文章编号：2095-2163(2021)01-0008-07 中图分类号：TP301.6 文献标志码：A

有向图上 k 步可达查询处理

杜 明，林 铿，周军锋
（东华大学 计算机科学与技术学院，上海 201620）

摘 要： 给定一个有向图，一个 k 步可达查询 $u\rightarrow_{?k}v$ 用来回答在该图中是否存在一条从顶点 u 到顶点 v 且长度不大于 k 的有向路径。k 步可达查询是一种基本的图操作并在过去十年间被广泛地研究。已有的 k 步可达查询算法仍存在许多弊端，例如不可达查询效率低，索引规模大和索引构建时间长等。本文针对上述问题提出了 2 种优化方法，分别是基于互逆拓扑序号以及基于等价顶点的图压缩方法. 前者提高了不可达查询的效率，后者减少了索引规模和索引构建时间。实验结果表明，本文提出的方法可以有效地处理 k 步可达查询，并支持大规模数据的处理。

关键词： 有向图；k 步可达查询；图压缩

k-hop reachability query processing on a directed graph

DU Ming, LIN Keng, ZHOU Junfeng
(School of Computer Science and Technology, Donghua University, Shanghai 201620, China)

【Abstract】 Given a directed graph, a k-hop reachability query $u\rightarrow_{?k}v$ is used to answer whether there is a directed path from vertex u to vertex v and the length of the path is not greater than k. The k-hop reachability query is a basic graph operation and has been extensively studied in the past years. Existing algorithms still have many drawbacks, such as being inefficient for unreachability queries, large index size and long index construction time. This paper proposes two optimization approaches to make improvements, i.e., the mutual reversed topological order and the graph compression based on equivalent vertices. The former improves the efficiency of unreachability queries, and the latter reduces the index size and index construction time. The experimental results show that the proposed method can effectively improve the performance of k-hop reachability queries processing and support large-scale graph processing.

【Key words】 directed graph; k-hop reachability query; graph compression

0 引 言

随着互联网的快速发展，各种数据的规模日益庞大。图是一种常见的数据表示模型，其中每个实体被简单地抽象成图中的一个顶点，实体间的关系被抽象成 2 个顶点之间的一条边。图被广泛地应用于各类领域，如社交网络、通信网络、交通网络等等[1-3]。在图模型上的一个基本操作是回答 2 个顶点之间的可达性查询，即判断在图中是否存在一条从源顶点 u 出发到目标顶点 v 结束的一条有向路径。可达性查询在过去被广泛地研究[4-8]，然而在实际应用中，可达性查询仅能回答 2 个实体之间是否存在某种关系，而无法回答这种关系的强弱程度。

另一种更有价值的操作是回答 2 个顶点之间的 k 步可达查询[9-10]，即判断在图中是否存在一条从源顶点 u 出发到目标顶点 v 结束的一条有向路径，并且满足该路径的长度不超过 k。k 步可达查询相较于可达性查询能提供更多关于 2 个顶点之间的信息，本质上，可达性查询是一种特殊的 k 步可达查询，即当 k 取 ∞ 时。k 步可达查询在现实生活中有很多实际的应用。比如在交通网络中，经常需要回答在 2 个地点之间是否存在一条路程不超过某个阈值的路线。又比如在社交网络中，通过 2 个实体之间的距离来判断这 2 个实体产生关联的可能性。

现有 k 步可达查询算法大多只能运行在有向无环图上，而有向图上的 k 步可达查询的相关研究却寥寥无几。在有向图上回答 k 步可达查询的基本方法是 BFS[11] 或者 DFS [12]。基于遍历的方法因为不具备良好的扩展性，因而查询效率低下。另一个简单的实现方法是将任意 2 个顶点之间的最短路径信

基金项目：国家重点研发计划(2017YFB0309800)。

作者简介：杜 明(1975-)，男，博士，副教授，主要研究方向：自然语言处理、信息查询、数据分析；林 铿(1995-)，男，硕士研究生，主要研究方向：图的可达性查询、极大团枚举；周军锋(1977-)，男，博士，教授，博士生导师，主要研究方向：大图数据的查询处理技术、推荐系统关键技术。

通讯作者：周军锋 Email：zhoujf@dhu.edu.cn

收稿日期：2020-11-11

息预先通过邻接矩阵的方式存储起来，回答查询时仅需要 $O(1)$ 的时间内即可返回结果。但是这种方法需要的空间代价过大，无法应用于大规模数据图。

针对上述算法存在的问题，文献[9]提出了一种基于 2-hop 索引 PLL。PLL 的基本思想是通过为每个顶点预先建立一组出标签和一组入标签，这 2 组标签分别保存了部分顶点与该顶点之间的最短路径信息，这些部分顶点被称为 hop 点。2 个顶点之间的 k 步可达查询问题可以被转化为判断两点间最短路径的长度是否小于等于 k 的问题。

PLL 算法本身也存在低效性的问题。首先，如果一个查询顶点对之间是不可达的，那么 PLL 算法需要遍历源顶点的所有出标签以及目标顶点的所有入标签才能判断出这对查询顶点对是不可达的。其次，PLL 算法在所有的顶点上构建索引，因此导致构建的索引的时间较长，同时构建的索引规模也较大。

针对 PLL 算法存在的以上 2 种问题，本文提出了 2 种针对性的优化方法。第一种是基于互逆拓扑序号来加快不可达查询的效率，第二种是基于等价顶点的图压缩方法，通过去除图中的冗余顶点和冗余边，使得图尽可能地小。实验结果表明，本文提出的优化方法可以显著改善索引的构建速度，索引规模更小。

1　相关工作

1.1　问题定义

给定一个有向图 $G=(V,E)$，其中 $V=\{v_1,v_2,\cdots,v_n\}$ 是图中顶点的集合，$E=\{(u,v)\mid u,v\in V\}$ 是图中边的集合。当一条边 $e=(u,v)\in E$ 时，表示存在一条从顶点 u 指向顶点 v 的有向边。对于每个顶点 u，本文使用 $scc(u)$ 来表示顶点 u 所在的强连通分量。同时使用 $out_G(u)=\{v\mid(u,v)\in E\}$ 来表示顶点 u 的出邻居集合以及使用 $in_G(u)=\{v\mid(v,u)\in E\}$ 表示顶点 u 的入邻居集合。用 $deg_{out}(u)=|out_G(u)|$ 表示顶点 u 的出度，$deg_{in}(u)=|in_G(u)|$ 表示顶点 u 的入度。

在一个有向无环图中，拓扑排序是将图中顶点从偏序状态转化为全序状态。一个顶点 u 的拓扑序号是顶点全序关系的序号，拓扑序号大的顶点与拓扑序号小的顶点之间不存在有向路径。

如果 2 个顶点 u,v 满足 $out_G(u)=out_G(v)$ 并且 $in_G(u)=in_G(v)$，即顶点 u 和顶点 v 存在相同的出邻居集合以及相同的入邻居集合，则称这 2 个顶点互为等价顶点。直观上，2 个顶点等价意味着二者可达相同的顶点集合，且可达二者的顶点集合也相同。

问题定义　给定一个有向图 G 和一个 k 步可达查询 $u\rightarrow_{?k}v$，如果在 G 中存在一条从 u 出发到 v 的有向路径，并且该路径的长度不超过 k，则返回 TRUE，否则返回 FALSE。

1.2　相关算法

1.2.1　PLL 算法

PLL(Pruned Landmark Labeling)算法的基本思想是为图中的每一个顶点 u 预先构建 2 组标签 $L_{out}(u)=\{(h_1,d_1),\cdots,(h_i,d_i)\}$ 和 $L_{in}(u)=\{(h_1,d_1),\cdots,(h_j,d_j)\}$。其中，$L_{out}(u)$ 保存了从顶点 u 可以到达的部分顶点以及顶点 u 到这些顶点之间的距离，类似地，$L_{in}(u)$ 保存了从部分可以到达顶点 u 的顶点以及这些顶点到顶点 u 之间的距离。每个顶点标签中的每一对元素 (h,d) 表示 hop 点 h 以及 h 和该顶点之间的最短距离 d。

PLL 算法将 2 个查询顶点对之间的 k 步可达查询转化为了判断源顶点 u 和目标顶点 v 的最短距离是否小于等于 k 的问题。由于标签中的 hop 点序号是以升序排列的，因此判断 2 个标签中是否存在一个公共的 hop 点的时间复杂度为 $O(n+m)$，其中 n，m 分别表示出标签和入标签中元素的个数。

PLL 算法使用剪枝策略保证了建立的索引中尽可能地减少冗余的最短路径信息。具体的做法是在对每一个 hop 点进行 BFS 遍历的过程中，如果已经建立的 2-hop 索引能够回答该 hop 点到当前遍历的顶点之间的最短路径信息，则 PLL 算法不会将该 hop 点加入到当前遍历的顶点的 2-hop 标签中，同时不再从当前遍历的顶点执行 BFS 遍历。

1.2.2　K-Reach 算法

K-Reach 算法[9,13]的基本思想是首先求解有向图 G 的最小顶点覆盖集 C，然后在求解得到的 C 上建立传递闭包，这个传递闭包只包含了 C 中各个顶点之间的可达信息。

当回答一个 k 步可达查询 $u\rightarrow_{?k}v$ 时，根据查询顶点对 u,v 是否属于 C 分为如下 3 种情况：

(1)顶点 u 和顶点 v 都属于最小顶点覆盖集，此时可以直接通过建立的传递闭包来回答查询。

(2)顶点 u 和顶点 v 中只有一个属于 C，在这种情况下需要通过遍历那个非最小顶点覆盖集顶点的出邻居集合的传递闭包或者入邻居集合的传递闭包来回答查询。

(3)顶点 u 和顶点 v 中都不属于 C，这是最坏的情况，需要遍历顶点 u 的出邻居集合的传递闭包以

及 v 的入邻居集合的传递闭包来回答查询。

K-Reach 算法存在如下问题:首先该算法构建的传递闭包索引只能用于回答 k 等于特定值的 k 步可达查询。其次,当查询顶点对不属于最小顶点覆盖集时,需要遍历查询顶点对的所有邻居顶点的传递闭包。最后,随着 k 值的增大,每个顶点的传递闭包数量将呈指数上升,因此该算法将无法有效地扩展到大图上。

2 优化方法

2.1 基于互逆拓扑序号的查询优化方法

在有向无环图中,一个顶点的拓扑序号是该顶点在拓扑排序中被处理的顺序。如果一个顶点的拓扑序号大于另一个顶点的拓扑序号,则前者必然不可达后者,反之不成立。因此,在一个有向无环图中,可以通过判断 2 个顶点的拓扑序号大小来快速地回答 2 个顶点间的不可达情况。

但是在有向图中,由于可能存在强连通分量,而拓扑排序必须在无环图中才能进行,因此必须对有向图进行一定的变换。在本文中采取的解决方案是将每一个强连通分量(SCC)压缩成一个超级顶点,从而将有向图转换成有向无环图。Tarjan 算法是用来求解强连通分量的经典算法,该算法可以在线性时间内求解出每个顶点所属的强连通分量。对于属于同一个强连通分量中的顶点,将为其赋予相同的拓扑序号。

在拓扑排序中,对于一个顶点来说,不同的处理顺序可以生成不同的拓扑序号,一个直观的想法就是为每个顶点设置多个拓扑序号从而过滤掉等多的不可达查询。然而,每个拓扑序号都需要一个整型的存储空间,过多的拓扑序号会导致索引规模增长。因此,本文提出构建互逆的拓扑序号来达到最佳的查询效率以及较小的空间开销。

互逆拓扑序号的大致思想是为每个顶点建立 2 个拓扑序号。如果建立顶点 u 的第一个拓扑序号要先于建立顶点 v 的第一个拓扑序号,则在建立第二个拓扑序号时,要优先处理顶点 v 的拓扑序号。这种机制保证了每个顶点的 2 个拓扑序号构成的区间尽可能地大,从而可以判断等多的不可达情况。至此,研究可得算法 1、算法 2 的代码设计详见如下。

算法 1 $genTopo(G=(V,E))$

输入:$G=(V,E)$

输出:所有顶点的互逆拓扑序号

```
1  G' ← tarjan(G)
2  construct(G',X)
3  construct(G',Y)
4  return (X,Y)
```

算法 2 $construct(G=(V,E),T)$

```
1  将入度为 0 的顶点按照特定的顺序入栈 S
2  topNum ← 0
3  while S≠ϕ do
4     u ← pop(S)
5     T_u ← topNum
6     topNum ← topNum + 1
7     for each v ∈ out_G(u) do
8        deg_in(v) ← deg_in(v) -1
9        if deg_in(v) = 0 do
10          push(S,v)
```

算法 1 展示了在一个有向图上求解互逆拓扑序号的过程。算法的输入是 G,输出是 G 上所有顶点的互逆拓扑序号。第 1 行首先调用 Tarjan 算法将 G 中的强连通分量压缩成超级顶点。第 2 ~ 3 行分别调用 *construct* 方法建立拓扑序号。

在算法 2 的 *construct* 方法中,第 1 行将入度为 0 的顶点按照特定的顺序入栈 S,第 2 行设置一个计数器 *topNum*,用来记录每个顶点的处理顺序。第 3 行当栈 S 不为空时,执行第 4 ~ 10 行的操作。第 4 行先弹出栈首顶点 v,并在第 5 行将顶点 v 的拓扑序号设置为 *topNum*,同时第 6 行更新 *topNum*。第 7 ~ 10 行按照特定的顺序处理顶点 v 的所有出邻居,将其入度均做减一处理,如果减后的入度为 0,则将该出邻居顶点入栈。

对于在同一个强连通分量中的每个顶点,各顶点的互逆拓扑序号都等于相应顶点所在的超级顶点的互逆拓扑序号。Tarjan 算法和 *construct* 方法的时间复杂度都为 $O(m)$,其中 m 为有向图 G 的边数,因此整个 genTopo 方法的时间复杂度为 $O(m)$。

2.2 基于等价顶点的图压缩方法

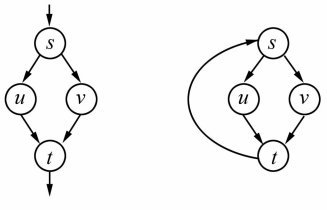
在 1.1 节中给出了等价顶点的相关定义。对于任意 2 个相互等价的顶点 u、v,在回答可达性查询时(除查询顶点对为 (u,v) 这种特殊情况,稍后讨论)可以相互替换。例如当回答从顶点 u 到顶点 w(其中 $w\neq v$)的 k 步可达查询时,可以替换成从顶点 v 到顶点 w 的 k 步可达查询。

根据该特性,可以实现基于等价顶点的图压缩算法。具体的做法是对于互为等价顶点的一组顶点,只保留其中的一个顶点,同时删除与该顶点等价的其他顶点。当需要回答涉及被删除顶点的 k 步可

达查询信息时，可以通过与查询顶点等价的那个保留顶点来判断可达性。在对图建立 2-hop 索引之前，通过先对图进行基于等价顶点的图压缩，能有效减小图的规模，从而降低构建索引的时间和索引大小。

本文采用文献[14]中提出的基于分治思想的求解等价顶点方法。该方法的基本思路是首先将顶点集 V 中的所有顶点视为可能相互等价的，再将集合 V 分成 2 个集合，这 2 个集合中的顶点可能互相等价，但不同集合中的顶点不可能等价。接下来继续对 2 个集合进行类似的分割，直到每个集合都无法再继续分割下去。最后每个集合中存储的顶点都是相互等价的，而集合之间的顶点是不可能等价的，该算法的时间复杂度是 $O(m)$，其中 m 是图中边的个数。

对于 2 个等价顶点 u、v，当查询顶点对为 (u,v) 时，根据顶点 u 和顶点 v 是否位于同一个强连通分量中，会存在 2 种不同的情况，具体如图 1 所示。在图 1(a)中，即顶点 u 和顶点 v 不在同一个 SCC 并且顶点 u 和顶点 v 互为等价顶点，此时顶点 u 必然不可达顶点 v、并且顶点 v 也不可达顶点 u，因此查询结果返回 FALSE。在图 1(b)中，顶点 u 和顶点 v 位于同一个 SCC、并且顶点 u 和顶点 v 互为等价顶点，则 u 和 v 可能是 k 步可达的(取决于 k 值是否大于或等于这 2 个顶点之间的最短路径长度)，此时可能就需要通过遍历或者额外的索引来处理这种特殊情况。为了处理这种不常见但又确实存在的查询，导致每一次回答查询时都要进行额外的判断，这显然是十分繁琐又低效的操作。



(a) 情形一　(b) 情形二
(a) Example 1　(b) Example 2

图 1　等价顶点

Fig. 1　Equivalent vertex

为了解决该问题，本文提出了一种新的策略，对于不在同一个 SCC 中的等价顶点，将对其进行压缩操作。而对于处于同一个 SCC 中的等价顶点，则将其视为不等价的，因此无需进行压缩，等价顶点之间的 k 步可达查询可以直接通过建立的 2-hop 标签回答。该策略能有效地避免在这种特殊条件下的冗长而低效的条件分支判断以及遍历图的操作。

2.3　查询方法

算法 3 给出了在 2 种优化方法下的 PLL 算法的查询方法，设计代码依次展开如下。

算法 3　$query(u,v,k)$

输入：源顶点 u，目标顶点 v，查询距离 k

输出：TRUE/FALSE

```
1   if X_u > X_v or Y_u > Y_v
2     return FALSE
3   u_e ← E_u, v_e ← E_v
4   if u_e = v_e
5     return FALSE
6   u_i ← 0, v_i ← 0
7   while u_i < L_out(u_e).size and v_i < L_in(v_e).size
8     h_u = L_out(u_e)[u_i].hop, h_v = L_in(v_e)[v_i].hop
9     d_u = L_out(u_e)[u_i].dist, d_v = L_in(v_e)[v_i].dist
10    if h_u = h_v do
11      if d_u + d_v <= k do
12        return TRUE
13      else
14        u_i ← u_i + 1, v_i ← v_i + 1
15    else if h_u < h_v do
16        u_i ← u_i + 1
17    else
18        v_i ← v_i + 1
19  return FLASE
```

算法 3 中，第 1 行先判断顶点 u 的拓扑序号 X 是否大于顶点 v 的拓扑序号 X 以及顶点 u 的拓扑序号 Y 是否大于顶点 v 的拓扑序号 Y。如果任意一个条件成立，表明顶点 u 不可达顶点 v，因此第 2 行直接返回 FALSE。第 3 行将顶点 u 和顶点 v 转换成对应的等价顶点u_e和v_e。数组 E 保存了每个顶点与其对应的等价顶点的关系，该数组由 2.2 节中所描述的方法求出。第 4 行如果转换后的等价顶点相等，则查询结果返回 FALSE。否则第 6 ~ 19 行通过 PLL 算法构建的 2-hop 标签来回答查询。

3　实验分析

3.1　实验环境

本文实验中所使用的硬件平台为 Intel(R) Core(TM) i5-4200M @ 2.5 GHz CPU，16 G 双通道内存，操作系统为 Ubuntu 18.04.3 LTS。本次实验算

法采用 C++语言实现,并且查询距离 k 均取值为 10。实验首先比较了 PLL 算法和 PLL 算法在分别使用 2 种优化算法的情况下的索引构造时间、索引大小以及查询时间。随后比较了 PLL 算法与同时使用 2 种优化算法的索引构造时间、索引大小以及查询时间。

3.2 数据集

本次研究中所使用的 10 个数据集见表 1。这 10 个数据集都来自斯坦福大型网络数据集(snap.stanford.edu/data/)。这些图数据集都是有向有环图,表 1 中标注了每个数据集的顶点数 $|V|$ 以及边数 $|E|$。本文将顶点数大于 100 000 的数据集称为大数据集,因此本次实验中共有 5 个大数据集以及 5 个小数据集。本次实验使用的查询集大小为 1 000 000,其中可达顶点对和不可达顶点对的比例为1:1。

表 1 数据集

Tab. 1 Datasets

数据集	$\|V\|$	$\|E\|$
Email-Eu	1 005	24 929
Wiki-Vote	8 298	103 689
Gnutella	10 879	39 994
Epinions	75 888	508 837
Slashdot	82 168	870 161
Sign	131 828	840 799
BerkStan	685 231	7 600 595
Google	916 428	5 105 039
WikiTalk	2 394 385	5 021 410
HepTh	9 912 294	352 768

3.3 性能比较分析

3.3.1 2 种优化技术的比较

PLL 算法和 PLL 算法在分别使用了 2 种优化方法之后的索引大小见表 2。从表 2 中可以看到,互逆拓扑序号所需的存储空间最多,因为需要额外的 2 个整型值来存储拓扑序号(即 PLL+Topo)。其次是 PLL 算法,最优的是使用了基于等价顶点压缩的 PLL 算法(即 PLL+GC),因为仅在部分点上建立2-hop 索引,从而具有最小的索引大小。

PLL 算法和 PLL 算法在分别使用了 2 种优化方法之后的索引构造时间见表 3。从表 3 中可以得到与在索引大小中类似的结论,由于需要额外计算 2 个互逆拓扑序号,因此 PLL+Topo 的时间开销最多,但由于求解拓扑序号是线性时间复杂度,因此差距并不明显。其次,虽然 PLL+GC 方法花费了线性时间来计算等价顶点,但是由于进行了图压缩,因此整个索引构造时间相较于 PLL 算法减少了。

表 2 索引大小

Tab. 2 Index size

数据集	PLL	PLL+Topo	PLL+GC
Email-Eu	0.5	0.5	0.2
Wiki-Vote	3.0	3.0	2.0
Gnutella	19.0	19.0	15.0
Epinions	109.0	109.0	87.0
Slashdot	236.0	236.0	193.0
Sign	184.0	185.0	154.0
BerkStan	837.0	842.0	648.0
Google	1 916.0	1 923.0	1 670.0
WikiTalk	2 483.0	2 501.0	2 100.0
HepTh	192.0	267.0	137.0

表 3 索引构造时间

Tab. 3 Index building time

数据集	PLL	PLL+Topo	PLL+GC
Email-Eu	2 062	2 063	1 579
Wiki-Vote	3 122	3 146	2 672
Gnutella	14 173	14 205	10 648
Epinions	70 627	70 829	58 267
Slashdot	236 006	236 274	17 264
Sign	141 879	142 372	10 357
BerkStan	1 964 210	1 965 677	1 539 751
Google	1 303 740	1 304 557	946 271
WikiTalk	1 815 360	1 820 991	1 364 175
HepTh	45 638	47 822	39 462

PLL 算法和 PLL 算法在分别使用了 2 种优化方法之后的查询时间见表 4。其中,查询距离 k 取 10。从表 4 中可以看出在应用了互逆拓扑序号后的查询效率有了明显的提升,在一些数据集上有 2~3 倍左右的提升。PLL 算法在回答不可达查询时的时间消耗要大于回答可达的查询顶点对所花费的时间,而互逆拓扑序号能在 $O(1)$ 的时间内回答绝大部分的不可达查询,因此提高了查询的效率。对于 PLL+GC 方法来说,由于进行了图压缩,因此建立的索引大小更小,从而使得查询时需要遍历的 hop 顶点数更少,因此也在一定程度上提高了查询效率。

3.3.2 PLL 算法与 PLL-O 方法的比较

PLL 算法和 PLL 算法同时应用了 2 种优化方法(即 PLL-O)之后的索引大小见表 5。从表 5 中可以看出在绝大多数的数据集上,PLL-O 方法的索引大小要优于 PLL 算法。只有在极个别图上(如 HepTh),图压缩方法对索引大小带来的收益没能抵消 2 个拓扑号带来的空间开销,此时 PLL-O 方法的索引规模会略大于 PLL 算法。

表 4　查询时间

Tab. 4　Query time

数据集	PLL	PLL+Topo	PLL+GC
Email-Eu	300	141	264
Wiki-Vote	381	134	367
Gnutella	1 503	759	1 347
Epinions	1 017	637	918
Slashdot	1 641	691	1 439
Sign	1 077	384	854
BerkStan	4 772	2 807	4 063
Google	3 415	1 897	3 279
WikiTalk	1 269	669	1 054
HepTh	1 131	570	983

表 5　索引大小

Tab. 5　Index size

数据集	PLL	PLL+Topo+GC
Email-Eu	0.5	0.2
Wiki-Vote	3.0	2.0
Gnutella	19.0	15.0
Epinions	109.0	87.0
Slashdot	236.0	193.0
Sign	184.0	155.0
BerkStan	837.0	653.0
Google	1 916.0	1 677.0
WikiTalk	2 483.0	2 118.0
HepTh	192.0	212.0

PLL 算法和 PLL 算法同时应用了 2 种优化方法之后的索引构造时间见表 6。由于综合了图压缩方法的优点，PLL-O 算法在整体上的性能都优于 PLL 算法。

表 6　索引构造时间

Tab. 6　Index building time

数据集	PLL	PLL-O
Email-Eu	2 062	1 603
Wiki-Vote	3 122	2 803
Gnutella	14 173	10 539
Epinions	70 627	58 369
Slashdot	236 006	17 892
Sign	141 879	11 034
BerkStan	1 964 210	1 634 801
Google	1 303 740	943 119
WikiTalk	1 815 360	1 436 792
HepTh	45 638	40 346

PLL 算法和 PLL 算法同时应用了 2 种优化方法之后的查询时间见表 7，其中查询距离 k 取 10。从表 7 中可以看出 PLL-O 方法相比于只应用了图压缩的方法的查询效率要高，然而相对于仅应用互逆拓扑序号的方法来说性能差距不大。但综合考虑索引大小以及索引构造时间后，PLL-O 方法较 PLL 算法以及仅使用一种优化方法的 PLL 方法仍具有较大的优势。

表 7　查询时间

Tab. 7　Query time

数据集	PLL	PLL+Topo+GC
Email-Eu	300	168
Wiki-Vote	381	131
Gnutella	1 503	762
Epinions	1 017	619
Slashdot	1 641	664
Sign	1 077	403
BerkStan	4 772	2 347
Google	3 415	1 704
WikiTalk	1 269	683
HepTh	1 131	439

4　结束语

本文针对已有 k 步可达查询算法存在的不可达查询效率低，索引构造时间长，索引规模大等问题提出了 2 种优化的方法，分别是基于等价顶点的图压缩方法以及基于互逆拓扑序号的不可达查询优化方法。实验结果表明，在同时应用了这 2 种算法之后，PLL 算法的查询效率、索引大小以及索引构造时间都有了明显的提升。

参考文献

[1] Van HELDEN J, NAIM A, MANCUSO R, et al. Representing and analysing molecular and cellular function using the computer [J]. Biological chemistry, 2000, 381(9-10): 921-935.

[2] 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. Anchorage, AK, USA: IEEE, 2003: 102-112.

[3] KUMAR R, TUZHILIN A, FALOUTSOS C, et al. Social networks: Looking ahead [C]//Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Las Vegas, USA: ACM, 2008: 1060.

[4] CHEN Yangjun, CHEN Yibin. An efficient algorithm for answering graph reachability queries [C]//2008 IEEE 24th International Conference on Data Engineering. Cancun, Mexico: IEEE, 2008: 893-902.

[5] YU J X, CHENG Jiefeng. Graph reachability queries: A survey [M]//Managing and Mining Graph Data. Boston, MA: Springer, 2010: 181-215.

[6] CHENG J, HUANG Silu, WU Huanhuan, et al. TF-Label: A topological-folding labeling scheme for reachability querying in a large graph [C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 193-204.

（下转第 19 页）