一种适用于大图的*k*步可达性查询算法

同正南, 卜天明

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230500031

一种适用于大图的 *k* 步可达性查询算法

同正南　　卜天明
华东师范大学软件工程学院　上海 200062
(orang_913@qq.com)

摘　要　*k* 步可达查询用于在给定的有向无环图(Directed Acyclic Graph,DAG)中回答两点之间是否存在长度不超过 *k* 的路径。针对现有方法的索引规模大、查询处理效率低的问题,提出了一种构建在大图上的基于树覆盖的倍增索引来提高索引查询效率,并结合 GRAIL 算法和改进的 FELINE 算法对本身就不可达查询点对进行剪枝。基于 19 个真实的数据集进行了实验测试,并将所提算法与现有算法在构建索引大小、索引时间、查询时间 3 个指标上进行了实验对比。实验结果验证了所提算法的高效性。

关键词:*k* 步可达性查询;倍增索引;索引标签;树覆盖;在线搜索
中图分类号　TP301.6

K-step Reachability Query Algorithm for Large Graphs

TONG Zhengnan and BU Tianming
School of software Engineering,East China Normal University,Shanghai 200062,China

Abstract　The *k*-step reachable query is used to answer whether there exists a path of length not exceeding *k* between two points in agiven directed acyclic graph(DAG). To address the problems of large index size and low query processing efficiency of existing methods,this paper proposes a multiplicative index built on a large graph based on tree cover to improve index query efficiency,and combines GRAIL algorithm and the improved FELINE algorithm for pruning the point pairs of inherently unreachable queries. The paper conducts experimental tests based on 19 real datasets and compares with existing algorithms in three metrics:index size,index time,and query time. The experimental results verify the efficiency of the proposed algorithm in this paper.

Keywords　*K*-step reachability query,Multiplicative index,Index label,Tree cover,Online query

1　引言

如今图模型已经被广泛应用于各个领域,如分布式计算、社交网络、数据挖掘、生物网络分析、金融系统、传感器网络、云计算、交通网络[1-3]等。这些应用网络中的实体数据可以以有向无环图的形式组织,而实体间的关系可以表示成图模型中的边,这样就能构成基于图模型存储的数据结构。

k 步可达性查询可以应用在无线传感器网络中,广播消息在每一跳传输中信号强度都会有所削弱,所以随着传输路径的增长,信号强度会呈指数级下降[5],而 *k* 步可达性可以模拟影响信号强度的级别和范围,在许多分析任务中是有用的。在社交网络中,有一个理论说每一个人最多可以通过 6 个人而认识任何一个陌生人。但随着相隔人数的增加,关系之间熟悉的程度也会呈指数级降低,因此可以利用 *k* 步可达性来模拟关系的熟悉程度。该问题还可应用于文献之间的引用关系,如果文献之间引用关系所连接的文献数量较多,则说明两篇文献相关联程度越低,因此文献之间的相关联程度的高低可以采用 *k* 步可达性查询来判断。

已知用来处理 *k* 步可达性查询的算法有 PLL[6] 算法和 *k*-reach[7] 算法。其中 PLL 算法采用了 2-hop[8] 框架计算出每个点的 L_{in} 和 L_{out} 集合,判定 *k* 步可达性的方式是若 u 到 $L_{out}(u)$ 和 $L_{in}(v)$ 的交集点 m 的最短距离加上 m 到 v 的最短距离不大于 *k*,则判定可达。整个判定的过程采用类似归并排序的思路实现,时间复杂度为 $O(|L_{out}(u)|+|L_{in}(v)|)$,但是构造 2-hop 索引过程的时间复杂度很大,因为需要针对每个点对整个图进行 2 次遍历,其时间复杂度为 $O(|V|(|V|+|E|))$。因此,PLL 算法只适用于小规模的图,对于大图其构造 2-hop 索引的时间是无法估量的。*k*-reach 算法判定 *k* 步可达性是通过预处理得到的索引子图的 *k* 步可达性的传递闭包来快速判定。这个传递闭包采用有序邻接表实现,实际查询是通过二分搜索找出索引子图中的最短距离。因此其查询时间复杂度为 $O(deg^+(u)\cdot deg^-(v)\cdot \lg|V|)$。但是 *k*-reach 算法构造的索引子图只是针对某个事先给定的 *k* 值,如果两次查询的 *k* 值不同,就需要重新构造索引子图。而构造索引子图需要对预处理得到的极小点覆盖中的所有点求解其 *k* 步以内的传递闭包。极小点覆盖的大小与原始图的稠密性有关,图越稠密,极小点覆盖就越小,对应的索引子图的规模也会越小。构造索引子图的时间复杂度为 $O(|S_v|(|V|+|E|))$,其中 $|S_v|$ 为极小节点覆盖的点的个数。因此,*k*-reach 算法更多地适用于稠密图。对于大的稀疏图,其极小点覆盖最坏情况下接近于原始图的大小,其构造索引子图的传递闭包的时间界同样无法估量。

本文主要针对有向无环图进行研究。针对现有 *k* 步可达性查询算法在大图上构建索引和查询等性能较低等问题,提

通信作者:卜天明(tmbu@sei.ecnu.edu.cn)

出了一种新的基于树覆盖构建倍增索引的 k 步可达性查询算法。本文的主要贡献如下：

(1)在有向无环图上构建最短路径森林，并在最短路径树上构建自适应 k 值的倍增索引算法。论文基于倍增索引设计了一种适用于大规模图的 k 步可达性查询算法。

(2)为倍增索引查询算法补充了在线搜索的不可达查询的剪枝，结合 GRAIL 以及 FELINE 索引，并改进了 FELINE 索引的构造时间。

(3)在 19 个真实的数据集上和已有算法进行了比较，验证了基于倍增索引的 k 步可达性查询算法在索引构造时间、索引大小、查询时间上的高效性。

2 相关概念及工作

2.1 图的相关概念及符号

图的相关概念及符号如表 1 所列。

表 1 图的相关概念及符号

Table 1 Concepts and symbols related to graphs

符号	含义
DAG	有向无环图 G
V	顶点的集合
E	边的集合
$dis(u,v)$	顶点 u 到顶点 v 的最短距离
$deg^{+}(u)$	顶点 u 的出度
$deg^{-}(u)$	顶点 u 的入度
$N^{+}(u)$	从顶点 u 发出的边指向的顶点集合
$N^{-}(u)$	指向顶点 u 的顶点集合
spt	最短路径树是在原 DAG 上进行广度优先遍历得到的一棵有向树
$u \rightarrow_k v$	顶点 u 在 k 步内可以到达顶点 v
$u \nrightarrow_k v$	顶点 u 在 k 步内不可到达顶点 v

2.2 相关工作

对于传统可达性查询问题，早期存在 2 种简单的处理方式。第一种方式是暴力进行深度或广度优先遍历[4]，其构建索引的空间复杂度为 O(1)，查询的时间复杂度为 $O(|V|+|E|)$。第二种方式为对整个原始图构建传递闭包[9]，其构建索引的空间复杂度为 $O(|V|^2)$，查询的时间复杂度为 O(1)。

近年来，一些研究学者主要考虑的方法是尽可能多地覆盖点对的可达性关系，同时压缩原有的传递闭包的空间开销。现在可将已存在的可达性查询算法分为 2 类。第一种是基于标签类索引的可达性查询算法。基于标签类索引的可达性查询算法根据建立索引的子图不同，还可进一步分为 3 类：第一类是基于 2-hop 标记的可达性查询算法，代表性算法有 2-hop[8]，DL[10]，TF[11]，TOL[12] 等；第二类是基于树覆盖的可达性查询算法，代表性算法有 Optimal Tree Cover[13]，Dual-Labeling[14]，GRIPP[15]，Label-SSPI[16]；第三类是基于链覆盖的可达性查询算法，代表性算法有 3-hop[17]，Path Tree[18]，Path Hop[19]。而第二种是在线搜索类的可达性查询算法，这类算法的代表有 GRAIL[20]，FELINE[21]，FERRARI[22] 和 IP+[23]。

由于传统可达性查询算法只能局限于判断两点是否可达，无法判断两点之间的最短距离是否满足某个给定长度，所以后续又有相关研究学者提出 k 步可达性查询算法。已有的 k 步可达性查询算法有 Cheng 等提出的基于极小顶点覆盖集的 k-reach[7] 算法，Yano 等提出的基于 2-hop 标记的 PLL[6] 算法，Jin 等提出的 HCL[24] 算法，Wong 等提出的 HLS[25] 算法。

2.2.1 基于标签的可达性查询算法

Cohen 和 Halperin 等提出的 2-hop[8] 覆盖算法的核心思想是为每个节点分配 2 个集合 $L_{in}(u)$ 和 $L_{out}(u)$。$L_{in}(u)$ 表示通过 DAG 连边到达 u 节点的顶点集合，$L_{out}(u)$ 表示从 u 节点出发通过 DAG 连边到达的顶点的集合。2-hop 覆盖算法判定可达性的方法是如果 $L_{out}(u) \cap L_{in}(v) \neq \emptyset$，则节点 u 可达节点 v；若 $L_{out}(u) \cap L_{in}(v) = \emptyset$，则节点 u 不可达节点 v。2-hop 覆盖算法最关键的地方在于如何找到一个尽可能小的 2-hop 标签集合覆盖到所有的点对之间的可达性关系。

Jin 和 Wang 等提出的 DL[10] 算法同样是基于 2-hop 标记覆盖的思想，只是在构建 2-hop 标签时通过剪枝条件尽可能缩小 2-hop 集合的规模，同时保证能够覆盖所有点对的可达性关系。2-hop 标签的构造过程为：先通过 u 节点出发进行正向的广度优先搜索，如果 $L_{out}(u) \cap L_{in}(v) = \emptyset$，则在所到达的顶点 v 的 $L_{in}(v)$ 标签集合中加入 u 节点，继续沿着 v 节点正向搜索，否则不对 $L_{in}(v)$ 作任何改变，搜索结束；再通过 u 节点进行反向边的广度优先搜索，如果 $L_{out}(v) \cap L_{in}(u) = \emptyset$，则在所到达的顶点 v 的 $L_{out}(v)$ 标签集合加入 u 节点，继续沿着 v 节点反向搜索，否则不对 $L_{out}(v)$ 作任何改变，搜索结束。

Cheng 和 Huang 等提出的 TF[11] 算法也是基于 2-hop 标记覆盖的，只是 TF 算法是先将原 DAG 的规模不断缩小，针对多个子图构建 2-hop 标记，来缩小构建 2-hop 标记覆盖集合的规模，从而缩减构建索引的大小、时间，以此来提高查询效率。具体的做法是 TF 算法会先通过拓扑排序生成每个顶点所在的拓扑层，再通过构建虚拟节点去除跨层边生成 k-partite DAG，即一条边的跨越层次不会超过 2，之后再将奇数拓扑层去除，不断重复这个过程，直到只剩下一个拓扑层为止。这个过程中会得到多个拓扑折叠子图，之后对这些拓扑折叠子图构建 2-hop 标记集合，在构造过程中 TF 算法的优化策略是移除了虚拟节点和按照顶点的度数降序构建 2-hop 标签。

2.2.2 基于树覆盖的可达性查询算法

Agrawal 和 Borgida 等提出的 Optimal Tree Cover[13] 算法是一个最优树覆盖算法。Optimal Tree Cover 算法的构造过程是先对 DAG 进行深度优先搜索，对于树边上的每个节点 u 会分配一个区间标签 $I_u = [min\_post, post]$，区间右侧 $post$ 标记为深度遍历时的后序编号，区间左侧标签 min_post 为深度遍历时当前节点子树中最小的后序编号。而对于非树边(如图 1 中的 $d \rightarrow e$)，为了完善原先树边不能覆盖到的可达性关系，会将节点 e 的树边区间[1,3]加入节点 d，对于其余非树边 $d \rightarrow h$，$d \rightarrow b$，会将区间[2,2][1,4]均加入节点 d 的区间列表中，这样节点 d 的区间列表集合为[6,7][1,3][2,2][1,4]，这里因为区间[2,2][1,3] 均被区间[1,4]覆盖，为了减小索引标签的规模，将其删除，所以节点 d 的最终区间列表为[6,7][1,4]。因为节点 a 为节点 d 的父亲节点，因此节点 a 要继承节点 d 的非树边区间[1,4]，但因为区间 $[1,4] \subseteq [1,8]$，所以将区间[1,4]删除。因此，对于顶点对 $u \rightarrow_? v$ 可达性的判定，只要节点 u 存在一个区间列表 I_u 包含节点 v 的树边区间 I_v，即 $I_v \subseteq I_u$，则节点 u 一定可达节点 v，否则节点 u 不可达节点 v。

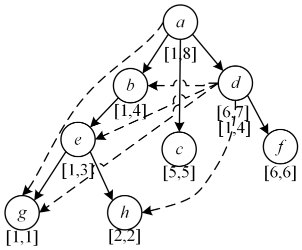

图 1　Optimal Tree Cover 算法实例

Fig. 1　Example of Optimal Tree Cover

Wang 和 He 等提出的 Dual-Labeling[14] 算法的核心思想是先通过深度搜索提取出原 DAG 的生成树，并为树边上的每个节点 u 分配一个区间标签 $I_u=[pre, min\_post)$，其中区间左侧 pre 表示深度搜索时的前序遍历编号，区间右侧 min_post 表示后续遍历子节点编号中的最小值，而节点的后续遍历编号是从叶子节点开始计数，当前节点的前序遍历序号加 1 便是后续遍历编号；并且对于非树边 (u,v)，$I_u=[a,b)$，$I_v=[c,d)$，如果 $c\in[a,b]$，则说明节点 u 已经可以通过树边到达节点 v 了，不做任何处理，如果 $c\notin[a,b]$，则说明节点 u 不能通过树边到达节点 v，但可以通过非树边 (u,v) 到达节点 v，这种情况会生成一个传递链接 $a\rightarrow[c,d)$ 记录进传递链接表中。

Dual-Labeling 算法判定节点 u,v 间可达性的定理为：节点 u 可以到达节点 v 当且仅当 $c\in[a,b)$ 或者存在传递链接实体 $i\rightarrow[j,k)$，$i\in[a,b)$ 且 $c\in[j,k)$，对于第二种情况可以根据传递性，因为 $i\in[a,b)$，则节点 u 一定可达 i 所在的节点 x，而根据传递链接 $i\rightarrow[j,k)$，则节点 x 可以通过非树边到达 $[j,k)$ 所在的节点 y，而 $c\in[j,k)$，说明节点 y 可以通过树边到达节点 c，所以根据传递性推出节点 u 可达节点 v。传递链接表中的传递链接可以根据传递性质再生成新的链接，例如 $i_1\rightarrow[j_1,k_1)$，$i_2\rightarrow[j_2,k_2)$，若存在 $i_2\in[j_1,k_1)$，则生成 $i_1\rightarrow[j_2,k_2)$。实际论文证明出传递链接表中的链接条数不会超过 $t(t+1)/2$，其中 t 是由非树边生成的初始传递链接条目的数量。

2.2.3　基于在线搜索类的可达性查询算法

Yildirim 和 Zaki 等提出的 GRAIL[20] 算法的核心思想是通过对 DAG 进行深度优先搜索为每个节点分配一个区间标签 $I_u=[min\_post, post]$，如果一个节点已经遍历分配过标签，则不再遍历分配标签，其中右标记 $post$ 是指对 DAG 采取深度优先遍历时当前节点 u 的后序值，而左标记 min_post 是当前节点 u 的所有子节点 $post$ 的最小的后序值。GRAIL 算法对于不可达判定的方法是对于可达性查询 $u\rightarrow_? v$，若终点 v 的区间标签不包含在起点 u 的区间标签范围内即 $I_v\subseteq I_u$，即可推出节点 u 不可达节点 v。但单个标签区间所能覆盖到的不可达查询点有限，而 GRAIL 算法可以通过修改深度优先搜索对于 u 节点的直接儿子节点的遍历顺序来生成多个不同的 $I_u=[min\_post, post]$ 区间标记，这样生成的 k 个区间标签 $L_u=\{I_{u_1}, I_{u_2}, \cdots, I_{u_k}\}$ 就可以覆盖更多的不可达查询点对，以此减少在线搜索时对不可达查询点对搜索的深度。GRAIL 算法在在线搜索时会先判断当前查询点对 (u,v) 是否符合不可达判定的条件，如果符合则直接返回不可达，否则继续进行深度优先搜索判断节点 u 的子节点和终点 v 的可达性。

Veloso 和 Cerf 等提出的 FELINE[21] 算法的核心思想是根据拓扑排序的拓扑序大小关系来快速判定 DAG 上两点之间的不可达性。算法思路是对 DAG 进行拓扑排序，每个节点 u 会产生一个拓扑序 $X(u)$，可以得出拓扑序较大的点一定是不可达拓扑序较小的点的，即若 $X(u)>X(v)$，可得出节点 u 不可达节点 v，亦即 $u\nrightarrow v$。由于只计算一个 X 拓扑序数组只能覆盖一部分不可达点对，因此 FELINE 索引的算法思想是分两次拓扑排序，并且要使得两次拓扑序尽可能序号不同，方法是第一次拓扑排序求出拓扑序 X 数组，第二次拓扑排序使用优先队列对于入度为零的点出队顺序按照第一次拓扑序 X 拓扑序较大的节点先出队，即第二次拓扑排序得到的拓扑序 $Y(u)$ 比第一次得到的拓扑序 $X(u)$ 小，这样才能使得 Y 拓扑序能尽可能多地覆盖到 X 拓扑序覆盖不到的不可达点对。判断法则为 DAG 上连点 u 和 v 满足 $X(u)>X(v)$ 或 $Y(u)>Y(v)$，则可推出 $u\nrightarrow v$。FELINE 算法在在线搜索时会先判断查询点对是否符合不可达判定关系，如果符合，则直接返回不可达，否则继续进行深度优先搜索判断节点 u 的子节点和终点 v 的可达性。

2.2.4　k 步可达性查询算法

Akiba 和 Iwata 等提出的 PLL[6] 算法是在 2-hop[8] 覆盖算法基础上将每个节点 u 的 L_{in} 和 L_{out} 集合存储的元素扩展为二元组，$L_{in}(u)$ 集合中的每个二元组为到达 u 的节点和到达 u 节点的最短距离，$L_{out}(u)$ 集合中的每个二元组为 u 节点能到达的节点和到达该节点的最短距离。PLL 算法判定 u 节点是否 k 步可达 v 节点的方法为遍历 $L_{out}(u)$ 和 $L_{in}(v)$ 集合找到集合的交集，并记录交集中所有点对应的距离之和的最小值 d，如果 $d\leqslant k$，则说明 $u\rightarrow_k v$，否则 $u\nrightarrow_k v$。

Cheng 和 Shang 等提出的 k-reach[7] 算法是基于极小顶点覆盖集 S 构建的索引子图二分查找算法。其核心思想是首先会依次选定当前 DAG 中剩余的度数最大的点加入集合 S 中，并断开该点的所有连边，依次重复这个过程，直到所有连边都被断开，所得到的 S 就是极小顶点覆盖集。然后对 S 中的所有点进行一遍长度限定为 k 的广度优先遍历，超过 k 搜索就会终止。算法对每个起始点构造一个邻接表，邻接表中存放的是当前遍历到点的编号，以及当前起始点遍历到其余所有点的最短距离的二元组，并且当起始点的广度搜索遍历完成后会对当前邻接表中的二元组按照存放点的编号进行升序排序，以便于后续查询时进行二分搜索。至此，限定距离为 k 的极小顶点覆盖集 S 上的索引子图就构建成功。

k-reach 算法的查询算法思想是对于给定查询 $u\rightarrow_{k?} v$，如果节点 u,v 都在极小顶点覆盖集 S 中，则直接对 u 节点所在的邻接表进行二分搜索是否存在 v 节点，若存在则说明 $u\rightarrow_k v$，否则 $u\nrightarrow_k v$。如果节点 u 不在 S 中，节点 v 在 S 中，则枚举节点 u 的出边集合中的点 x，二分查询 x 的邻接表是否存在点 v 并且距离不超过 $k-1$。若不超过则说明 $u\rightarrow_k v$，否则 $u\nrightarrow_k v$。如果节点 u 在 S 中，节点 v 不在 S 中，则枚举节点 v 的入边集合中的点 x，二分查询邻接表 u 中是否存在点 x 并且距离不超过 $k-1$，若不超过则说明 $u\rightarrow_k v$，否则 $u\nrightarrow_k v$。如果节点 u 不在 S 中，并且节点 v 也不在 S 中，则枚举节点 u 的出边集合中的点 x，再对应枚举节点 v 的入边集合中的点 y，二分查询邻接表 x 中是否存在点 y 并且距离不超过 $k-2$，若不超过则说明 $u\rightarrow_k v$，否则 $u\nrightarrow_k v$。

本文第 3 章将具体介绍论文所提算法中索引的构建部分，包括了在最短路径树上的倍增索引的构建，对 FELINE

索引的改进，以及构建索引所需要的时间和空间复杂性分析；第 4 章在所构建的索引的基础上，完整地叙述了 k 步可达性查询算法，证明了该可达性算法的正确性，分析了查询时间复杂度；第 5 章对上述算法做了完整的实验，并和其他算法做了全面的对比；最后总结全文。

3 索引的构建

3.1 问题分析

关于节点 u 到节点 v 是否 k 步可达，如果只是暴力搜索出所有可达路径，这样并不能保证先行搜索到距离小于等于 k 的路径，如果先行搜索到的路径长度一直大于 k，就需要一直搜索下去，在最坏情形下需要将节点 u 到达节点 v 的所有可能路径搜索完全才能得出 k 步可达性的判定结果。因此，可以考虑能否预先计算出节点 u 到达节点 v 的最短距离，直接判断节点 u 到节点 v 的最短距离是否大于 k，如果节点 u 到达节点 v 的最短距离都大于 k，那么就可以直接得出节点 u k 步不可达节点 v，而不必再去搜索 u 到 v 的剩余可达路径了。

已有算法的局限性在于构建索引的对象是整个大图，现在考虑研究对象为原图的各个生成子图，将研究对象的规模缩小以便于将其扩展到大图上。本文研究的 k 步可达性问题是在边权值恒为 1 的有向无环图上。对于有向无环图，只是保证了其没有环路，但一个节点可能有多个入边，有多个前驱节点。一个节点有多个前驱节点，这还并不是一个足够简单的图论模型，不便在这种情形下建立与节点相关的数据结构。如果一个节点只有一个前驱节点，并且所有节点对间的路径长度在原始有向无环图上都是最短的，就可以依据之前判断最短距离的方法得出两点间的 k 步不可达性。可以考虑这个理想化的图论模型可以是一棵有向最短路径树 spt。

3.2 倍增索引的构建

定义 1 给定一棵最短路径树，$fa(u,i)$（其中 $0\leqslant i\leqslant\lfloor\lg dep(u)\rfloor, i\in Z$）表示该树上距离节点 u 长度为 2^i 的祖先节点。

说明：距离节点 u 最长的祖先节点为根节点，长度为节点 u 的深度 $dep(u)$，因此由 $fa(u,i)$ 的定义可知，距离节点 u 2^i 的最长的有效祖先节点对应的幂次 i 为 $\lfloor\lg dep(u)\rfloor$，所以 i 的取值范围为从 0 到 $\lfloor\lg dep(u)\rfloor$ 的所有整数。

如图 2 所示，黑色实线是最短路径树上的树边，节点右侧区间内的编号是当前节点 fa 数组索引存储的距离其 2^i（$0\leqslant i\leqslant\lfloor\lg dep(u)\rfloor$）长度的祖父节点编号，黑色虚线是每个节点指向 fa 索引数组存储的祖先节点的指针。1 号节点为根节点，不存在祖先节点，fa 索引数组第二维为空，因此标记为 *。以节点 9 为例，距离节点 9 长度为 2^0 的祖先节点的编号为 8，即 $fa(9,0)=8$；距离节点 9 长度为 2^1 的祖先节点的编号为 7，即 $fa(9,1)=7$；距离节点 9 长度为 2^2 的祖先节点的编号为 5，即 $fa(9,2)=5$；距离节点 9 长度为 2^3 的祖先节点的编号为 1，即 $fa(9,3)=1$。

现在给出 fa 二维索引数组的定理如下：

定理 1 $fa(u,i)=fa(fa(u,i-1),i-1)$（$1\leqslant i\leqslant\lfloor\lg dep(u)\rfloor$）。

证明：令 $fa(u,i)=v$ 表示距离节点 u k（2^i）个单位长度的祖先节点为 v，令 $fa(u,i-1)=v_1$ 表示距离节点 u k_1（2^{i-1}）个单位长度的祖先节点为 v_1，而 $fa(fa(u,i-1),i-1)=fa(v_1,i-1)=v_2$，因为 $fa(u,i)$ 是存在的，所以 $fa(v_1,i-1)$ 距离节点 v_1 k_2（2^{i-1}）个单位长度的祖先节点 v_2 必然是存在的，因此 $i-1\in[0,\lfloor\lg dep(v_1)\rfloor]$。而 $k_1+k_2=2^{i-1}+2^{i-1}=2\cdot 2^{i-1}=2^i=k$，而最短路径树上距离节点 u k 个单位长度的祖先节点是唯一的，即 $v_2=v$，可知 $fa(u,i)=fa(fa(u,i-1),i-1)$，定理 1 得证。

定理 1 同时也是求解倍增索引数组的树形动态规划方程。关于该动态规划的性质分析如下：

(1)初始状态：由定义 1 中 $fa(u,i)$ 的 i 下界可得，最短路径树上距离节点 u 2^0 个单位长度的祖先节点是其父亲节点 $fa(u,0)$。

(2)结束状态：由定义 1 $fa(u,i)$ 的 i 的上界可得，最短路径树上距离节点 u $2^{\lfloor\lg dep(u)\rfloor}$ 的单位长度的祖先节点是 $fa(u,\lfloor\lg dep(u)\rfloor)$。

(3)最优子结构：求解当前问题 0 最短路径树上距离节点 u 的 2^i 个单位长度的祖先节点需要依附于它的子问题 1 即距离节点 u 的 2^{i-1} 个单位长度的祖先节点 v，以及子问题 2 即距离 v 节点 2^{i-1} 个单位长度的祖先节点。当前状态问题 0 是由之前状态子问题 1 和子问题 2 组合而成的。

(4)重叠子问题：最短路径树上求解当前问题 0 上距离当前节点 u 2^i 的单位长度的祖先节点的状态，需要先行求解问题 1 距离节点 u 2^{i-1} 的单位长度的祖先节点 v 的祖先节点的状态，对于多次求解最短路径树除根节点外任意不同的节点 $u\in V_{spt}, i\in[1,\lfloor\lg dep(u)\rfloor]$ 问题 1 是问题 0 的重叠子问题。因此在求解 $fa(u,i)$ 时，可以从根节点出发先行求解当前节点的状态，再去求解子节点的状态，这样可以避免对问题 1 的重复计算，依次类推，直到求解完叶子节点的状态。至此，整棵最短路径树的节点状态求解完成。

(5)无后效性：求解当前状态 $fa(u,i)$ 会直接利用到过去所求解到的状态 $fa(u,i-1)$ 和 $fa(fa(u,i-1),i-1)$，同时也不会影响到过去的状态。

(6)状态转移方程：由最优子结构可得 $fa(u,i)=fa(fa(u,i-1),i-1)$。

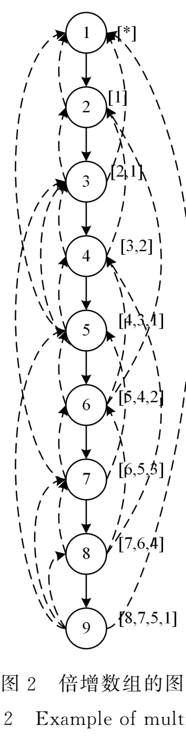

图 2 倍增数组的图示

Fig. 2 Example of multi array

下面给出构造倍增索引的算法描述。

算法 1 构造倍增索引算法 constructMultiIndex(G)

输入：Directed acyclic graph G=(V,E)

输出：(treeId,dep,fa)

```
1. rnk←0
2. for all u∈V(G) do
3.     if deg⁻(u)==0 then
4.         rnk←rnk+1
5.         Set Q an empty queue
6.         Insert u into Q
7.         Set u visited
8.         dep(u)←0
9.         treeId(u)←rnk
10.        while Q is not empty do
11.            Dequeue u from Q
12.                for i=1 to ⌊lgdep(u)⌋ do
13.                    fa(u,i)←fa(fa(u,i−1),i−1)
14.        end for
15.        for all v∈N⁺(u) do
16.            if v is not visited then
17.                Set v is visited
18.                treeId(v)←rnk
19.                dep(v)←dep(u)+1
20.                fa(v,0)←u
21.                Enqueue v onto Q
22.            end if
23.        end for
24.        end while
25.    end if
26. end for
27. return (treeId,dep,fa)
```

算法 1 在通过广度优先搜索生成最短路径森林的同时，构建倍增索引。其中第 12—13 行是根据定理 1 递推求解当前 u 节点的倍增索引 fa 数组；第 17—20 行标记 u 节点的儿子节点 v 已加入 spt，标记节点 v 所在 spt 的编号，求解节点 v 的深度 $dep(v)$，并设置 v 节点 fa 倍增索引数组的第二维的空间大小为$\lfloor \lg dep(v)\rfloor+1$，再设置倍增索引 fa 数组的初始状态 $fa(v,0)=u$。算法其余部分就是标准的通过广度优先搜索来生成最短路径森林。

根据上述构造倍增索引的算法描述，可以得到如图 3 的倍增索引。原始的有向无环图有 2 个入度为 0 的节点 1 和 18，分别得到了以 1 为根节点和以 18 为根节点的最短路径树。其中，黑色实线是构建在最短路径树中的树边，而黑色虚线表示原始有向无环图中存在但未加入最短路径树的边，节点右侧区间内的黑色连续数字标签表示节点的 fa 倍增索引，红色的 2 位数字标签左边数字表示节点的深度，右边数字表示节点所在最短路径树的唯一编号。从图中可看出是先对 1 节点进行 BFS 得到以 1 为根节点的最短路径树。再对 18 进行 BFS 得到以 18 为根节点的最短路径树，在构造 18 为根节点的最短路径树时，会访问到已经加入到 1 节点为根节点的最短路径树中的节点，例如节点 3，12，14，15，16，17，实际都未加入 18 为根的最短路径树，而是以非树边黑色虚线只存在于原先的有向无环图上。

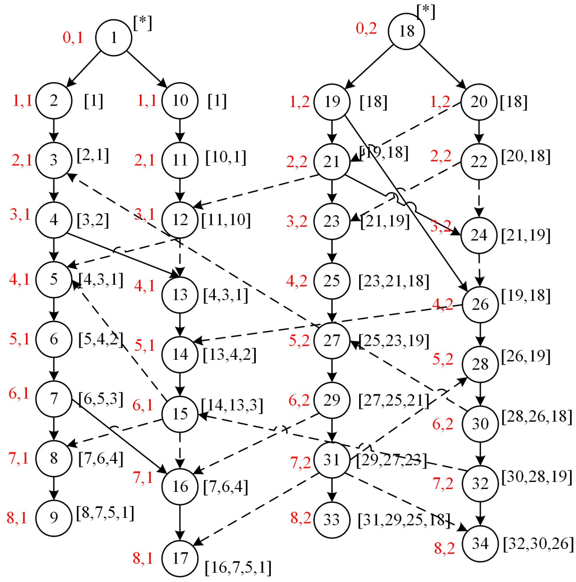

图 3 倍增索引算法构建出的最短路径树森林

Fig. 3 Shortest path tree forest constructed by the multi index algorithm

3.3 不可达判定索引的引入

3.3.1 引入 FELINE 索引

原始 FELINE 索引[21]的构建算法在实现第二次拓扑排序时，入度为零的点的出栈顺序是采用了大根堆按照第一次拓扑序 X 拓扑序较大的节点先出栈，因此 FELINE 索引构造的时间复杂度为 $O(\lg|V||V|+|E|)$。实际上，这个第二次拓扑排序出栈弹出第一次拓扑序 X 较大的节点过程可以不必采用大根堆来实现，可以直接在线性时间内完成。第二次拓扑排序的 Y 数组的构造的具体做法是多开出一个原始 DAG 的邻接表副本，第一次拓扑排序计算 X 数组时按照栈的出栈顺序将出栈节点重新加入其父亲节点的副本邻接表中，这样较大的拓扑序节点就会被加入邻接表较靠后的位置；第二次拓扑排序采用副本邻接表，计算 Y 序号时，由于 X 拓扑序较大的在副本邻接表的较后面，X 拓扑序较大的节点就会后加入栈顶，从而入度为 0 的情况下先出栈，那么对应 X 拓扑序较大的节点，Y 拓扑序自然就偏小。由于第一次拓扑排序计算 X 拓扑序时，对于原始 DAG 中入度为 0 的点不存在父亲节点，因此为了记录这些没有入边的节点的出栈顺序，需要虚拟一个节点 n（原始 DAG 节点编号为 0 至 $n-1$），这些本身没有入边的节点会被加入节点 n 的副本邻接表中。改进后虽然多出了 $O(|V|+|E|)$的空间复杂度，但第二次拓扑排序时间复杂度已降低到线性时间 $O(|V|+|E|)$。

3.3.2 引入 GRAIL 索引

由于 FELINE 索引并不能将原 DAG 中的所有不可达点对都覆盖到，因此同时引入 GRAIL[20]索引来增加不可达点对的覆盖率。

3.4 复杂度分析

生成倍增索引的时间复杂度为 $O(|E|+|V|\lg|V|)$，倍增索引的空间复杂度为 $O(|V|\lg|V|)$。本文改进的构造 FELINE 索引的时间复杂度为 $O(|V|+|E|)$，空间复杂度为 $O(|V|)$。构造 GRAIL 索引的时间复杂度为 $O(|V|+|E|)$，空间复杂度为 $O(|V|)$。

4 基于倍增索引的 k 步可达性查询算法

4.1 基于倍增索引的 k 步可达性查询算法

应用查询算法 2，以图 3 为例。例如查询为(30，15，8)，开始节点 30 和节点 15 并不在同一棵树上，而 $deg^+(30,G) \leqslant deg^-(15,G)$ 按照双向搜索的原则将查询转化为 $query(27,15,7)$，而当前节点 27 和节点 15 仍旧不在同一棵树上，$deg^+(27,G) \leqslant deg^-(15,G)$ 按照双向搜索的原则将查询转化为(3，15，6)，发现节点 3 和节点 15 在同一棵树上，并且 $dep(15) > dep(3)$，$dep(15) - dep(3) = 4 < 6$。根据查询算法 2 验证距离节点 15 4 个单位长度的祖先节点确实是节点 3，推出 $3 \rightarrow_6 15$，进而判定 $30 \rightarrow_8 15$。

再例如查询(26，9，3)，开始节点 26 和节点 9 并不在同一棵树上，而 $deg^+(20,G) > deg^-(9,G)$ 按照双向搜索的原则将查询转化为 $query(26,8,2)$，当前节点 26 和节点 8 仍旧不在同一棵树上，而 $deg^+(26,G) = deg^-(8,G)$，按照双向搜索的原则将查询转化为 $query(14,8,1)$，而当前节点 14 和节点 8 在同一棵最短路径树上，并且 $dep(8) - dep(14) = 2 > 1$ 根据经过查询算法 2 可推出 $14 \nrightarrow_1 8$，进而判定 $26 \nrightarrow_3 9$。

算法 2 基于倍增索引的 k 步可达性查询算法 query(s,t,k)

```
输入:s,t∈V,k
输出:true or false
1. if s=t then
2.         return true
3. end if
4. if k=0 then
5.         return false
6. end if
7. if X(s)>X(t) or Y(s)>Y(t) then
8.         return false
9. end if
10. if L_t^1⊈L_s^1 or L_t^2⊈L_s^2 then
11.         return false
12. end if
13. //若查询点对在同一棵树上，应用倍增索引
14. if treeId(s)=treeId(t) and dep(t)>dep(s) then
15. if dep(t)-dep(s)≤k then
16. 令 ancestor 为距离 t 长度为 dep(t)-dep(s)的祖先节点
17.         if ancestor=s then
18.             return true
19.         end if
20.         else
21. return false
22. end if
23. end if
24. if deg+(s,G)≤deg-(t,G) then
25.         for all v∈N+(s,G) do
26.             if query(v,t,k-1)=true then
27.                 return true
28.             end if
29.         end for
30. else
31.         for all v∈N-(t,G) do
32.             if query(s,v,k-1)=true then
33.                 return true
34.             end if
35.         end for
36. end if
37. return false
```

4.2 正确性分析

性质 1[21] 若 $X(s) > X(t)$ 或者 $Y(s) > Y(t)$，则节点 s 无法到达 t。

性质 2[20] $L_t^1 \not\subseteq L_s^1$ 或者 $L_t^2 \not\subseteq L_s^2$，则节点 s 无法到达 t。

性质 3 若 s 和 t 在同一棵最短路径树上且 $dep(t) - dep(s) > 0$，则在原来的有向无环图中一定不存在 s 到 t 的长度小于 $dep(t) - dep(s)$ 的路径。

证明：第一种情况，因为 $dep(t) - dep(s) > 0$，若最短路径树上存在节点 s 到达节点 t 的路径 l，节点 s 必须是节点 t 的祖先节点，路径 l 的长度为 $dep(t) - dep(s)$。

第二种情况，将不在最短路径树上的非树边构成的节点 s 到达节点 t 的路径定义为 l'，路径 l' 的长度为 $dis(s,t) > 0$。令最短路径树的根节点为 r，由于采用广度优先遍历构造出最短路径树，因此根节点 r 到达节点 s，t 的最短路径就是节点 s，t 所在的深度，即 $dis(r,s) = dep(s)$，$dis(r,t) = dep(t)$。因为 $dep(t) - dep(s) > 0$，而 $dis(s,t) > 0$，可以假设存在路径 l' 的长度 $dis(s,t)$ 小于 $dep(t) - dep(s)$，即 $dis(s,t) < dep(t) - dep(s)$，而 $dis(r,t) = dis(r,s) + dis(s,t)$，将 $dis(s,t) = dis(r,t) - dep(s)$ 代入不等式中，可得 $dis(r,t) < dep(t)$ 与已知条件 $dis(r,t) = dep(t)$ 相矛盾，因此不存在 l' 的路径长度小于 $dep(t) - dep(s)$。证毕。

性质 4 若 s 和 t 在同一棵最短路径树上且 s 是 t 的祖先节点，则在原来的有向无环图中从 s 到 t 存在一条长度为 $dep(t) - dep(s)$ 的最短路径。

证明：首先，假定在原图中从 s 到 t 存在一条长度为 $dep(t) - dep(s)$ 的路径。其次，若在从 s 到 t 的最短路径的这些节点中，最先被算法 1 访问到的节点不是 s，由于原图是有向无环图，所以 s 和 t 必定不在同一棵最短路径树上，但这和前提相矛盾。所以算法 1 最先访问到的节点必是 s。根据广度优先搜索的性质，从 s 到 t 的最短路径的长度就是 $dep(t) - dep(s)$。证毕。

定理 2 算法 2 能够正确判断从 s 到 t 是否存在一条长度不超过 k 的路径。

证明：根据性质 1，算法 2 的第 7－9 行不会产生误判。根据性质 2，第 10－12 行也不会产生误判。在第 14－23 行中，根据性质 3，若 s 和 t 在同一棵最短路径树上，且 $dep(t) - dep(u) > k$，则从 s 到 t 不存在一条长度不超过 k 的路径。根据性质 4，若 s 和 t 在同一棵最短路径树上，且 $dep(t) - dep(u) \leqslant k$ 以及 s 是 t 的祖先节点，则从 s 到 t 存在一条长度不超过 k 的路径。若仍然无法根据上述索引直接判断，第 23－36 行中由于从 s 到 t 的所有路径中，下一个节点必定是 $N^+(s)$ 中的那些节点，所以从 $N^+(s)$ 的节点中进行长度为 $k-1$ 的递归查询；或者，从 s 到 t 的所有路径中，倒数第二个节点必定是 $N^-(t)$ 中的那些节点，所以对 $N^-(t)$ 中的节点进行长度为 $k-1$ 的递归查询。第 37 行，若以上性质都无法判定 $s \rightarrow_{k?} t$，则返回 $s \nrightarrow_k t$。这样可以保证不会有任何的遗漏。

所以算法 2 能够正确判断从 s 到 t 是否存在一条长度不超过 k 的路径。证毕。

4.3 复杂性分析

性质 5 在最短路径树上,距离节点 v 长度为 k 的祖先节点为 u,若 k 的二进制表示中"1"所出现的位的幂次从高到低分别是 $\lfloor \lg k \rfloor = x_1 \geqslant \cdots \geqslant x_m \geqslant 0$,则 $u = fa(\cdots fa(fa(v, x_1), x_2), \cdots, x_m)$。

证明:对于通过 $fa(\cdots fa(fa(v,x_1),x_2),\cdots x_m)$ 多次求取距离当前节点长度为 $2^{x_i}(1\leqslant i\leqslant m)$ 的祖先节点,记当前节点为 a,因为 $x_i(1\leqslant i\leqslant m)=\lfloor \lg(dep(a)-dep(u))\rfloor$,而 $\lfloor \lg(dep(a)-dep(u))\rfloor \in [0, \lfloor \lg dep(a)\rfloor]$,所以 $x_i(1\leqslant i\leqslant m)$ 在 $fa(a,j)(0\leqslant j\leqslant[0,\lfloor \lg dep(a)\rfloor])$ 的二维下标范围内总是存在的。而 $fa(v,x_1)$ 返回的是距离 v 长度为 2^{x_1} 的祖先节点,$fa(fa(v,x_1),x_2)$ 返回的是距离 v 长度为 $2^{x_1}+2^{x_2}$ 的祖先节点,$fa(\cdots fa(fa(v,x_1),x_2),\cdots,x_m)$ 返回的是距离 v 长度为 $(2^{x_1}+2^{x_2}+\cdots+2^{x_m}=k)$ 的祖先节点 u。证毕。

根据上述性质,算法 2 第 16 行的运行时间为 $O(\lg k)$,第 7-12 行应用 FELINE 索引和 GRAIL 索引进行不可达判定的时间复杂度为 $O(1)$。就整个查询算法而言,由于最坏情况下算法会穷举所有可能的长度为 k 的路径,但从下一章的实验性能上来看,该算法还是具有非常优异的实际表现的。

5 实验

5.1 实验环境

实验所用的计算机基本配置为 Intel Core i7-8750HCPU@2.20GHz (12 CPUs),DDR4 16GB 内存,512GB 固态硬盘,操作系统为 Windows 10 64 位 20H2,编程语言为 C++11,集成开发环境为 CodeBlocks 17.12。实验用于比较的算法基于 2-hop 框架最短路径的 PLL 算法和基于节点覆盖索引子图的 k-reach 算法,算法都是在 CodeBlocks 17.12 C++11 环境下实现。

5.2 实验数据集

实验数据由 19 个数据集构成,这些数据集均保证了是有向无环图。其中,agroCyc,anthra,ecoo,human,mtbrv,vchocyc 数据集来自 ecocyc.org 描述大肠杆菌 k-12MG1655 的基因组和生化机制;amaze 和 kegg 数据集描述代谢网络,nasa 和 xmark 数据集描述 XML 文档的结构;arxiv,citeseer,pubmed,citeseerx,citpatents 数据集描述引用网络;go 数据集描述基因本体图;yago 数据集描述语义知识数据库中术语间关系的结构;go-uniprot 数据集描述基因本体论的联合图和 UniProt 数据库的注释关系;uniprotenc_22m 数据集是 UniProt 的完全 RFG 图的子集。这些数据集的相关信息如表 2 所列,其中 $|V|$ 表示有向无环图的节点的个数,$|E|$ 表示图中边的个数,$inDeg_{\max}$ 表示图中节点的最大入度,$outDeg_{\max}$ 表示图中节点的最大出度。

如表 2 所列,前 13 个数据集是较小的图,节点数不超过 10^5,而后 6 个数据集是较大的图,节点数最大可达六七百万,其中 yago,arxiv,pubmed,go-uniprot,citeseerx,citpatents 是较为稠密的数据集。

可达性查询算法的效率要从构建索引的大小、构建索引的时间、查询时间 3 个方面度量,而 k 步可达性查询在查询时间上的度量还需特别测试不同 k 值下查询的性能好坏,这样才能更全面地度量查询算法的有效性。查询实验是以查询 10^5 个点对的总时间来度量的,为了保证实验结果的准确性,实际查询时间记录的是测试 10 次后的平均时间。

表 2 数据集统计信息

Table 2 DatasetStatistics

数据集	$\|V\|$	$\|E\|$	$inDeg_{\max}$	$outDeg_{\max}$
amaze	3710	3947	1237	1860
anthra	12499	13327	486	5400
human	38811	39816	553	28570
yago	6642	42392	2370	55
agrocyc	12684	13657	571	5487
nasa	5605	6538	11	30
xmark	6080	7051	192	803
ecoo	12620	13575	542	5434
mtbrv	9602	10438	467	4004
vchocyc	9491	10345	486	3916
kegg	3617	4395	1234	2048
arxiv	6000	66707	695	173
go	6793	13361	6	70
uniprotenc_22m	1595444	1595442	1539898	1
pubmed	9000	40028	432	87
citeseer	693947	312282	55757	1
go-uniprot	6967956	34770235	1186281	170
citeseerx	6540401	15011260	203695	181247
citpatents	3774768	16518947	779	770

5.3 构造索引的时间

表 3-5 列出了对比算法 PLL 和 k-reach 以及本文 MultiIndex 倍增索引构造索引的时间。如表 3-5 所列,在小规模数据集上,MultiIndex 构建索引的时间和 k-reach 是相差不大的,但明显比 PLL 小一个数量级。对于 arxiv 异常稠密的数据集,MultiIndex 构造时间几乎比 PLL 和 k-reach 小 2 个数量级。对于大规模数据集,PLL 算法已经不能在规定的有效时间内构造出索引,因此 PLL 算法并不能适用于大图,而 MultiIndex 构造时间同样要比 k-reach 短一些。对于 citeseerx 和 citpatents 数据集,MultiIndex 要比 k-reach 小一个数量级的。

表 3 构造索引时间 $k=3$

Table 3 Time of construction index $k=3$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	78.83	79.79	**8.98**
anthra	815.16	27.96	**21.01**
human	7513.94	74.82	**62.87**
yago	273.86	**16.94**	31.91
agrocyc	832.25	30.92	**22.94**
nasa	182.57	20.50	**14.96**
xmark	210.44	27.93	**13.96**
ecoo	830.20	30.93	**20.94**
mtbrv	487.27	25.95	**16.99**
vchocyc	474.29	24.95	**15.99**
kegg	74.84	88.11	**8.93**
arxiv	1231.24	757.98	**32.94**
go	277.35	41.89	**26.93**
uniprotenc_22m	—	5769.01	**5161.24**
pubmed	511.54	115.72	**32.91**
citeseer	—	4498.39	**2075.49**
go-uniprot	—	**22566.08**	32986.02
citeseerx	—	53937.37	**16762.19**
citpatents	—	39434.47	**18027.42**

表 4 构造索引时间 $k=5$

Table 4 Time of construction index $k=5$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	78.83	89.83	**8.98**
anthra	815.16	35.97	**21.01**
human	7513.94	82.82	**62.87**
yago	273.86	**17.99**	31.91
agrocyc	832.25	37.92	**22.94**
nasa	182.57	27.94	**14.96**
xmark	210.44	57.88	**13.96**
ecoo	830.20	40.34	**20.94**
mtbrv	487.27	29.84	**16.99**
vchocyc	474.29	29.93	**15.99**
kegg	74.84	109.26	**8.93**
arxiv	1231.24	2530.24	**32.94**
go	277.35	53.89	**26.93**
uniprotenc_22m	—	5799.57	**5161.24**
pubmed	511.54	195.67	**32.91**
citeseer	—	4563.84	**2075.49**
go-uniprot	—	**23247.77**	32986.02
citeseerx	—	71543.64	**16762.19**
citpatents	—	87009.74	**18027.42**

表 5 构造索引时间 $k=7$

Table 5 Time of construction index $k=7$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	78.83	96.73	**8.98**
anthra	815.16	39.89	**21.01**
human	7513.94	93.69	**62.87**
yago	273.86	**19.95**	31.91
agrocyc	832.25	39.90	**22.94**
nasa	182.57	35.76	**14.96**
xmark	210.44	77.73	**13.96**
ecoo	830.20	41.45	**20.94**
mtbrv	487.27	35.96	**16.99**
vchocyc	474.29	34.90	**15.99**
kegg	74.84	114.76	**8.93**
arxiv	1231.24	3379.99	32.94
go	277.35	60.50	**26.93**
uniprotenc_22m	—	5755.46	**5161.24**
pubmed	511.54	202.55	**32.91**
citeseer	—	4522.90	**2075.49**
go-uniprot	—	**22377.82**	32986.02
citeseerx	—	79369.51	**16762.19**
citpatents	—	161817.80	**18027.42**

5.4 构造索引的大小

表6－表8列出了对比算法PLL和k-reach以及本文MultiIndex倍增索引构造索引的大小。如表6－表8所列，在小规模数据集上，PLL和MultiIndex索引规模相差不大，而在大多数数据集上k-reach算法索引较小。而对于arxiv异常稠密的数据集，k-reach索引规模要远大于PLL和MultiIndex。而对于大规模数据集，PLL算法不能在有效时间构造出索引，其无法应用于大图。

表 6 构造索引大小 k=3

Table 6 Size of construction index $k=3$

(Mb)

数据集	PLL	k-reach	MultiIndex
amaze	**0.09**	0.26	0.14
anthra	0.30	**0.05**	0.57
human	0.91	**0.08**	1.78
yago	0.51	**0.06**	0.24
agrocyc	0.31	**0.06**	0.58
nasa	0.22	**0.06**	0.27
xmark	0.25	**0.11**	0.27
ecoo	0.31	**0.07**	0.58
mtbrv	0.24	**0.05**	0.44
vchocyc	0.24	**0.05**	0.44
kegg	**0.09**	0.30	0.14
arxiv	2.68	7.22	**0.25**
go	0.41	**0.18**	0.30
uniprotenc_22m	—	**2.12**	54.93
pubmed	0.70	1.10	**0.34**
citeseer	—	**1.83**	24.18
go-uniprot	—	**8.19**	239.34
citeseerx	—	**17.28** *	255.25
citpatents	—	**50.17** *	151.56

表 7 构造索引大小 $k=5$

Table 7 Size of construction index $k=5$

(Mb)

数据集	PLL	k-reach	MultiIndex
amaze	**0.09**	0.31	0.14
anthra	0.30	**0.06**	0.57
human	0.91	**0.09**	1.78
yago	0.51	**0.07**	0.24
agrocyc	0.31	**0.07**	0.58
nasa	0.22	**0.11**	0.27
xmark	0.25	**0.22**	0.27
ecoo	0.31	**0.08**	0.58
mtbrv	0.24	**0.06**	0.44
vchocyc	0.24	**0.06**	0.44
kegg	**0.09**	0.40	0.14
arxiv	2.68	17.18	**0.25**
go	0.41	**0.27**	0.30
uniprotenc_22m	—	**2.12**	54.93
pubmed	0.70	1.82	**0.34**
citeseer	—	**1.91**	24.18
go-uniprot	—	**9.09**	239.34
citeseerx	—	**18.57** *	255.25
citpatents	—	153.89 *	**151.56**

表 8 构造索引大小 $k=7$

Table 8 Size of construction index $k=7$

(Mb)

数据集	PLL	k-reach	MultiIndex
amaze	**0.09**	0.32	0.14
anthra	0.30	**0.06**	0.57
human	0.91	**0.09**	1.78
yago	0.51	**0.07**	0.24
agrocyc	0.31	**0.07**	0.58
nasa	0.22	**0.15**	0.27
xmark	**0.25**	0.33	0.27
ecoo	0.31	**0.08**	0.58
mtbrv	0.24	**0.06**	0.44
vchocyc	0.24	**0.06**	0.44
kegg	**0.09**	0.44	0.14
arxiv	2.68	21.65	**0.25**
go	0.41	**0.30**	**0.30**
uniprotenc_22m	—	**2.12**	54.93
pubmed	0.70	1.96	**0.34**
citeseer	—	**1.93**	24.18
go-uniprot	—	**9.39**	239.34
citeseerx	—	**18.65** *	255.25
citpatents	—	243.31 *	**151.56**

对于 citeseerx 和 citpatents,大规模并不稠密的数据集实际 k-reach 的极小顶点覆盖数非常大,按照原始算法并不能在有效时间构造出索引,实验时只构造其部分节点覆盖进而将其扩展到大图,缩短了索引子图规模,牺牲了查询时的效率,因此实际其极小顶点覆盖的索引子图规模比 MultiIndex 要大很多。后续的实际查询测试中,k-reach 在 citeseerx 和 citpatents 的效率要远低于 MultiIndex。

5.5 查询时间

表 9—表 11 列出了对比算法 PLL 和 k-reach 以及本文 MultiIndex 倍增索引的查询时间。

表 9 查询时间 $k=3$

Table 9 Time of query index $k=3$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	6.49	17.95	**5.98**
anthra	4.98	10.97	**2.98**
human	6.06	9.91	**2.99**
yago	12.04	41.88	**6.99**
agrocyc	4.95	11.99	**2.99**
nasa	7.18	17.95	**2.00**
xmark	7.98	16.95	**4.03**
ecoo	5.98	10.97	**1.99**
mtbrv	4.99	11.97	**1.96**
vchocyc	5.95	10.98	**2.00**
kegg	6.95	20.94	**5.02**
arxiv	**67.93**	111.74	126.66
go	**14.96**	32.90	18.95
uniprotenc_22m	—	20.94	**6.98**
pubmed	**38.90**	63.83	96.63
citeseer	—	**34.91**	40.89
go-uniprot	—	99.77	**69.78**
citeseerx	—	769.87	**222.39**
citpatents	—	653.78	**490.72**

表 10 查询时间 $k=5$

Table 10 Time of query index $k=5$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	6.49	18.93	**5.99**
anthra	4.98	11.96	**2.99**
human	6.06	10.96	**2.98**
yago	12.04	44.89	**8.97**
agrocyc	4.95	10.97	**2.99**
nasa	7.18	18.95	**2.99**
xmark	7.98	16.96	**4.99**
ecoo	5.98	11.97	**3.03**
mtbrv	4.99	11.96	**2.96**
vchocyc	5.95	11.97	**3.03**
kegg	6.95	20.94	**5.98**
arxiv	**67.93**	118.69	1 076.09
go	**14.96**	32.91	19.95
uniprotenc_22m	—	20.94	**7.89**
pubmed	**38.90**	64.83	232.34
citeseer	—	**35.93**	41.92
go-uniprot	—	97.74	**90.76**
citeseerx	—	1 221.77	**422.87**
citpatents	—	**4 728.32**	**2 874.31**

表 11 查询时间 $k=7$

Table 11 Time of query index $k=7$

(ms)

数据集	PLL	k-reach	MultiIndex
amaze	6.49	18.95	**6.02**
anthra	4.98	11.98	**3.93**
human	6.06	11.01	**2.96**
yago	12.04	45.91	**8.98**
agrocyc	4.95	11.93	**2.99**
nasa	7.18	18.98	**3.98**
xmark	**7.98**	17.95	8.01
ecoo	5.98	11.97	**2.99**
mtbrv	4.99	11.97	**3.00**
vchocyc	5.95	11.97	**3.04**
kegg	6.95	20.94	**6.02**
arxiv	**67.93**	115.73	3 102.74
go	**14.96**	31.95	20.94
uniprotenc_22m	—	20.94	**8.98**
pubmed	**38.90**	57.85	370.91
citeseer	—	**35.90**	41.89
go-uniprot	—	**96.71**	98.73
citeseerx	—	4 234.36	**673.17**
citpatents	—	31 768.16	**9 825.92**

如表 9—表 11 所列,在小规模数据集上,MultiIndex 和 PLL 及 k-reach 的查询时间相差不大。但对于 arixv 这个异常稠密的数据集,k-reach 的查询效率要比 MultiIndex 快一个数量级。PLL 算法无法应用于 uniprotenc_22m 等大的数据集,其索引无法在有效时间内构造出来。go-uniprot 数据集上 k-reach 和 MultiIndex 查询效率是相差不多的。但对于 citeseerx 和 citpatents,MultiIndex 要远好于 k-reach。

结束语 本文针对大图上的 k 步可达性查询问题中已有算法大图索引规模较大,查询效率较低,且 k 值通识性较差的问题,提出了基于倍增索引的 k 步可达性查询算法。为了将问题扩展到大图,提出了在各个生成子图上构建独立索引,降低索引规模。为避免穷举搜索所有路径,在最短路径树上构建倍增索引进行 k 步可达性的判定,并给出关于最短路径树和倍增索引的相关定理、性质以及证明作为查询算法的正确性的保证。之后又通过构建不可达索引 FELINE 和 GRAIL 来进行不可达判定的剪枝,并改进了 FELINE 索引的构造算法,将构造时间降至线性。接着提出倍增索引查询算法,及其查询算法正确性相关的证明,并分析相关时间、空间复杂度。最后在 19 个真实的数据集上进行实验,验证了倍增索引在索引构造时间、索引大小、查询时间的高效性。本文提出的倍增索引实际比较容易扩展至动态图以支持加边、删边操作,只需对最短路径树的局部分支进行更新,这是本文在实际应用上的价值体现。

参 考 文 献

[1] FU L Z,MENG X F. Reachability indexing for large-scale-graphs:studies andforecasts[J]. Journal of Computer Research and Development,2015,52(1):14.

[2] ZHANG Y,LIU Y B,XIONG G,et al. Survey on Succinct Representation of Graph Data[J]. Journal of Software,2014,25(9):16.

[3] DING Y,ZHANG Y,LI Z H,et al. Research and advances on graph data mining[J]. Journal of Computer Applications,2012,32(1):9.

[4] WEISS M A. Data structures and algorithm analysis in C [M]. [S. 1]. Benjamin/Cummings, 1993.

[5] STANN F, HEIDEMANN J. Rmst: Reliable data transport in sensor networks[C]// Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003. IEEE, 2003: 102-112.

[6] TAKUYA A, YOICHI I, YUICHI Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling [C]// SIGMOD Conference. ACM, 2013: 349-360.

[7] CHENG J, SHANG Z, CHENG H, et al. Efficient processing of k-hop reachability queries[J]. VLDB J, 2014, 23(2): 227-252.

[8] COHEN E, HALPERIN E, KAPLAN H, et al. Reachability and distance queries via 2-hop labels [C]// Proceedings of the Thirteenth Annual ACMSIAM Symposium on Discrete Algorithms. ACM/SIAM, 2002: 937-946.

[9] KING V. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs[C]// 40th Annual Symposium on Foundations of Computer Science (FOCS '99). IEEE Computer Society, 1999: 81-91.

[10] JIN R, WANG G. Simple, fast, and scalable reachability oracle [J]. Proc. VLDB Endow., 2013, 6(14): 1978-1989.

[11] CHENG J, HUANG S, WU H, et al. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2013). ACM, 2013: 193204.

[12] ZHU A D, LIN W, WANG S, et al. Reachability queries on large dynamic graphs: a total order approach[C]// International Conference on Management of Data (SIGMOD 2014). ACM, 2014: 13231334.

[13] AGRAWAL R, BORGIDA A, JAGADISH H V. Efficient management of transitive relationships in large data and knowledge bases[C]// Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data. ACM, 1989: 253-262.

[14] WANG H, HE H, YANG J, et al. Duallabeling: Answering graph reachability queries in constant time [C]// Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006). IEEE Computer Society, 2006: 75.

[15] TRISSL S, LESER U. Fast and practical indexing and querying of very large graphs[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data. ACM, 2007: 845-856.

[16] CHEN L, GUPTA A, KURUL M E. Stack based algorithms for pattern matching on dags[C]// Proceedings of the 31st International Conference on Very Large Data Bases. ACM, 2005: 493-504.

[17] JIN R, XIANG Y, RUAN N, et al. 3-hop: a high-compression indexing scheme for reachability query[C]// Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2009). ACM, 2009: 813826.

[18] JIN R, RUAN N, XIANG Y, et al. Path-tree: An efficient reachability indexing scheme for large directed graphs [J]. ACM Trans. Database Syst., 2011, 36(1): 7: 1-7: 44.

[19] CAI J, POON C K. Path-hop: efficiently indexing large graphs for reachability queries[C]// Proceedings of the 19th ACM Conference on Information and Knowledge Management (CIKM 2010). ACM, 2010: 119-128.

[20] YILDIRIM H, CHAOJI V, ZAKI M J. GRAIL: scalable reachability index for large graphs[J]. Proc. VLDB Endow., 2010, 3(1): 276-284.

[21] VELOSO R R, CERF L, JR W M, et al. Reachability queries in very large graphs: A fast refined on line search approach[C]// Proceedings of the 17th International Conference on Extending Database Technology (EDBT 2014). 2014: 511-522.

[22] SEUFERT S, ANAND A, BEDATHUR S J, et al. FERRARI: flexible and efficient reachability range assignment for graph indexing[C]// 29th IEEE International Conference on Data Engineering (ICDE 2013). IEEE Computer Society, 2013: 1009-1020.

[23] WEI H, YU J X, LU C, et al. Reachability querying: An independent permutation labeling approach [J]. Proc. VLDB Endow., 2014, 7(12): 1191-1202.

[24] JIN R, RUAN N, XIANG Y, et al. A highwaycentric labeling approach for answering distance queries on large sparse graphs [C]// Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2012). USA, ACM, 2012: 445-456.

[25] WONG T. Answering reachability queries on incrementally updated graphs by hierarchical labeling schema[J]. J. Comput. Sci. Technol., 2016, 31(2): 381-399.

TONG Zhengnan, born in 1997, master. His main research interests include reachability query and graph data management.

BU Tianming, born in 1977, associate professor. His main research interests include algorithmic design and analysis, and algorithmic game theory.