后向遍历距离 4 以内, 可得到以顶点 8 为中心顶点的划分区域 P_1 为{8, 3, 4, 14, 9, 10, 2, 7}. 需注意顶点 3 已经属于划分区域 P_1, 应从排序队列中移除. 接着分别从顶点 12, 顶点 0 和顶点 5 开始遍历, 得到了如图 4 所示的图数据 G 的近优划分覆盖, 虚线框住的为一个划分区域, 共有 4 个划分区域 P_1、P_2、P_3 和 P_4, 并得到中心顶点集合 C, 各划分区域中边框加粗的顶点为该划分区域的中心顶点, 即中心顶点集合 C={8, 12, 0, 5}.

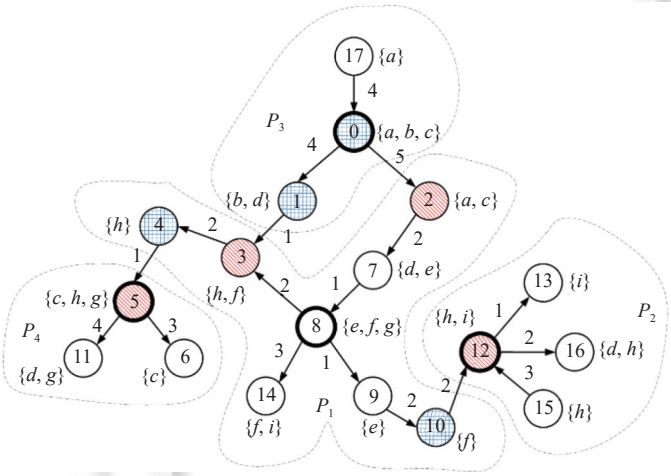


图 4 近优划分覆盖

在得到近优划分覆盖后, 逐一去遍历各划分区域中的顶点, 例如在遍历划分区域 P_1 时, 顶点 3 的入度为 2, 并且其入顶点 1 属于其他划分区域, 则顶点 3 为入边界顶点. 遍历完各划分区域后, 得到了图 G 的出边界顶点集合 OUT

和入边界顶点集合 IN. 最终提取重要顶点的结果如图 4 所示, 每一划分区域中, 十字交叉背景的顶点为出边界顶点, 斜线背景的顶点为入边界顶点, 即入边界顶点集合 $IN=\{\{2,3\},\{12\},\varnothing,\{5\}\}$, 出边界顶点集合 $OUT=\{\{4,10\},\varnothing,\{1,0\},\varnothing\}$.

3.2 构建索引

本文提出的统一索引结构由 3 部分组成: 其一是基于中心顶点集合建立中心索引, 其二是基于各划分区域的重要顶点得到临界顶点集合, 其三是基于最小划分覆盖建立关键字的倒排索引.

中心索引是一个索引图 $G_{CI}(V_{CI}, E_{CI})$, 可用于加速可达、最短路径和关键字的查询. 其顶点集合 V_{CI} 即为算法 1 得到的中心顶点集合 C, 为保证不破坏图 G 中原有的可达性, 需为这些顶点添加带有准确权值的有向边. 根据上述划分策略可知, 两个可达的中心顶点的距离不大于 2ε. 因此, 生成中心索引图边集合 E_{CI} 的主要思想是: 对于集合 V_{CI} 中任意顶点 u 做距离 2ε 的前向广度优先遍历, 若在 2ε 距离内访问到其他中心顶点 v, 则说明顶点 u 和顶点 v 在图 G 中可达, 需为 u 和 v 添加一条边, 即 $(u, v)\in E_{CI}$, 并且该条边的权值是 u 到 v 的最短路径距离. 例如, 图 3 的中心索引图 G_{CI} 如图 5 所示.

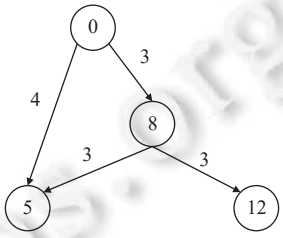
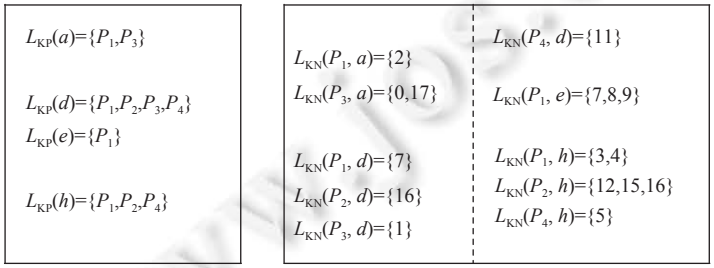


图 5 中心索引

临界顶点集合 B, 采用 *key-value* 的形式存储第 3.1 节中提取出的入边界顶点集合和出边界顶点集合, 可用于加速图匹配查询. 其中 *key* 为入边界顶点集合和出边界顶点集合中的顶点, *value* 可取{1, 2, 3}这 3 个整数中的任意一个, 其中 1 代表顶点 *key* 为入边界顶点, 2 代表顶点 *key* 为出边界顶点, 3 代表顶点 *key* 既为入边界顶点又为出边界顶点. 待图匹配查询时, *value* 中存放的值可以给出当前划分区域应向哪个方向进行扩展. 例如, 基于图 4 的近优划分结果和提取出的重要顶点, 可以得到图 3 的临界顶点集合 B={<2, 1>, <3, 1>, <5, 1>, <12, 1>, <0, 2>, <1, 2>, <4, 2>, <10, 2>}.

倒排索引 II (inverted index), 也采用 *key-value* 的形式存储标签的分布情况, 可用于加速关键字和图匹配的查询速度. 倒排索引 II 包含两部分, 其一是关键字与划分区域 (KP) 映射的关系列表 $L_{KP}(k)$, 记录着含有关键字 k 的划分区域, 其中 *key* 为关键字, *value* 为划分区域, 基于图 4 的划分结果可得到如图 6(a) 所示的 $L_{KP}(k)$; 其二是各划分区域中关键字和顶点 (KN) 的关系列表 $L_{KN}(P, k)$, 记录着划分区域 P 中含有关键字 k 的顶点, 其中 *key* 为划分区域和关键字的二元组, *value* 为顶点, 基于图 4 的划分结果可得到如图 6(b) 所示的 $L_{KN}(P, k)$.



(a) $L_{KP}(k)$ (b) $L_{KN}(P, k)$

图 6 倒排索引

3.3 复杂度分析

构建统一索引结构的时间复杂度主要包括 5 个部分, 总体可以简化为图中顶点数 N_V 与边数 N_E 的函数: 第一,

在图划分前需对图中所有顶点按照度值进行排序, 复杂度为 $O(N_V \cdot \log N_V)$; 第二, 执行图划分算法 (算法 1), 每个顶点在 ε 步广度优先遍历中最多访问一个邻域, 其复杂度在最坏情况下为 $O(N_V + N_E)$; 第三, 执行中心索引构建算法 (算法 2), 对所有顶点进行局部遍历以生成中心索引图, 最坏情况下每个顶点都需访问整个图, 则其复杂度最多为 $O(N_V \cdot (N_V + N_E))$; 第四, 临界顶点集合的提取与处理时间复杂度可视为 $O(N_V)$ 因其最多只需遍历所有顶点一次; 第五, 倒排标签索引的构建为线性扫描过程, 时间复杂度为 $O(N_V)$. 综上, 构建统一索引的总体时间复杂度为: $O(N_V \cdot \log N_V + N_V + N_E + N_V \cdot (N_V + N_E))$, 即最终简化为: $O(N_V \cdot (N_V + N_E))$.

统一索引的空间复杂度主要由存储中心索引图、临界顶点集合以及倒排标签索引这 3 部分构成. 第一, 中心索引图在最坏情况下需存储所有顶点之间的连边, 空间复杂度为 $O(N_V^2)$; 第二, 临界顶点集合通过对每个顶点打标记存储, 消耗 $O(N_V)$ 的空间; 第三, 倒排标签索引存储时需要记录每个标签对应的划分区域, 以及划分区域中对应的顶点, 同样需遍历所有顶点及其标签, 整体空间复杂度为 $O(N_V)$. 综上, 构建统一索引的总空间复杂度为 $O(N_V^2 + N_V)$.

需要指出的是, 在实际应用中, 图数据通常是稀疏的, 中心顶点集合远小于总体顶点数, 实际的时间和空间开销远低于上述最坏情况.

4 查询算法

本节将分别介绍基于统一索引设计的可达、最短路径、关键字和图匹配这 4 种查询处理算法.

4.1 可达查询算法

根据上述划分算法可知, 图 G 中任意顶点 u, 在广度优先遍历 2ε 步后, 至少会访问到一个中心顶点. 因此, 当给定数据图 G 和图中的两顶点 u, v 后, 可达查询算法的主要思想是: 先判断顶点 u 和顶点 v 是否在同一划分区域, 若在同一划分区域, 则由顶点 u 广度优先遍历 ε 步, 若访问到顶点 v 则直接返回可达, 否则直接返回不可达; 若不在同一划分区域, 则由顶点 u 前向遍历 2ε 步并记录下访问到的中心顶点并构成集合 U^*, 再由顶点 v 反向遍历 2ε 步并记录下访问到的中心顶点并构成集合 V^*, 最终通过中心索引来查询集合 U^* 和集合 V^* 中是否存在可达的顶点对, 从而返回最终结果.

可达查询算法 *RQUI* (reachability query on unified index) 的流程如算法 2 所示, 第 1 行将存放中心顶点的集合 U^* 和 V^* 初始化. 第 2–9 行讨论当顶点 u 和顶点 v 在同一划分区域的情况. 第 3–7 行从顶点 u 开始广度优先遍历 2ε. 第 4 行若访问到顶点 v 返回 true, 否则返回 false. 第 10 行记录下由顶点 u 前向遍历 2ε 步内访问到的中心顶点. 第 11 行记录下顶点 v 反向遍历 2ε 步内访问到的中心顶点. 第 12–18 行用于查找集合 U^* 和集合 V^* 中是否存在可达的一对顶点, 若存在则返回 true, 否则返回 false.

算法 2. 可达查询算法 *RQUI*.

输入: 图 G, 顶点 u, 顶点 v, 中心索引图 G_{CI};
输出: 顶点 u 是否可达顶点 v.

```
1. U* ← ∅, V* ← ∅;
2. IF u, v in the same partition THEN
3.     FOR each w that u visits in 2ε steps DO
4.         IF w == v THEN
5.             RETURN true;
6.         END IF
7.     END FOR
8.     RETURN false;
9. END IF
10. U* ← center vertices that u visits in 2ε forward BFS;
```

11. $V^* \leftarrow$ center vertices that v visits in 2ε backward BFS;
12. FOR each $u^* \in U^*$, each $v^* \in V^*$ DO
13. IF u^* can reach v^* in G_{CI} THEN
14. RETURN true;
15. END IF
16. END FOR
17. RETURN false.

例如, 在图 3 所示的图 G 中查询顶点 15 和顶点 3 是否可达, 并且给定 ε=2, 首先判断出这两个顶点不在同一划分区域, 则由顶点 15 前向广度优先遍历 4 步, 记录下访问到的中心顶点集合 $U^*=\{12\}$, 再由顶点 3 反向广度优先遍历 4 步, 记录下访问到的中心顶点集合 $V^*=\{0,8\}$, 然后通过图 5 所示的中心索引图 G_{CI} 可得到, 顶点 12 不可达顶点 0 和顶点 8, 因此该查询结果为 false. 可达查询算法的时间复杂度最差情况下由两部分构成: 首先从顶点 u 和 v 开始的 2ε 步广度优先遍历消耗的时间为 $O(N_{2\varepsilon}(u)+E_{2\varepsilon}(u))$, 其中 $N_{2\varepsilon}(u)$ 和 $E_{2\varepsilon}(u)$ 是顶点广度优先遍历 2ε 内访问到的顶点数和边数; 其次在中心索引图中查询是否存在可达的中心顶点对时需消耗的时间为 $O(N_{E_{CI}})$, $N_{E_{CI}}$ 为中心索引图中的总边数. 即最差情况下的时间复杂度为 $O(N_{2\varepsilon}(u)+E_{2\varepsilon}(u)+N_{E_{CI}})$.

4.2 最短路径查询算法

当给定图 G 和图中的两顶点 u、v 后, 最短路径查询算法同可达查询算法 *RQUI* 的实现相似, 都是基于统一索引中的中心索引图 G_{CI} 得以加速. 其首先判断顶点 u 和顶点 v 是否在同一划分区域, 若在同一划分区域, 则由顶点 u 广度优先遍历 2ε, 在遍历的过程中需记录下遍历的步数, 直到访问到顶点 v 后, 遍历的步数即为要返回的结果 $D_s(u, v)$; 若不在同一划分区域, 则先由顶点 u 前向广度优先遍历 2ε 步, 此时有两种情况: (1) 在遍历过程中遇到顶点 v, 则直接返回两者的距离 $D_s(u, v)$. (2) 未遇到顶点 v, 则以 *key value* 的形式记录下访问到的中心顶点 u^* 和相应的步数 $D_s(u,u^*)$ 并构成集合 U^*, 然后由顶点 v 后向广度优先遍历 2ε 步, 记录下访问到的中心顶点 v^* 和相应的步数 $D_s(v^*,v)$ 并构成集合 V^*. 最终通过中心索引图查询集合 U^* 和集合 V^* 中可达中心顶点间的最短距离 $D_s(u^*,v^*)$, 得到查询结果 $D_s(u,v)=\min(D_s(u,u^*)+D_s(u^*,v^*)+D_s(v^*,v))$.

最短路径查询算法 *SPUI* (shortest path query on unified index) 的流程如算法 3 所示. 可以看出, 其和可达查询算法相似, 本文不再一一赘述, 最大的不同之处便是最短路径查询算法遍历的同时需记录下遍历的步数以供返回最终结果. 第 3–10 行讨论顶点 u 和顶点 v 在同一划分区域的情况, 其中在第 4 行记录遍历的步数, 第 6 行将记录的结果作为最终结果返回. 第 11–15 行讨论顶点 u 和顶点 v 不在同一划分区域的情况, 第 14 行更新最终结果.

算法 3. 最短路径查询算法 *SPUI*.

输入: 图 G, 顶点 u, 顶点 v, 中心索引图 G_{CI};
输出: 顶点 u 和顶点 v 的最短路径距离 $D_s(u, v)$.

1. $U^* \leftarrow \varnothing, V^* \leftarrow \varnothing, dist \leftarrow 0, (u,v) \leftarrow \infty$;
2. IF u, v in the same partition THEN
3. FOR each w that u visits in 2ε steps DO
4. $dist \leftarrow dist+1$;
5. IF $w == v$ THEN
6. $D_s(u, v) \leftarrow dist$;
7. END IF
8. END FOR
9. RETURN (u, v);

10. END IF
11. $U^* \leftarrow$ center vertices that u visits in 2ε forward BFS;
12. $V^* \leftarrow$ center vertices that v visits in 2ε backward BFS;
13. FOR each $u^* \in U^*$, each $v^* \in V^*$ DO
14. $D_s(u,v) = \min(D_s(u,u^*) + D_s(u^*,v^*) + D_s(v^*,v))$;
15. END FOR
16. RETURN (u, v).

例如, 给定 ε=2, 在图 3 中查询顶点 2 到顶点 14 的最短路径距离时, 顶点 2 与顶点 14 在同一划分区域, 则由顶点 2 开始前向广度优先遍历, 遍历 3 步后访问到顶点 14, 因此 D_s(2, 14)=3. 当查询顶点 17 和顶点 10 的最短路径距离时, 这两个顶点不在同一划分区域, 于是由顶点 17 前向广度优先遍历 4 步, 记录下访问到的中心顶点和相应的距离, 得到 $U^* = \{<0,1>, <8,4>\}$, 同理由顶点 10 后向广度优先遍历 4 步得到 $V^* = \{<8,2>\}$, 因此 D_s(17, 10) 即为 D_s(17, 0) D_s(0, 8) D_s(8, 10) 与 D_s(17, 8) D_s(8, 8) D_s(8, 10) 两者的最小值 6.

最短路径查询算法的时间复杂度与可达查询算法的时间复杂度相同, 即最差情况下为 $O(N_{2\varepsilon}(u)+E_{2\varepsilon}(u)+N_{E_{CI}})$.

4.3 关键字查询算法

当给定图 G 和一组关键字 $q_k=(w_1, w_2, \ldots, w_m)$ 后, 关键字查询算法基于统一索引中的中心索引和倒排索引得以加速实现. 其主要思想为: 首先基于倒排索引获取包含各关键字的顶点集合 $Node(k_i)$, 选取出顶点集合中包含元素个数最少的关键字 k_i 以及其集合中的顶点, 构成后向搜索列表 $BL(k_i)$, 其数据结构为 (*node*, *dist*), *dist* 初始化为 0; 然后逐一遍历后向搜索列表中的元素, 每遍历一个新顶点 u, 则最短路径算法查找到其他关键字 k_j 的最短距离 $dist_j$, 得到以顶点 u 为根顶点的查询结果 t=<u, (n_1, n_2, …, n_m) >, 并根据 $score(t) = \sum_{i=1}^{m} dist_i$ 评价函数来判断是否要更新当前最优结果; 接着将顶点 u 后向遍历一步访问到的新顶点添加到后向搜索列表中; 直到遍历结束返回查询结果.

为加快关键字查询算法的查询效率, 本文还将给出相应的剪枝策略. 根据后向搜索列表 $BL(k_i)$ 的头元素 (*node*, *dist*), 可得到目前未被访问的顶点中到关键字 k_i 距离最近的是顶点 *node*, 且距离为 *dist*. 因此, 若此时后向搜索列表头元素的 *dist* 值已经大于当前最优结果 t 的评价函数, 则说明后向搜索列表中不存在更优的结果, 可提前终止查询.

关键字查询算法 *KSUI* (keyword search on unified index) 的流程如算法 4 所示. 第 1 行初始化最终结果, 并将最优的评价函数置为无穷大, 以及申请了布尔类型的一维数组用于记录各顶点的访问情况. 第 2 行根据统一索引中的倒排索引获得各关键字的顶点集合, 并选取相应关键字初始化后向搜索列表 $BL(k_i)$. 第 4 行获取后向搜索列表的头元素, 每获取头元素需初始化两项内容. 第 5 行将顶点 *node* 到其他关键字 k_j 的距离 $dist_j$ 置为无穷大, 并将其中的 $dist_i$ 置为 *dist*, 除此之外还初始化了一棵新结果树 *newr*, 用于记录下到达关键字 k_j 的顶点 n_j. 第 6–8 行用于判断是否可以剪枝, 若可以直接返回结果树. 第 9–15 行基于中心索引获取当前顶点到其他关键字的最短距离, 其中第 12 行记录下最短距离和相应的顶点. 第 16–18 行用于判断当前顶点的评价函数是否优于当前最优结果, 若优于则更新结果和当前最优评价函数. 第 20–24 行更新后向搜索列表. 直到后向搜索列表为空, 第 26 行返回最优结果.

算法 4. 关键字查询算法 *KSUI*.

输入: 图 G, $q_k=(k_1, k_2, \ldots, k_m)$, 中心索引 G_{CI}, 倒排索引 II;
输出: 最优结果 t.

1. $t \leftarrow <\varnothing, (\varnothing, \varnothing, \ldots, \varnothing)>, t \leftarrow \infty, visited(v_i) \leftarrow$ false;
2. get $Node(k_i)$ and initialize $BL(k_i)$;
3. WHILE $BL(k_i)$ is not empty DO

4. <*node*, *dist*>← *BL*(k_i).*pop*;
5. $dist_j \leftarrow \infty(1 \leqslant j \leqslant m)$, $dist_i \leftarrow dist$, $newr \leftarrow <\varnothing,(\varnothing,\varnothing,\ldots,\varnothing)>$, $newr_i \leftarrow node$;
6. IF $dist \geqslant t$ THEN
7. RETURN *t*;
8. END IF
9. FOR $j \in [1,m]$ and $j \neq i$ DO
10. FOR each $u \in Node(k_j)$ DO
11. IF $dist_j > SPUI(node, u)$ THEN
12. $dist_j \leftarrow SPUI(node, u)$, $newr_j \leftarrow u$;
13. END IF
14. END FOR
15. END FOR
16. IF $\sum_{j=1}^{m} dist_j < t$ THEN
17. $r_{\text{root}} \leftarrow node$, $t \leftarrow \sum_{j=1}^{m} dist_j$, $r \leftarrow newr$
18. END IF
19. *visited*(*node*) ← true;
20. FOR each *v* that *node* visits in 1 backward DO
21. IF *visited*(*v*) == false THEN
22. *BL*(k_i).*push*(*v*, *dist*+1);
23. END IF
24. END FOR
25. END WHILE
26. RETURN *t*.

例如, 给定图 3 所示的图 *G* 和查询的关键字组 q_k=(*d*, *e*, *h*), 各关键字的顶点集合和后向搜索列表 *BL*(*e*) 的更新过程如图 7 所示. 查询时, 首先获取后向搜索列表 *BL*(*e*) 的头元素 (7, 0), 计算出顶点 7 到关键字 *d* 和 *h* 的距离, 得到评价函数 $\sum_{i=1}^{3} dist_i = D_s(7,7)+D_s(7,3)+D_s(7,7)=2$; 然后将顶点 7 后向遍历一步可以访问到的新顶点 2 添加到后向搜索列表 *BL*(*e*) 中, 即添加元素 (2, 1), 并将头元素 (7, 0) 从列表中删除; 接着获取头元素 (8, 0) 后, 得到顶点 8 评价函数 $\sum_{i=1}^{3} dist_i = D_s(8,11)+D_s(8,8)+D_s(8,3)=5$, 不优于当前的评价函数, 不需更新最优结果树; 如此反复遍历后向搜索列表 *BL*(*e*), 直到获取头元素 (0, 2) 后, 此时 *dist* 值已经大于等于当前最优评价函数, 因此可以终止本次查询, 并返回最优结果 *t*=<7, (7, 3, 7)>.

(a) 顶点集合 (b) 后向搜索列表

图 7 顶点集合与后向搜索列表

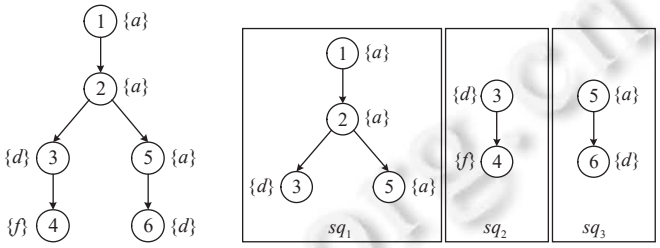
当给定图 *G* 和一组关键字 q_k=(k_1, k_2,…, k_m) 后, 关键字查询算法的时间主要用于遍历后向搜索列表 *BL*(k_i) 中的元素, 每访问到一个新顶点 *u* 时, 需要消耗 $m \cdot (O(N_{2\varepsilon}(u)+E_{2\varepsilon}(u))+O(N_{E_{\text{Cl}}}))$ 去查询到其他关键字的最短距离, 其中 *m* 为关键字的个数. 最差情况下, 需遍历图 *G* 中全部的顶点, 即时间复杂度为 $N_V \cdot m \cdot (O(N_{2\varepsilon}(u)+E_{2\varepsilon}(u))+O(N_{E_{\text{Cl}}}))$, 其中 N_V 为数据图 *G* 中的顶点总数, 但实际情况访问到的顶点个数要远小于 N_V.

4.4 图匹配查询算法

当给定图 G 和模式图 q_m 后, 图匹配算法首先将模式图划分为多个模式子图 sq_j; 然后在各划分区域 P_i 中匹配模式子图 sq_j, 并得到匹配子图 $sg_j(i)$; 接着在划分区域一定的情况下, 对所有匹配子图做连接操作, 得到该划分区域下对模式图的匹配结果 $g(i)$; 最终对所有 $g(i)$ 做并集操作得到查询结果.

4.4.1 模式图划分

划分模式图的主要思想是: 按照边覆盖原则将模式图划分为多个三层树结构的模式子图, 每个模式子图含有一个中间顶点, 第 1 层为中间顶点后向搜索一步访问到的顶点, 第 3 层为中间顶点前向搜索一步访问到的顶点, 第 1 层或第 3 层可不存在. 例如, 图 8(a) 所示的模式图的划分如图 8(b) 所示的 3 个模式子图.



(a) 模式图 (b) 模式子图

图 8 模式图划分

经研究发现, 模式子图的数量以及模式子图的匹配顺序会影响图匹配算法的效率, 模式子图越少, 连接操作越少, 算法效率越高; 后匹配的模式子图中已被匹配的标签越多, 算法的效率越高. 为得到最少的模式子图和最优的子图匹配顺序, 本文遵循以下策略划分模式图: 在模式图中首选与已选过的边相连且与选择性高的顶点有联系的边 $e=(u, v)$, 生成以选择性高的顶点为中间顶点的模式子图, 并移除模式图中与该顶点有关的边, 直到模式图中不存在可选的边时停止划分. 其中, 顶点的度越大, 且含有标签出现的频率越低, 则该顶点的选择性越高.

4.4.2 模式子图匹配

当给定划分区域 P_i 和模式子图 sq_j 后, 匹配模式子图的策略是: 首先基于倒排索引 II 中的 $L_{KN}(P, k)$ 列表去寻找可匹配中间顶点的顶点, 并构成候选顶点集合; 然后再逐一访问该集合中的顶点, 探索该顶点后向遍历一步访问的顶点是否可匹配第 1 层顶点, 该顶点前向遍历一步访问到的顶点是否可以匹配第 3 层顶点. 若都可找到匹配顶点, 则可获得划分区域 P_i 对模式子图 sq_j 的匹配结果 $sg_j(i)$.

但需注意一种特殊情况, 即候选顶点集合中可能存在边界顶点, 在匹配模式子图 sq_j 时会涉及跨划分区域匹配, 从而需要考虑当前划分区域的扩展问题. 因此, 在遍历候选顶点集合中的顶点时, 首先基于临界顶点集合来判断该顶点是否为边界顶点, 若该顶点在临界顶点集合中, 且其 *value* 值标记为 1 (2) 则为入 (出) 边界顶点, 需判断其入 (出) 顶点 v 是否可以匹配模式子图第 1 (3) 层顶点的某个顶点, 若可以匹配, 此时划分区域 P_i 则为 P_i 与 P_j 的并集, 使得当前进行匹配的划分区域得以扩展, 其中 P_j 则为顶点 v 所属的划分区域.

模式子图的匹配算法 *sub-PM* (pattern subgraph matching) 的流程如算法 5 所示. 第 1 行对最终结果初始化, 并将候选顶点集合 Q 置为空. 第 2 行获取中间顶点标签. 第 3 行根据中间顶点标签获取候选顶点. 第 4–15 行逐一访问候选顶点集合中的顶点, 每访问到一个新顶点 u, 第 5 行和第 6 行分别去匹配模式子图的第 1 层和第 3 层顶点. 当以顶点 u 匹配中间顶点并且第 1、3 层顶点都有相应匹配的顶点时, 第 8 行则将匹配的子图压入最终结果 $sg_j(i)$ 中; 若完成匹配并且顶点 u 为边界顶点时, 第 11 行则扩展当前匹配的划分区域; 直到候选顶点集合中的顶点都被访问过一次后, 便可返回最终结果 $sg_j(i)$.

算法 5. 模式子图的匹配算法 *sub-PM*.

输入: 划分区域 P_i, 模式子图 sq_j, 临界顶点集合, 倒排索引;
输出: 匹配子图 $sg_j(i)$.

1. $sg_j(i) \leftarrow \varnothing, Q \leftarrow \varnothing$;
2. *label* ← the key of the middle node in sq_j;
3. $Q \leftarrow$ get node from $L_{KN}(P_i, label)$;
4. FOR each u in Q DO
5. match nodes of the first level of sq_j;
6. match nodes of the third level of sq_j;
7. IF match all nodes of the sq_j THEN
8. $sg_j(i).push(all\ matches\ nodes)$;
9. IF $u \in B$ THEN
10. FOR each v in *all matches nodes* and $v \in (j \neq i)$ DO
11. $P_i \leftarrow P_i \cup P_j$;
12. END FOR
13. END IF
14. END IF
15. END FOR
16. RETURN $sg_j(i)$.

例如, 在图 4 所示的划分区域 P_3 中匹配图 8(b) 中的模式子图 sq_1 时, 由于该模式子图的中间顶点标签为 *a*, 则根据 $L_{KN}(P_3, a)$ 列表将候选顶点集合初始化为{17, 0}. 然后逐一去访问该集合中的顶点, 其中顶点 17 的入度为 0, 无法匹配第 1 层顶点. 当访问到顶点 0 时, 其双向广度优先遍历一步后, 分别找到了可匹配模式子图 sq_1 第 1 层和第 3 层的顶点, 即{17}和{1, 2}. 但由于顶点 0 属于边界顶点, 并且其中匹配的顶点 2 属于其他划分区域, 因此需将划分区域 P_3 更新为 $P_3 \cup P_1$, 接下来则在更新后的划分区域 P_3 中匹配其他模式子图.

4.4.3 模式图匹配

为加快模式图匹配算法的执行效率, 本文还给出了相应的剪枝策略, 即当划分区域 P_i 不可匹配其某一个模式子图 sq_j 时, 则说明在该划分区域中不存在可以匹配模式图的子图, 可直接返回 $sg_j(i)$ 为空. 除此之外, 按照最优匹配顺序逐一匹配模式子图的过程, 也可以看作是不断缩小最终结果的过程, 也可视为一种剪枝. 划分模式图后得到的最优匹配顺序, 可最大程度上约束后匹配的模式子图中的顶点, 使得在匹配后序模式子图时, 仅需在已匹配的顶点中选取含有中心顶点标签的顶点, 构成当前模式子图的中间顶点的候选顶点. 例如, 当在某划分区域匹配图 8(b) 中的模式子图 sq_2 时, 其中的顶点 3 已被约束在了匹配模式子图 sq_1 的结果子图中. 因此, 虽然在模式匹配的过程中会不断扩大划分区域, 但匹配的模式子图的约束顶点也越来越多, 符合的匹配结果会因约束越来越少, 而不会因为匹配区域的扩大而获得更多的匹配结果. 基于模式子图的匹配算法, 模式图匹配算法 *PMUI* (graph pattern matching on unified index) 的流程如算法 6 所示. 第 1 行初始化最终结果. 第 2–14 行为在各划分区域 P_i 中匹配模式图 q_m: 第 5 行调用 4 模式子图的匹配算法在各划分区域 P_i 中匹配模式子图 sq_j, 并得到匹配结果 $sg_j(i)$. 第 6–9 行用于判断是否满足剪枝条件. 第 11–13 行是对未剪枝情况的操作, 将同一划分区域中对模式子图的匹配结果做连接, 得到该划分区域下对模式图 q_m 的匹配结果. 第 16 行为对所有划分区域下对模式图 q_m 的匹配结果做并集操作. 第 17 行返回最终的匹配子图集合.

算法 6. 模式图匹配算法 *PMUI*.

输入: 划分覆盖 *P*, 模式图划分覆盖 *sp*, 临界顶点集合 *B*;
输出: 匹配的子图集合 *g*.

1. $g \leftarrow \varnothing$;

2. FOR $P_i \in P$ DO
3. $g(i) \leftarrow \varnothing$, $flag \leftarrow$ false;
4. FOR $sp_j \in sp$ DO
5. $sg_j(i) \leftarrow sub\text{-}PM(P_i, sp_j)$;
6. IF $sg_j(i)$ is empty THEN
7. $flag \leftarrow$ true;
8. BREAK;
9. END IF
10. END FOR
11. IF $flag ==$ false THEN
12. $g(i) \leftarrow^{\bowtie} sg_j(i)$;
13. END IF
14. END FOR
15. $g \leftarrow \bigcup g(i)$;
16. RETURN g.

例如, 在图 3 中匹配图 8(a) 所示的模式图 q_m 时, 由于在划分区域 P_1、P_2 和 P_4 中都无法匹配模式子图 sq_1, 即 $sg_1(1)=\varnothing$、$sg_1(2)=\varnothing$ 且 $sg_1(4)=\varnothing$, 则可直接返回 $g(1)=\varnothing$、$g(2)=\varnothing$ 和 $g(4)=\varnothing$. 第 4.4.2 节中我们给出划分区域 P_3 中匹配模式子图 sq_1 的过程, 在此不再赘述. 当在 P_3 中分别匹配模式子图 sq_2 和 sq_3 时, 都可找到相应的匹配顶点, 各匹配子图做连接操作后可得 $g(3)=\{17, 0, 1, 3, 2, 7\}$. 则最终结果 $g = \bigcup_{i=1}^{4} g(i) = g(3)$.

图匹配查询算法的时间复杂度主要由 3 部分构成, 其一是划分模式图 q_m 消耗一定的时间, 计算各顶点的选择性并进行排序需消耗时间 $O(N_{V_m} \cdot \log N_{V_m})$, 其中 N_{V_m} 是模式图中顶点的总数; 还需消耗时间 $O(E_{V_m})$ 逐一覆盖模式图 q_m 中的边, 其中 E_{V_m} 为模式图 q_m 中边的总数. 其二是在各划分区域 P_i 中匹配模式图 q_m 消耗一定时间, 匹配一个模式子图需消耗的时间为 $O(N_j \cdot (N_1(u) + E_1(u)))$, 若模式图 q_m 可划分为 n 个模式子图 sq_j, 则匹配模式图 q_m 的时间为匹配一个模式子图时间的 n 倍, 即 $n \cdot O(N_j \cdot (N_1(u) + E_1(u)))$, 最终连接 n 个模式子图 sq_j 的匹配子图所需的时间为 $O(N_j^2)$, 其中 N_j 为模式子图 sq_j 中顶点的个数, $N_1(u)$ 和 $E_1(u)$ 为从顶点 u 广度优先遍历一步访问到的顶点数和边数. 其三则是消耗常量时间对各划分区域的匹配结果做并集操作.

5 实验分析

本文选取了 4 组真实数据集, 为每组数据构建统一索引结构, 并执行本文提出的查询算法, 与现有非统一处理机制和高效的查询算法进行比较.

5.1 实验环境与数据集

实验环境: CPU 为 i5@3.30 GHz, 内存 8 GB, 硬盘 500 GB, 编程环境 GNU C.

实验数据集: 本文 4 组数据集的详细信息见表 1, 都是稀疏有向并带有标签的无环大规模数据图.

表 1 实验数据集

编号	数据集	顶点数 (k)	边数 (k)	标签数
1	WordNet	82	133	5
2	DBLP	409	591	6k
3	US Patent	3 774	16 522	416
4	Facebook	17 672	103 576	22 104k

数据集 1 表示了英语单词之间的关系 (http://snap.stanford.edu/data/). 数据集 2 是基于 DBLP XML 数据生成的带标签的有向图 (http://dblp.uni-trier.de/xml/), 首先在原始 XML 数据上为含有引用关系的论文添加有向边, 并删除大多数未引用或未被引用的论文. 数据集 3 中记录了美国专利的引用 (http://vlado.fmf.uni-lj.si/pub/networks/data/). 数据集 4 是基于 Facebook 用户数据生成的带标签有向图.

除此之外, 为保证数据集为 DAG 图, 通过深度优先遍历一次将图中的强连通分量拟合成一个顶点.

5.2 实验结果与分析

5.2.1 索引比较

本节将对比统一查询处理机制和非统一查询处理机制在解决 4 种查询问题时构建统一索引和非统一索引所消耗的时间和空间.

统一大图查询处理机制, 即构建统一索引, 便可进行可达、最短路径、关键字和图匹配这 4 种查询. 首先利用本文提出的图划分算法并给定 ε=2 对 4 组图数据集进行划分, 然后为每组数据集建立统一索引结构, 在 4 种数据集上构建统一索引所消耗的时间和占用的空间如图 9 所示. 由图 9(a) 可知, 构建统一索引的时间由划分图数据和生成统一索引结构两部分构成, 由图 9(b) 可知, 统一索引结构所占用的空间用于存储中心索引图、临界顶点集合和倒排索引这 3 部分内容, 为更好地与非统一查询处理机制进行对比, 本文将建立统一索引所消耗的时间和空间的总和总结如表 2 所示.

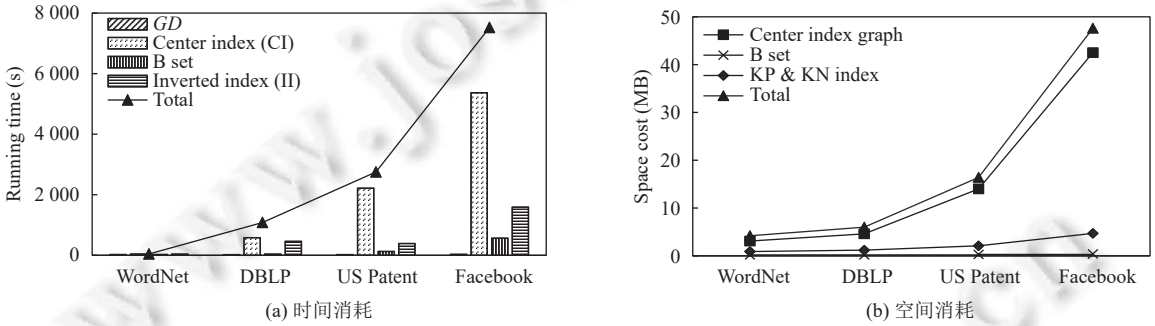


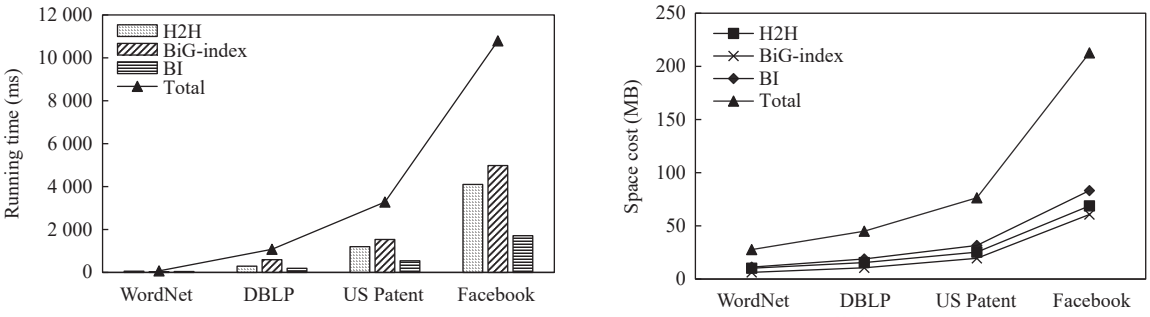
图 9 建立统一索引结构消耗的时间和空间

表 2 建立统一索引结构消耗的时间和空间

数据集	时间 (s)	空间 (MB)
WordNet	43.4	4.2
DBLP	1 081.6	6.0
US Patent	2 751.8	16.4
Facebook	7 521.3	47.6

现有的非统一查询处理机制, 即针对一种查询问题构建一种索引, 各索引的结构均不相同. 我们在 4 组数据集合中分别构建 Ouyang 等人[34]于 2023 年提出的 H2H 索引解决可达和最短路径查询、构建 Jiang 等人[23]于 2021 年提出的 BiG-index 索引解决关键字匹配查询, 以及构建 Sun 等人[33]于 2020 年提出的索引结构 BI 解决图匹配问题. 在 4 组数据集上, 构建上述 3 组索引消耗的时间与空间如图 10(a) 和 (b) 所示. 此外, 3 种索引在 4 组数据集上分别消耗的时间和空间如表 3 和表 4 所示.

由表 2–表 4 可以看出, 针对同一个图进行可达、最短路径、关键字和图匹配这 4 种查询时, 统一查询处理机制在构建统一索引过程中, 相较于现有非统一机制分别为多类查询单独构建索引, 最多可节省 39% 的构建时间, 并显著减少 77%–85% 的存储空间开销.



(a) 时间消耗 (b) 空间消耗

图 10 建立非统一索引结构消耗的时间和空间

表 3 建立非统一索引结构消耗的时间 (s)

数据集	H2H	BiG-index	BI	总计
WordNet	25.8	35.2	10.5	71.5
DBLP	292.1	603.6	202.6	1 098.3
US Patent	1 200.6	1 535.8	543.2	3 279.6
Facebook	4 100.8	4 980.7	1 710.2	10 791.7

表 4 建立非统一索引结构消耗的空间 (MB)

数据集	H2H	BiG-index	BI	总计
WordNet	10.2	6.3	11.2	27.7
DBLP	15.5	10.6	18.9	45.0
US Patent	25.3	19.5	31.6	76.4
Facebook	68.8	60.7	83.2	212.7

5.2.2 可达查询算法比较

本节针对可达查询问题, 将本文提出的基于统一索引的可达查询算法 *RQUI* 同现有的无索引广度优先遍历算法和基于 H2H 索引算法的执行效率做以比较. 首先随机选取 10 000 对查询顶点 u 和 v, 在 4 组数据集上执行上述 3 种可达查询算法, 查找顶点 u 是否可达顶点 v, 查询时间取 10 000 次查询的平均值. 然后为使比较结果更有意义, 每次随机从出度大于 0 的顶点中选取顶点 u, 从入度大于 0 的顶点中选取顶点 v, 确保每次查询无论顶点 u 和 v 是否可达, 必定会访问一些顶点并消耗一定的时间.

Bit-parallel BFS[35]、基于 H2H 索引和本文基于统一索引提出的 *RQUI* 这 3 种可达查询算法在不同数据集上所消耗的时间如图 11 所示, 为方便比较, 将统一索引结构和 H2H 索引在不同数据集上所消耗的存储空间也展示在了图 11 中. 可以看出, 基于 H2H 索引结构的算法和 *RQUI* 算法与 Bit-parallel BFS (图中以 B-P BFS 代替) 算法进行对比, 执行效率提高了 20%–28%. 还可以看出, 基于统一索引的 *RQUI* 算法与基于 H2H 索引的算法执行效率相似, 在大规模图中基于 H2H 索引的算法执行效率稍快一些, 其可基于索引直接给出答案, 而 *RQUI* 算法还需在中心索引上查询中心顶点的可达性, 但大规模图中 H2H 索引消耗的空间大于统一索引结构消耗的空间. 因此, 可以得出 *RQUI* 算法在保证尽量少的占用系统空间的前提下还能保证较为高效的查询效率.

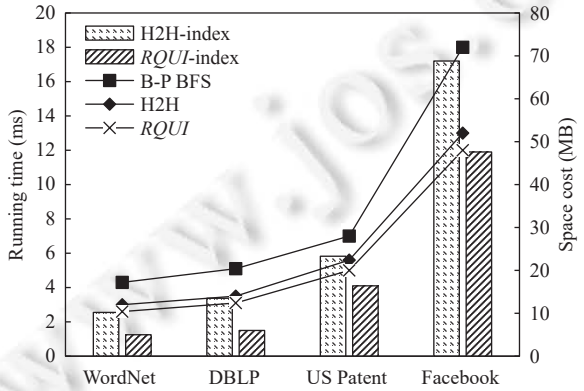


图 11 可达查询算法的效率比较

5.2.3 最短路径查询算法比较

本节针对最短路径查询问题, 将本文提出的基于统一索引的最短路径查询算法 *SPUI* 同现有的 Bit-parallel BFS 算法、IS-Label 算法[36]和基于 H2H 索引的最短路径查询算法的执行效率做以比较. 首先随机选取 10 000 对查询顶点 u 和 v, 在 4 组数据集上执行上述 4 种最短路径查询算法, 查找顶点 u 到顶点 v 的最短路径的距离 $D_s(u, v)$, 查询时间取 10 000 次查询时间的平均值. 然后为了使结果更有意义, 这里选取顶点的规则同可达查询算法比较时选取顶点的规则相同, 即保证顶点 u 的出度大于 0, 顶点 v 的入度大于 0, 确保每次查询无论顶点 u 和顶点 v 之间是否存在路径, 必定会访问一些顶点并消耗一定的时间.

Bit-parallel BFS、IS-Label 算法、基于 H2H 索引和本文基于统一索引提出的 *SPUI* 这 3 种最短路径查询算法在不同数据集上所消耗的时间如图 12 所示, 同样为了方便比较, 将统一索引结构和 H2H 索引在不同数据集上所消耗的空间也展示在了图 12 中. 可以看出, 基于 H2H 索引的查询算法和 *SPUI* 算法与 Bit-parallel BFS 算法进行对比, 执行效率提高 13%–33%; 与 IS-Label 算法进行对比, 在前 3 个数据集下执行效率提高 6%–11%, 但在规模较大的 Facebook 数据集下略差于 IS-Label 算法. H2H 算法和 *SPUI* 算法的流程相似, 最终 $D_s(u, v)$ 的结果都是由 3 部分构成, 即 $D_s = D_s(u,v) = D_s(u,u^*) + D_s(u^*,v^*) + D_s(v,v^*)$. 除此之外, 基于统一索引的 *SPUI* 算法与基于 H2H 索引的查询算法的执行效率相似, 且在大规模图中 H2H 算法的查询效率略快一些. 以上 IS-Label 算法和基于 H2H 索引的最短路径查询算法是现有大图查询处理机制针对最短路径距离查询提出的高效算法, 相似的结果证明了基于统一索引的 *SPUI* 算法仍可高效地解决最短路径查询问题, 并且 *SPUI* 算法中涉及的统一索引在大规模图中占用较少的存储空间.

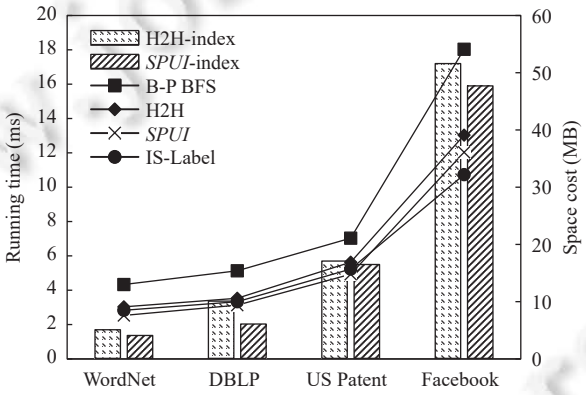


图 12 最短路径查询算法的效率比较

5.2.4 关键字查询算法比较

本节针对关键字查询问题, 首先验证了查询关键字组 q_k 中关键字的个数对本文提出的基于统一索引的关键字查询算法 *KSUI* 的影响, 随后, 将 *KSUI* 算法与现有的双向搜索算法, 以及结合最新索引技术 BiG-index 加速后的 BLINKS 算法 (命名为 BLINKS-BiG) 在执行效率上进行了对比分析.

本文在 4 组数据集的标签上随机选取 10 000 个查询关键字组, 每组含有 k 个关键字, 其中 $k \leqslant 10$. 执行 *KSUI* 算法得到关键字匹配的查询结果, 图 13 给出了 *KSUI* 算法的执行效率与查询关键字组中含有关键字个数的关系. 可以看出, 整体趋势上关键字组 q_k 中含有的关键字个数越多, 其查询响应的时间越长. 除此之外, 数据集中所含的标签数量越多, 标签分布越稀疏, 当查找的关键字较多时, 前向搜索需计算最短路径的顶点对反而越少, 从而可以提高整体的查询效率. 例如, DBLP 数据集中的标签分布较为稀疏, 当查询的关键字变多时, *KSUI* 算法的查询效率并没有像在 US Patent 和 Facebook 数据集上那样呈线性增长, 反而比较平缓.

本文在各组数据集的标签集合中随机选 10 000 组关键字, 为避免上述查询关键字组中关键字的个数对算法的影响, 每组关键字统一含有 4 个关键字, 执行上述 3 种关键字匹配算法, 查找与该组关键字匹配的结果, 查询时间取

10000 次查询的平均值. 为避免随机生成的关键字组并不存在查询结果而消耗太多时间, 规定各算法超过 90 s 则直接结束.

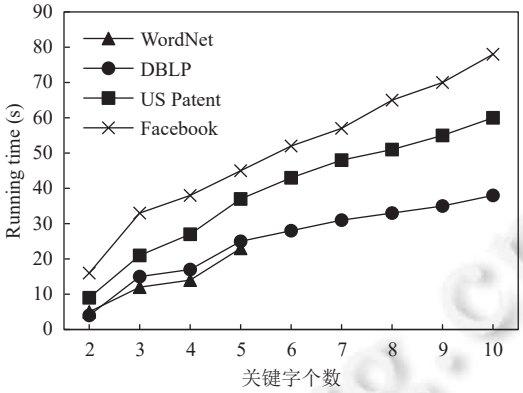


图 13 关键字个数对 *KSUI* 算法的影响

双向搜索、BLINKS-BiG 和本文提出的基于统一索引的 *KSUI* 这 3 种关键字查询算法在不同数据集上所消耗的时间如图 14 所示, 为方便比较, 将统一索引结构和 BLINKS-BiG 算法的索引结构 BiG-index 在不同数据集上所消耗的空间也展示在了图 14 中. 由图 14 可以看出, BLINKS-BiG 算法和 *KSUI* 算法, 与无索引的双向搜索算法对比, 执行效率至少提高了 30%, 在 DBLP 数据集上提高近 75%. BLINKS-BiG 算法和 *KSUI* 算法采用的都是双向搜索的思想, 并都是基于索引结构加速了双向搜索的过程. 其中 BLINKS-BiG 算法利用 BiG-index 预先构建的多层次摘要图结构, 在前向搜索时仅需在摘要图上定位包含查询关键字的摘要节点, 进而减少原图上的遍历开销, 而基于统一索引的 *KSUI* 算法前向搜索时需通过统一索引结构计算顶点与关键字的距离, 因此 *KSUI* 算法的执行效率低于 BLINKS-BiG 算法, 但整体上较为相似. 除此之外, 统一索引结构在空间上的优势较为明显, BLINKS-BiG 算法中涉及的 BiG-index 索引结构记录了过多的多层次摘要图信息, 从而增加系统的存储压力. BLINKS-BiG 算法是现有非统一查询处理机制专门针对关键字问题提出的, 图 14 中较为近似的查询效率和较为优越的空间占用率, 证明了基于统一索引的 *KSUI* 算法可高效地解决关键字查询问题.

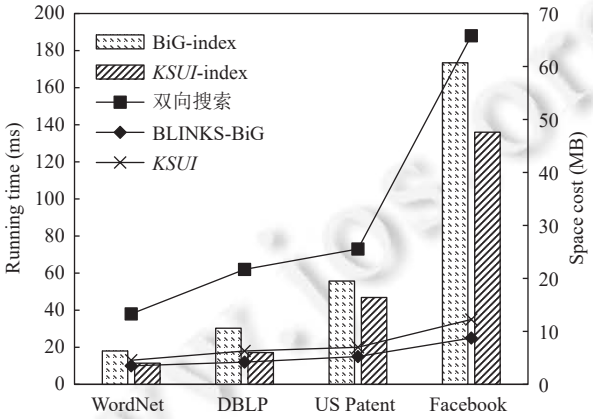


图 14 关键字查询算法的效率比较

5.2.5 图匹配查询算法比较

本节针对图匹配查询问题, 首先验证模式图的大小对本文基于统一索引提出的图匹配查询算法 *PMUI* 执行效率的影响, 然后将 *PMUI* 算法与现有的图匹配算法的执行效率做比较. Sun 等人[33]提出了 VC 算法用于加快精确

图匹配查询, 该算法通过代价模型生成优化匹配顺序, 并离线构建 BI 索引以记录候选顶点及其筛选邻居, 从而在查询阶段高效执行子图同构匹配.

本文在每组数据集上随机选择一个顶点, 从该顶点开始深度优先遍历, 利用遍历到的前 k 个顶点形成待匹配的模式图 q_k, 其中 k≤10, 如此反复, 生成 10 000 个规模不等的模式图 q_k. 在 4 组数据集上分别执行 *PMUI* 算法, 查询不同规模下模式图 q_k 的匹配子图, 图 15 给出了 *PMUI* 算法查询效率与模式图规模大小的关系, 其中横坐标为模式图 q_k 中含有的顶点个数. 可以看出, 整体趋势上模式图的规模越大图匹配的查询响应时间越长. 但在数据集 WordNet、US Patent 和 Facebook 中, 具有 10 个顶点的模式子图查询的执行时间却比具有 9 个顶点模式图的查询的执行时间低一些, 说明 *PMUI* 算法对中间顶点的约束导致其中间结果减少, 从而减少了连接操作消耗的时间, 降低了整体响应时间.

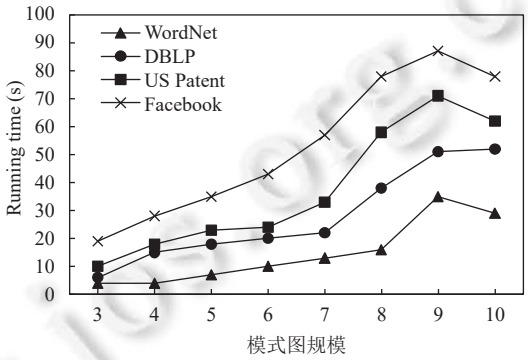


图 15 模式图规模对 *PMUI* 算法的影响

为避免模式图的规模对查询算法的影响, 采用上述实验生成的顶点个数为 5 和 10 的模式图, 在 4 组数据集上执行上述两种图匹配的查询算法, 查询两种规模的模式图的子图匹配, 消耗的时间为查询 10 000 次后的平均时间, 为避免某一模式图的匹配子图过多从而消耗大量时间, 则规定当查找到 1 024 个结果后即结束查询.

基于 BI 的 VC 算法与基于统一索引的 *PMUI* 算法在不同数据集上的运行时间如图 16 所示. 为便于对比, 图中还展示了两种索引结构在不同数据集上的空间开销. 由图 16 可见, *PMUI* 算法在执行效率上与现有的 VC 算法相近, 当模式图规模较大时, VC 在运行时间上略占优势. 然而, 统一索引结构显著降低了系统的存储压力, 使得 *PMUI* 在空间消耗上具有明显优势. 鉴于 VC 算法是专门针对图匹配查询设计的, 该实验结果表明, 基于统一索引结构的 *PMUI* 算法不仅能够高效支持图匹配查询, 还展现出良好的扩展性.

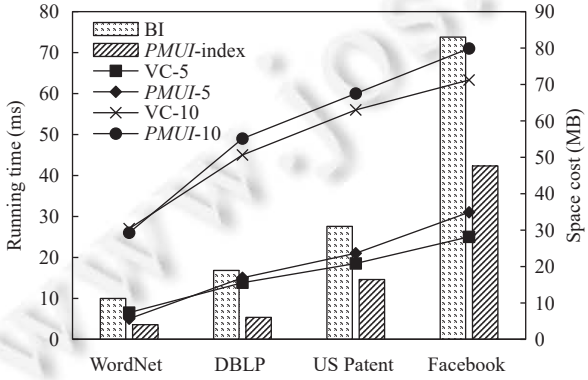


图 16 图匹配查询算法的效率比较

6 总 结

本文研究了统一的大图查询处理机制, 即为大图构建统一且高效的索引结构, 并基于统一索引结构设计了可达、最短路径、关键字和图匹配这 4 种查询算法, 该统一索引结构规模比图数据规模小, 并且能高效地支持上述 4 种查询. 最后, 通过在 4 组真实数据上的实验验证了统一索引结构和 4 种查询处理算法的高效性和扩展性. 在未来的工作中将会做如下的深入研究: (1) 考虑动态图中统一索引的更新问题; (2) 考虑其他查询问题的统一查询处理机制, 如 PageRank、SimRank、*k*-core、*k*-truss 和 Clique 等, 以建立最少的索引结构解决最多的查询问题为最终研究目标; (3) 考虑将统一查询处理机制融合到现有的图数据库中.

References

[1] Goyal P, Ferrara E. Graph embedding techniques, applications, and performance: A survey. Knowledge-based Systems, 2018, 151: 78–94. [doi: 10.1016/j.knosys.2018.03.022]

[2] Chen NY, Litvak N, Olvera-Cravioto M. Generalized PageRank on directed configuration networks. Random Structures & Algorithms, 2017, 51(2): 237–274. [doi: 10.1002/rsa.20700]

[3] Liu Y, Zheng BL, He XD, Wei ZW, Xiao XK, Zheng K, Lu JH. Probesim: Scalable single-source and top-*k* simrank computations on dynamic graphs. Proc. of the VLDB Endowment, 2017, 11(1): 14–26. [doi: 10.14778/3151113.3151115]

[4] Li Y Z, Zhu YY, Zhong M. *k*-core filtered influence maximization algorithms in social networks. Journal of Computer Applications, 2018, 38(2): 464–470 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2017071820]

[5] Huang X, Cheng H, Qin L, Tian WT, Yu JX. Querying *k*-truss community in large and dynamic graphs. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. Snowbird: ACM, 2014. 1311–1322. [doi: 10.1145/2588555.2610495]

[6] Zeume T. The dynamic descriptive complexity of *k*-clique. Information and Computation, 2017, 256: 9–22. [doi: 10.1016/j.ic.2017.04.005]

[7] Wu LK, Xiao XK, Deng DX, Cong G, Zhu AD, Zhou SG. Shortest path and distance queries on road networks: An experimental evaluation. Proc. of the VLDB Endowment, 2012, 5(5): 406–417. [doi: 10.14778/2140436.2140438]

[8] Zhang YF, Wang GR. SPTI: Efficient answering the shortest path query on large graphs. In: Proc. of the 2013 IEEE Int'l Congress on Big Data. Santa Clara: IEEE, 2013. 195–202. [doi: 10.1109/BigData.Congress.2013.34]

[9] Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword searching and browsing in databases using BANKS. In: Proc. of the 18th Int'l Conf. on Data Engineering. San Jose: IEEE, 2002. 431–440. [doi: 10.1109/ICDE.2002.994756]

[10] He H, Wang HX, Yang J, Yu PS. BLINKS: Ranked keyword searches on graphs. In: Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of data. Beijing: ACM, 2007. 305–316. [doi: 10.1145/1247480.1247516]

[11] Cheng JF, Yu JX, Ding BL, Yu PS, Wang HX. Fast graph pattern matching. In: Proc. of the 24th IEEE Int'l Conf. on Data Engineering. Cancun: IEEE, 2008. 913–922. [doi: 10.1109/ICDE.2008.4497500]

[12] Gao JR, Chen W, Xu JJ, Liu A, Li ZX, Yin HZ, Zhao L. An efficient framework for multiple subgraph pattern matching models. Journal of Computer Science and Technology, 2019, 34(6): 1185–1202. [doi: 10.1007/s11390-019-1969-x]

[13] Moayed H, Mansoori EG, Moosavi MR. An efficient pruning method for subgraph matching in large-scale graphs. The Journal of Supercomputing, 2023, 79(10): 10511–10532. [doi: 10.1007/s11227-023-05061-1]

[14] Claude F, Navarro G. Fast and compact Web graph representations. ACM Trans. on the Web (TWEB), 2010, 4(4): 16. [doi: 10.1145/1841909.1841913]

[15] Brisaboa NR, Ladra S, Navarro G. k^2-trees for compact Web graph representation. In: Proc. of the 16th Int'l Symp. on String Processing and Information Retrieval. Saariselkä: Springer, 2009. 18–30. [doi: 10.1007/978-3-642-03784-9_3]

[16] Rossi RA, Zhou R. GraphZIP: A clique-based sparse graph compression method. Journal of Big Data, 2018, 5(1): 10. [doi: 10.1186/s40537-018-0121-z]

[17] Yu CY, Ren TM, Li WY, Liu HM, Ma HT, Zhao YH. BL: An efficient index for reachability queries on large graphs. IEEE Trans. on Big Data, 2024, 10(2): 108–121. [doi: 10.1109/TBDATA.2023.3327215]

[18] Agrawal R, Jagadish HV. Algorithms for searching massive graphs. IEEE Trans. on Knowledge and Data Engineering, 1994, 6(2): 225–238. [doi: 10.1109/69.277767]

[19] Yang YJ, Li ZF, Wang X, Hu QH. Finding the shortest path with vertex constraint over large graphs. Complexity, 2019, 2019: 8728245. [doi: 10.1155/2019/8728245]

[20] Wang Y, Wang Q, Koehler H, Lin Y. Query-by-Sketch: Scaling shortest path graph queries on very large networks. In: Proc. of the 2021

Int'l Conf. on Management of Data. ACM, 2021. 1946–1958. [doi: 10.1145/3448016.3452826]
[21] Wei F. TEDI: Efficient shortest path query answering on graphs. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. Indianapolis: ACM, 2010. 99–110. [doi: 10.1145/1807167.1807181]
[22] Sun Z, Wang HZ, Wang HX, Shao B, Li JZ. Efficient subgraph matching on billion node graphs. Proc. of the VLDB Endowment, 2012, 5(9): 788–799. [doi: 10.14778/2311906.2311907]
[23] Jiang JX, Choi B, Xu JL, Bhowmick SS. A generic ontology framework for indexing keyword search on massive graphs. IEEE Trans. on Knowledge and Data Engineering, 2021, 33(6): 2322–2336. [doi: 10.1109/tkde.2019.2956535]
[24] Ghanbarpour A, Niknafs K, Naderi H. Efficient keyword search over graph-structured data based on minimal covered *r*-cliques. Frontiers of Information Technology & Electronic Engineering, 2020, 21(3): 448–464. [doi: 10.1631/FITEE.1800133]
[25] Singh K, Singh V. Graph pattern matching: A brief survey of challenges and research directions. In: Proc. of the 3rd Int'l Conf. on Computing for Sustainable Global Development. New Delhi: IEEE, 2016. 199–204.
[26] Bouhenni S, Yahiaoui S, Nouali-Taboudjemat N, Kheddouci H. Distributed graph pattern matching via bounded dual simulation. Information Sciences, 2022, 610: 549–570. [doi: 10.1016/j.ins.2022.08.038]
[27] Ullmann JR. An algorithm for subgraph isomorphism. Journal of the ACM, 1976, 23(1): 31–42. [doi: 10.1145/321921.321925]
[28] Cordella LP, Foggia P, Sansone C, Vento M. A (sub) graph isomorphism algorithm for matching large graphs. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2004, 26(10): 1367–1372. [doi: 10.1109/TPAMI.2004.75]
[29] Shasha D, Wang JTL, Giugno R. Algorithmics and applications of tree and graph searching. In: Proc. of the 21st ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. Madison: ACM, 2002. 39–52. [doi: 10.1145/543613.543620]
[30] Zhao PX, Han JW. On graph query optimization in large networks. Proc. of the VLDB Endowment, 2010, 3(1–2): 340–351. [doi: 10.14778/1920841.1920887]
[31] Han WS, Lee J, Lee JH. Turbo$_{iso}$: Towards ultrafast and robust subgraph isomorphism search in large graph databases. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 337–348. [doi: 10.1145/2463676.2465300]
[32] Carletti V, Foggia P, Saggese A, Vento M. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2018, 40(4): 804–818. [doi: 10.1109/TPAMI.2017.2696940]
[33] Sun SX, Luo Q. Subgraph matching with effective matching order and indexing. IEEE Trans. on Knowledge and Data Engineering, 2022, 34(1): 491–505. [doi: 10.1109/tkde.2020.2980257]
[34] Ouyang D, Wen D, Qin L, Chang LJ, Lin XM, Zhang Y. When hierarchy meets 2-hop-labeling: Efficient shortest distance and path queries on road networks. The VLDB Journal, 2023, 32(6): 1263–1287. [doi: 10.1007/s00778-023-00789-x]
[35] Akiba T, Iwata Y, Yoshida Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 349–360. [doi: 10.1145/2463676.2465315]
[36] Fu AWC, Wu HH, Cheng J, Wong RCW. IS-Label: An independent-set based labeling scheme for point-to-point distance querying. Proc. of the VLDB Endowment, 2013, 6(6): 457–468. [doi: 10.14778/2536336.2536346]

附中文参考文献

[4] 李阅志, 祝园园, 钟鸣. 基于 *k*-核过滤的社交网络影响最大化算法. 计算机应用, 2018, 38(2): 464–470. [doi: 10.11772/j.issn.1001-9081.2017071820]

作者简介

陈迪, 博士, 讲师, 主要研究领域为图数据分析与管理, 时空众包在线算法设计.
袁野, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为大图数据管理与分析, 基于大数据的人工智能, 区块链.
潘雅妮, 硕士, 主要研究领域为图数据挖掘, 图计算.
王国仁, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为不确定数据管理, 数据密集型计算, 可视媒体数据管理与分析, 非结构化数据管理, 分布式查询处理与优化技术.