软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(Suppl.(2)):213−224

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

RIAIL:大规模图上的可达性查询索引方法*

解 宁, 申德荣, 冯 朔, 寇 月, 聂铁铮, 于 戈

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

通讯作者: 解宁, E-mail: andy.xning@gmail.com, http://www.neu.edu.cn

摘 要: 图被广泛用来建模在社交网络、语义网、计算生物学和软件分析中的应用.可达性查询是图数据上的一种基础查询.当前,针对图上的可达性查询已经提出了一些索引算法,但是它们不能灵活地扩展到大的图数据.因此,提出了一种索引方法 RIAIL(reachability index augmented by interval labeling).RIAIL 将结点的标记信息表示成四元组.前两个元素是区间标记,编码生成树的可达性信息,后两个元素编码非树边的可达性信息.RIAIL 查询时只需索引且索引创建代价小.最后,通过大量真实和人工生成数据集上的实验说明,RIAIL 能够高效地处理可达性查询,并且可以简单地扩展到大的图数据.

关键词: 图;可达性查询;索引算法;区间标记;生成树

中文引用格式: 解宁,申德荣,冯朔,寇月,聂铁铮,于戈.RIAIL:大规模图上的可达性查询索引方法.软件学报,2014,25(Suppl.(2)):213−224. http://www.jos.org.cn/1000-9825/14039.htm

英文引用格式: Xie N, Shen DR, Feng S, Kou Y, Nie TZ, Yu G. RIAIL: An index method for reachability query in large scale graphs. Ruan Jian Xue Bao/Journal of Software, 2014,25(Suppl.(2)):213−224 (in Chinese). http://www.jos.org.cn/1000-9825/14039.htm

RIAIL: An Index Method for Reachability Query in Large Scale Graphs

XIE Ning, SHEN De-Rong, FENG Shuo, KOU Yue, NIE Tie-Zheng, YU Ge

(College of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

Corresponding author: XIE Ning, E-mail: andy.xning@gmail.com, http://www.neu.edu.cn

Abstract: Graph has been widely used to model the applications in social network, semantic web, computational biology and software analysis. Reachability query is one kind of basic queries in graph data. Currently, in allusion to graph, several index algorithms have been proposed to answer reachability query, however, they can not scale to large graph flexibly. To address the issue, a new index method called RIAIL (reachability index augmented by interval labeling) is developed in this paper. RIAIL labels each node with four-tuple. The first two elements are interval labels that encode reachability information of the spanning tree, and the last two elements encode reachability information of non-tree edges. RIAIL only needs to index when querying and the cost of index construction is little. Finally, a wide range of experiments on real and synthetic datasets are conducted to demonstrate that RIAIL can efficiently handle reachability query and easily scale to large graph.

Key words: graph; reachability query; index algorithm; interval labeling; spanning tree

近年来,图模型因其适用性而被广泛使用在计算生物学、计算机网络和本体管理等领域的多种应用中,图数据的有效管理成为一个热门的研究领域.不同的应用对应不同的图数据模型,需要不同的处理方法,从而导致了图数据管理上的不同算法.图数据上有几类典型查询:可达性查询、图匹配和关键字查询[1].

处理图上的可达性查询有两种比较直接的方法,第 1 种方法是对图进行深度或宽度优先遍历[2].这种方法

* 基金项目: 国家自然科学基金(61033007, 61472070); 国家重点基础研究发展计划(973)(2012CB316201); 教育部-英特尔信息技术专项科研基金(MOE-INTEL-2012-06)

收稿时间: 2014-05-07; 定稿时间: 2014-08-19

不使用索引,只是在原始图上遍历,最坏情况下查询时间复杂度是 $O(m+n)$[2],其中 n 是图的结点数,m 是图的边数.第 2 种方法是提前计算图 G 的传递闭包并将传递闭包作为索引,通过查询(u,v)是否在传递闭包里而在 $O(1)$时间内返回查询结果.这种方法在查询时不依赖原始图,但图传递闭包的空间复杂度是 $O(n^2)$.上述两种方法都因其局限性而不能扩展到大的图数据.现实应用中的图数据大都是有百万结点甚至千万结点的大图,为了高效地解决这个问题,人们已经提出了很多不同的算法[3−14],这些算法的基本思想是在图数据上建立索引,通过索引来判断结点之间的可达性.索引算法的不同之处在于使用不同的方法压缩图的传递闭包,并在索引创建时间、索引空间和查询性能之间进行权衡.

与预先计算图的传递闭包并将其作为索引和对图进行深度或宽度优先遍历相比,已有的可达性查询索引算法能够扩展到比较大的图数据,但这些算法在处理大图时还存在局限性.GRAIL[11,13]将索引和原始图同时保存在内存中,才能有很好的查询性能.GRIPP[9]在查询时只依赖索引,当不能直接判断两个结点可达时,要通过 hop 结点进行递归查询.但 GRIPP 在查找 hop 结点时要查询索引表上的联合索引,使得查询性能降低.文献[13]中的对比实验说明 GRIPP 查询性能不高.

本文提出了一种新的索引方法,并在此基础上给出了结点可达性的判定方法.该索引算法能够很好地解决上述算法存在的问题,并且能够简单地扩展到大的图数据.本文的主要贡献如下:

1) 提出了一种新的索引方法 RIAIL.查询时,只需索引就能判断结点之间的可达性,并且创建索引的时间复杂度是 $O(m+n)$.

2) 在该索引方法的基础上,给出了结点可达性的判定方法,并且查询的时间复杂度是 $O(m-n)$.

3) 通过大量在真实和人工生成数据集上的实验说明了所提出方法的高效性和扩展性.

1 相关工作

已有处理可达性查询的索引算法主要分为两类,一类是集合覆盖算法,其中主要包括 hop 标记算法.hop 标记算法的基本思想是对图中每个结点 u 生成两个集合:u_{in} 表示到达结点 u 的结点集合,u_{out} 表示结点 u 能到达的结点集合.查询结点 u 到结点 v 是否可达,只需判断 u_{out} 和 v_{in} 是否包含相同的结点.2-hop[4]是最早提出的 hop 标记算法,其使用集合覆盖的方法生成结点的 hop 标记,但该方法的计算复杂度太高.为此,进一步提出了一些改进和扩展 2-hop 标记的算法[6−8].

另一类是区间标记算法,Optimal Tree-cover[3]是最早利用区间标记处理有向图中可达性查询的方法,该算法在文献[15]的基础上,将树上的区间编码拓展到有向图.为了保持可达性信息的完整性,Optimal Tree-cover 在遍历图后,按逆拓扑序将每条边终点的标记传递给起始点,这样在查询阶段通过结点间标记的包含关系就可以判断两个结点间的可达性.文献[3]提出了一种优化的树覆盖方法以减少标记数量,但这种算法的标记数量还是很大,不能扩展到大图[13].Dual Labeling[5]是针对稀疏图提出的索引算法.该算法先对图进行遍历,生成区间标记,然后计算并保存非树边的传递闭包.当图是稀疏图或者树形图时,该方法的性能很好,但当图变稠密之后,生成和存储索引的代价变大,性能降低.

与本文密切相关的区间标记算法主要有 GRAIL,Ferrari[14]和 GRIPP.

GRAIL 在有向无环图上进行区间标记,是一种近似标记[14].这种标记方法能够直接判断结点之间不可达,但结点之间可达时,不能通过标记直接判断,要在原始图上进行遍历.为了减少不能直接判断的查询数量,文献[11]中提出多趟随机标记方法,针对每个结点生成多个标记,当多个标记都不能直接判断时再进行遍历.文献[13]进一步扩展了该方法,并且提出了多种优化措施.Ferrari 扩展了 GRAIL 近似标记的思想,并且可以根据实际的存储空间限制结点的标记数量,但 Ferrari 和 GRAIL 均需索引和原始图同时保存在内存中才能保证性能.文献[12]提出 Yes-Label 来减少 GRAIL 在原始图上的查询次数,并且提出了一种当索引和原始图不能同时保存在内存中时,优化磁盘 I/O 的数据结构.

GRIPP 在不能通过区间标记直接判断可达性时,查询联合索引找到 hop 结点,并针对结点 u 的每个 hop 结点 w,递归生成新的结点 w 到结点 v 的查询.查询联合索引的时间复杂度不是常量,导致 GRIPP 实际的查询时间

复杂度高于 $O(m-n)$.

为了保证算法的可扩展性,需用有限的空间将所有的可达性信息保存在索引中,这样,在查询阶段就不再依赖原始图.本文提出的 RIAIL 算法将所有的可达性信息保存在创建的索引中,并且创建索引的时间复杂度是 $O(m+n)$,查询的时间复杂度是 $O(m-n)$.通过大量实验说明,在处理大部分稠密图时,RIAIL 的索引大小小于 GRIPP.

2 基本概念

区间标记:主要有两种区间标记方式.一种是[min_*post*,*post*],其中,*post* 是该结点在图的后序遍历序列中的值,min_*post* 是生成树中以该结点为根的所有后代结点中,*post* 值最小的结点的 *post* 值.另一种是[*pre*,*post*],其中,*pre* 和 *post* 分别是对图进行深度优先遍历第 1 次经过该结点时的值和遍历完该结点的所有后代结点后再次经过该结点时的值.

树边和非树边:对图进行深度优先遍历后,图被分成生成树和一些边.既在图中也在生成树中的边,记作树边;只存在于图中的边,记作非树边.

hop 结点:一条非树边的终点记作 hop 结点.

特殊结点:非树边的起始结点记作特殊结点.

结点 *u* 的 hop 结点:生成树中以结点 *u* 为根的后代结点中,如果结点 *w* 是特殊结点,则结点 *w* 对应的非树边终点是结点 *u* 的 hop 结点;否则,不是.以图 1 中的有向无环图 G_{dag} 为例,对其进行深度优先遍历,实线表示树边,虚线表示非树边.生成树中,结点 *B* 的所有后代结点中,结点 *F*,*I*,*J* 和 *H* 是特殊结点,按照定义,这些结点对应的非树边终点 *G* 和 *K* 是结点 *B* 的 hop 结点.

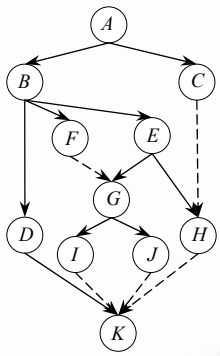


Fig.1 Directed acyclic graph G_{dag}

图 1 有向无环图 G_{dag}

3 基础索引 B-RIAIL

已有算法在研究可达性查询时,都是针对有向无环图,因此本文也面向有向无环图.对于有向图,可以通过将有向图中的强连通分量压缩成一个结点将有向图转换成有向无环图.后文中如无特殊声明,图均指有向无环图.

3.1 构建B-RIAIL

GRIPP 只通过索引查询图上任意两个结点之间的可达性,但是由前所述,GRIPP 查询性能不高.为了减少查找 hop 结点的复杂度,一种直接的方法是保证结点的索引包含该结点所有的 hop 结点.本文将该索引结构称为基础索引 B-RIAIL.B-RIAIL 索引结构表示为三元组 $L_{\text{node}}=\langle pre,post,hops\rangle$,其中,区间标记选择[*pre*,*post*]标记方式,*hops* 是该结点的所有 hop 结点的集合.

Table 1 Index of B-RIAIL

表 1 B-RIAIL 索引

Node	*pre*	*post*	*hops*
A	0	21	[*K*,*G*,*H*]
B	1	18	[*K*,*G*]
C	19	20	[*H*]
D	2	5	[ ]
E	6	15	[*K*]
F	16	17	[*G*]
G	7	12	[*K*]
H	13	14	[*K*]
I	8	9	[*K*]
J	10	11	[*K*]
K	3	4	[ ]

随机遍历图,每个结点只遍历一次.当一个结点通过非树边遍历到结点 *w* 时,将结点 *w* 加入该结点的 *hops*.如果一个结点的 *hops* 不为空,则该结点的 *hops* 将沿着生成树中的树边反向传递到祖先结点.B-RIAIL 索引可以通过一次深度优先遍历生成,因此生成索引的时间复杂度是 $O(m+n)$.

按照上述方法对图 1 中的有向无环图 G_{dag} 进行标记,生成的基础索引 B-RIAIL 见表 1.

B-RIAIL 查询时,当不能通过区间标记直接判断结点之间的可达性时,可直接得到结点的 hop 结点,然后对每个 hop 结点 *w* 生成一个新的查询,最坏情况下生成递归查询的次数与遍历图后非树边的数量相

同,因此查询的时间复杂度是 $O(m-n)$.但是这种方法存在以下不足:任一结点的 *hops* 是通过合并得到的,可能存在重复的 hop 结点,需要去重,这会增加建立索引的时间.以图 1 中结点 *G* 为例,结点 *I* 和结点 *J* 的 *hops* 都为 *K*,生成结点 *G* 的标记时需要去重,最后只保留一个 hop 结点 *K*.

4 RIAIL 索引

为了解决 B-RIAIL 索引的问题,提出了 RIAIL 索引方法.

RIAIL 索引结构为四元组 $L_{node}=\langle pre,post,hops,directs\rangle$.其中,*pre* 和 *post* 与 B-RIAIL 中的相同,如果一个结点是特殊结点,则 *hops* 是一个包含该结点的 hop 结点的数组,否则为空.对于任一结点,*directs* 是生成树中以该结点为根的所有后代结点中特殊结点的数组,即从当前结点到 *directs* 中特殊结点之间存在只包含树边的路径.

随机遍历图,并且每个结点也只遍历一次.当一个结点通过树边遍历到结点 *w* 时,如果结点 *w* 是特殊结点,则将结点 *w* 加入该结点的 *directs*;如果结点 *w* 不是特殊结点,则将结点 *w* 的 *directs* 加入该结点的 *directs*.当一个结点通过非树边遍历到结点 *w* 时,将结点 *w* 加入该结点的 hops.

以图 1 中的图 G_{dag} 为例,按照 RIAIL 索引方法进行遍历,生成的标记信息保存在表 2 中.对于结点 *B*,因为结点 *B* 不是特殊结点,因此 *hops* 为空.在生成树中以结点 *B* 为根的所有后代结点中,结点 *F*,*H*,*I*,*J* 为特殊结点并包含在结点 *B* 的 *directs* 中.虽然结点 *B* 到结点 *D*,*K*,*E*,*G* 的路径也只包含树边,但它们不是特殊结点,因此没有包含在结点 *B* 的 *directs* 中.对于结点 *F*,因为结点 *F* 是特殊结点,因此 *hops* 不为空并包含结点 *F* 的所有 hop 结点,即结点 *G*.因为结点 *F* 的出边中只有一条非树边,因此结点 *F* 的 *directs* 为空.

下面给出一个关于 *directs* 的定理.

定理 1. 任一结点的 *directs* 不包含重复结点.

证明:假设结点 *u* 的 *directs* 包含重复结点 *q*,根据 *directs* 的定义,生成树中从结点 *u* 到结点 *q* 存在两条不同的路径.因此在生成树中,存在结点 *w*(结点 *w* 或者是结点 *q* 或者是结点 *q* 在生成树中的祖先结点并且结点 *w* 的 *directs* 中包含结点 *q*),满足从结点 *u* 到结点 *w* 存在两条不同的路径 *P*1 和 *P*2,并且从结点 *w* 到结点 *q* 存在一条路径 *P*3,如图 2(a)所示.即结点 *u* 从两条不同的生成树路径 *P*1 和 *P*2 通过结点 *w* 将结点 *q* 加入其 *directs*,即结点 *w* 在生成树中有两条不同的入边,这显然与生成树中结点只有一条入边矛盾.因此,任一结点的 *directs* 不包含重复结点.

Table 2 Index of RIAIL

表 2 RIAIL 索引

Node	*pre*	*post*	*hops*	*directs*
A	0	21	[ ]	[*I*,*J*,*H*,*F*,*C*]
B	1	18	[ ]	[*I*,*J*,*H*,*F*]
C	19	20	[*H*]	[ ]
D	2	5	[ ]	[ ]
E	6	15	[ ]	[*I*,*J*,*H*]
F	16	17	[*G*]	[ ]
G	7	12	[ ]	[*I*,*J*]
H	13	14	[*K*]	[ ]
I	8	9	[*K*]	[ ]
J	10	11	[*K*]	[ ]
K	3	4	[ ]	[ ]

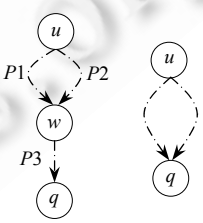


Fig.2 Examples

图 2 示例

依据该定理可知,生成标记时不用判断 *directs* 中是否包含重复结点,减少了标记的时间.RIAIL 索引方法在标记过程中,可能会出现一种特殊情况,如图 2(b)所示,标记结束后,一对结点之间有超过一条非树边,这时结点 *u* 的 *hops* 包含重复结点 *q*.出现这种情况是因为图中两个结点之间存在多于一条边的情况,但是针对本文所考虑的可达性查询,可以将图中两个结点之间的多条边合并成一条.经过该合并操作,任一结点的 *hops* 不包含重复结点.

RIAIL 同样只需对图进行一次随机的深度优先遍历即可获得结点的标记,生成索引的时间复杂度也是 $O(m+n)$,但因为不用考虑生成索引过程中的结点去重,因此实际 RIAIL 的索引创建时间优于 B-RIAIL.由于不能

准确计算结点标记中 *directs* 包含的结点数量,因此对于 RIAIL 索引的空间复杂度,不能给出一个确切的上界.但是,大量的实验说明 RIAIL 的索引大小在大部分数据集上小于 B-RIAIL.

4.1 索引算法

算法 1 描述了生成标记的过程.前两行是初始化,将所有结点的访问状态初始化为未访问.一个有向无环图可能有多个根结点,第 3 行~第 5 行对有向无环图进行遍历并标记结点.第 11 行~第 14 行生成结点 j 的 *directs*,如果结点 k 的 *hops* 不为空,即结点 k 是特殊结点,则只需将结点 k 加入 *cur_directs*;否则将结点 k 的 *directs* 加入到 *cur_directs*.最后,第 17 行将 *cur_directs* 赋值给结点 j 的 *directs*.算法并没有为 *directs* 检查重复结点,因为根据定理 1,任一结点的 *directs* 都不会包含重复结点.

算法 1. 生成结点标记 L_G.

输入:有向无环图 G_{dag}.

输出:有向无环图 G_{dag} 中 n 个结点的标记 L_G.

LabelGraph(G_{dag}):

```
1:  foreach node i in G_dag do
2:      visited[i]←false;
3:  foreach root node j in G_dag do
4:      Traverse(G_dag,j,post,directs);
5:      label j with pre, post and directs;
Traverse(G_dag,j,post,directs):
6:  visited[j]←true;
7:  foreach node k in out_nodes[j] do
8:      if visited[k]=false then
9:          Traverse(G_dag,k,post,directs);
10:         label k with pre, post and directs;
11:         if k.hops≠[] then
12:             cur_directs←cur_directs∪k;
13:         else
14:             cur_directs←cur_directs∪k.directs;
15:     else
16:         j.hops←j.hops∪k;
17:  directs←cur_directs;
```

5 查询处理

查询阶段首先通过结点的区间标记判断可达性,当不能直接判断时,根据结点的 *hops* 和 *directs* 递归生成新的查询.最坏情况下递归查询的次数与图遍历后非树边的数量相同,因此查询的时间复杂度是 $O(m-n)$.

5.1 查询处理过程

查询处理分为两步.假设查询有向无环图 G_{dag} 中结点 u 到结点 v 是否可达,则查询处理过程如下:

I. 根据结点 u 和结点 v 的区间包含关系判断可达性.如果 $u_{pre}<v_{post}\leqslant u_{post}$,则结点 u 到结点 v 可达;否则不能直接判断,进行步骤II.

II. 根据结点 u 的类型生成新的查询.

1) 结点 u 是特殊结点,对于 *hops*,生成 *hops* 结点到结点 v 的新查询;对于 *directs*,生成 *directs* 结点到结点 v 的新查询,并直接进行步骤II;

2) 结点 u 不是特殊结点并且 *directs* 不为空,则生成 *directs* 结点到结点 v 的新查询,并直接进行步骤II;

3) 结点 u 不是特殊结点并且 *directs* 为空,则直接返回不可达.

以图 1 中结点 B 到结点 C 的可达性查询为例,因为不满足 $B_{pre}<C_{post}\leqslant B_{post}$,所以不能直接判断可达性,进行步骤II.结点 B 不是特殊结点且 *directs* 不为空,生成结点 I、结点 J、结点 H 和结点 F 到结点 C 的新查询.结点 I 是特殊结点且 *directs* 为空,生成结点 K 到结点 C 的查询,因为不满足 $K_{pre}<C_{post}\leqslant K_{post}$,结点 K 不是特殊结点且 *directs* 为空,因此结点 K 到结点 C 不可达,即结点 I 到结点 C 不可达.依次处理结点 J、结点 H 到结点 C 的查询,这两个查询都会生成结点 K 到结点 C 的查询,根据已有查询结果可知,结点 J、结点 H 到结点 C 不可达.结点 F 是特殊结点且 *directs* 为空,生成结点 G 到结点 C 的查询,因为结点 G 不满足 $G_{pre}<C_{post}\leqslant G_{post}$,所以不能直接判断可达性,进行步骤II.结点 G 不是特殊结点且 *directs* 不为空,因此生成结点 I 和结点 J 到结点 C 的新查询,根据已有查询结果可知,结点 G 到结点 C 不可达,即结点 F 到结点 C 不可达.最后得出结点 B 到结点 C 不可达.

5.2 查询优化

为了提高查询性能,本文采用如下 4 种优化策略:

优化 1:通过标识重复查询实现查询优化.在查询进行到第II步时会生成新查询,这些新查询可能包含重复查询,需要去重.在第 5.1 节的查询例子中就包含两个重复的结点 K 到结点 C 的查询.可以在查询时保存已检查的结点集合,并且在生成新的查询时,先检查是否已经查询过该结点,如果没有,则生成新的查询;否则,说明该结点已经被检查过,直接考虑下一个结点.

优化 2:避免不必要的区间包含判断.第 5.1 节查询结点 I 到结点 C 可达性时,结点 I 的区间标记包含在结点 B 中,由此可知生成树中结点 I 到结点 C 不可达.否则,如果生成树中结点 I 到结点 C 可达,则生成树中结点 B 到结点 C 可达,与已知生成树中结点 B 到结点 C 不可达矛盾.因此,可以不必再判断结点 C 的区间是否包含在结点 I 中,直接对结点 I 的 *hops* 和 *directs* 结点进行扩展.

优化 3:应用近似标记思想.在[*pre*,*post*]区间标记的基础上,为了减少索引空间大小,提出了在有向无环图上的[min_*pre*,*post*]区间标记.min_*pre* 值是以该结点为根的子图中 *pre* 值最小的结点的 *pre* 值,*post* 值与[*pre*,*post*]标记中的 *post* 值相同,因此每个结点实际只需要存储一个 *post* 值,节省了索引空间.以图 1 中的有向无环图 G_{dag} 为例,生成的[min_*pre*,*post*]标记见表 3.根据[min_*pre*,*post*]标记的生成规则,如果结点 v 的标记不包含在结点 u 的标记中,则结点 u 到结点 v 不可达;否则不能直接判断结点 u 到结点 v 的可达性,需要进行进一步的判断.以图 1 中的结点 B 到结点 C 的可达性查询为例,根据结点 B 和结点 C 的[min_*pre*,*post*]标记,[1,18]⊅[3,20],则可直接判断结点 B 到结点 C 不可达.查询结点 C 到结点 E 是否可达,根据结点 C 和结点 E 的[min_*pre*,*post*]标记,[3,20]⊃[3,15],但是实际在图 1 中结点 C 到结点 E 不可达.因此,[min_*pre*,*post*]标记可快速判断结点之间的不可达.

Table 3 Label of [min_*pre*,*post*]

表 3 [min_*pre*,*post*]标记

Node	min_*pre*	*post*
A	0	21
B	1	18
C	3	20
D	2	5
E	3	15
F	3	17
G	3	12
H	3	14
I	3	9
J	3	11
K	3	4

优化 4:采用文献[14]中 Level Filter 方法优化.图中每个结点都赋一个值,用于表示该结点在图中的层级.如果结点 u 的层级大于结点 v 的层级,则结点 u 到结点 v 肯定不可达,依此来过滤查询.

通过采用上述 4 种优化策略过滤查询和快速回答不可达查询,有效提高了查询性能.

5.3 查询算法

为了方便展示,算法 2 描述只加了优化 1 和优化 2 的查询算法.*checked* 保存已经检查过的结点,第 1 行初始化 *checked*.第 2 行~第 3 行是查询的第I步,根据两个结点的区间包含关系判断可达性.第 9 行~第 19 行是查询的第II步,其中,第 9 行~第 18 行对应结点 u 是特殊结点的情况,第 14 行~第 18 行对应结点 u 不是特殊结点且 *directs* 不为空的情况,第 19 行对应结点 u 不是特殊结点且 *directs* 为空的情况.第 23 行~第 27 行意思是,如果判断结点 p 到结点 v 不可达,就将结点 p 作为已被检查过的结点加入 *checked*.分别在第 11 行和第 16 行利用 *checked* 检查

是否生成了重复的查询.第 17 行利用优化 2 对于 *directs* 中的结点调用 Check 方法而不是 Reachability 方法.

算法 2. 查询算法.

输入:有向无环图 G_{dag} 的标记 L_G,结点 u 和 v.

输出:结点 u 和 v 之间的可达性.可达,返回 true;不可达,返回 false.

```
Query(L_G,u,v):
1:   checked.clear();
2:   if v⊆u then
3:       return true;
4:   else
5:       if Check(L_G,u,v) then
6:           return true;
7:       else
8:           return false;
Check(L_G,u,v):
9:   if special[u]=true then
10:      foreach node p in u.hops do
11:          if checked[p]=false then
12:              if Reachability(L_G,p,v) then
13:                  return true;
14:  if u.directs≠[] then
15:      foreach node q in u.directs do
16:          if checked[q]=false then
17:              if Check(L_G,q,v) then
18:                  return true;
19:  return false;
Reachability(L_G,p,v):
20:  if v⊆p then
21:      return true;
22:  else
23:      if Check(L_G,p,v) then
24:          return true;
25:      else
26:          checked←checked∪p;
27:          return false;
```

6 实 验

为了保证实验的可信性,实验中采用了在已有可达性查询算法中使用较多的数据集(http://code.google.com/p/grail/).数据集分为两大类:真实数据集和人工生成的数据集.数据集根据图的稀疏、稠密和图的大小分为 4 个不同的类别.表 4~表 7 分别对应的是小的稀疏图、小的稠密图、大图(上半部两个数据集是大的稀疏图,下半部 3 个数据集是大的稠密图)和人工生成的图数据.

B-RIAIL 和 RIAIL 算法都用 C++语言实现.为了降低索引创建时间,B-RIAIL 用哈希集合存储每个结点的 hops,这导致 B-RIAIL 在遍历 hops 结点生成新查询时具有随机性,也导致了 B-RIAIL 的查询性能在一些数据集

上比 RIAIL 差.同时,由于哈希结构的额外存储开销,B-RIAIL 不能扩展到 10m5x 数据集.

Table 4 Small sparse graphs

表 4 小的稀疏图

Dataset	Nodes	Edges	Average-Degree
agrocyc	12 684	13 657	1.06
amaze	3 710	3 947	1.06
anthra	12 499	13 327	1.07
ecoo	12 620	13 575	1.08
human	38 811	39 816	1.03
kegg	3 617	4 395	1.22
mtbrv	9 602	10 438	1.09
nasa	5 605	6 538	1.17
vchocyc	9 491	10 345	1.09
xmark	6 080	7 051	1.16

Table 5 Small dense graphs

表 5 小的稠密图

Dataset	Nodes	Edges	Average-Degree
arxiv	6 000	66 707	11.12
citeseer	10 720	44 258	4.13
go	6 793	13 361	1.97
pubmed	9 000	40 028	4.45
yago	6 642	42 392	6.38

Table 6 Large graphs

表 6 大图

Dataset	Nodes	Edges	Average-Degree
citeseer	693 947	312 282	0.92
uniprot22m	1 595 444	1 595 442	1.00
citeseerx	6 540 401	15 011 259	2.30
cit-Patents	3 774 768	16 518 947	4.38
go-uniprot	6 967 956	34 770 235	4.99

Table 7 Synthetic graphs

表 7 人工生成的图

Dataset	Nodes	Edges	Average-Degree
10m2x	10M	20M	2
10m5x	10M	50M	5

实验中将本文的方法与 GRAIL 和 Ferrari(http://code.google.com/p/ferrari-index/)在索引创建时间、索引大小和查询性能这 3 个方面进行了比较,其中 GRAIL 和 Ferrari 算法的相关参数均按照原文的推荐设置.索引大小以编码结点标记值的数值类型的值的个数表示,本实验中标记值使用的是 C++编程语言中的整数类型,因此索引大小以索引中整数的个数表示.查询性能是执行 100 000 个查询的总运行时间.实验中每个数据集使用的 100 000 个查询与文献[13]中的查询相同,人工生成的数据集上的查询也采用文献[13]中相应规模数据集对应的查询.实验中使用的操作系统是 64 位的 Ubuntu 12.04 LTS,内核版本号是 3.2.0,4GB 内存,Intel Quad-Core i7-2600 CPU,3.40GHZ.文献[9]中的 GRIPP 作为数据库中的存储过程实现,而本实验中参与比较的算法都采用 C++语言实现,为了保证实验结果的可信性,实验中没有与 GRIPP 算法比较.然而,文献[13]的作者实现了 GRIPP 算法,并将 GRAIL 和 GRIPP 性能做了对比.本实验中使用的数据集与文献[13]中的部分数据集相同,为了与文献[13]中 GRIPP 索引大小实验结果进行对比,针对两个结点之间存在多条边的数据集,RIAIL 没有将两个结点之间的多条边合并成一条.

(1) 小的稀疏图数据

实验结果如图 3~图 5 所示.对于所有数据集,B-RIAIL 的索引创建时间大约是 RIAIL 的 2 倍.RIAIL 的索引创建时间是最优的,因为它只对图进行一次深度优先遍历,而 GRAIL 要进行 2 趟随机标记,Ferrari 要先用一种近

似最优的树覆盖算法为每个结点生成准确的区间标记,然后再按逆拓扑序生成最终的混合标记.RIAIL 的索引大小也是最优的.查询性能最好的是 Ferrari-G,这主要是因为 Ferrari-G 通过混合标记过滤了很多查询,减少了在原始图上进行遍历的次数.RIAIL 的查询性能在所有数据集上优于 GRAIL,这主要是因为 GRAIL 在不能直接判断结点之间的可达性时需在原始图上进行遍历,而 RIAIL 这时只需对非树边进行遍历.

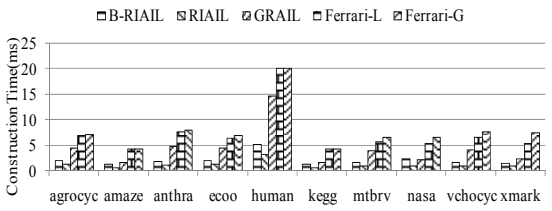


Fig.3 Index construction time of small sparse graphs

图 3 小的稀疏图索引创建时间

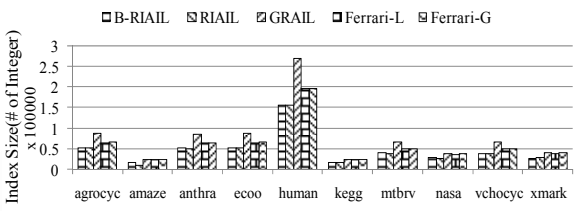


Fig.4 Index size of small sparse graphs

图 4 小的稀疏图索引大小

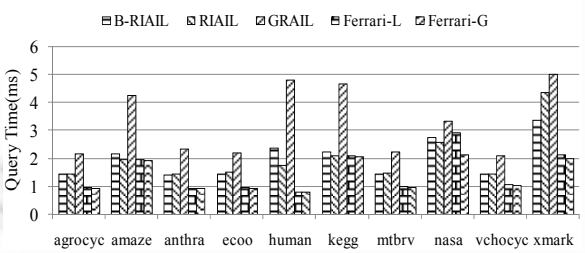


Fig.5 Query time of small sparse graphs

图 5 小的稀疏图查询时间

(2) 小的稠密图数据

实验结果如图 6~图 8 所示.对于所有数据集,B-RIAIL 的索引创建时间是 RIAIL 的 4~9 倍.根据文献[13]中的实验,对于大部分数据集,RIAIL 的索引大小小于 GRIPP,这主要是因为稠密图在遍历后生成很多非树边,GRIPP 对于每条非树边的终点都会生成一个新的区间标记.RIAIL 的索引创建时间是最优的,原因与小的稀疏图一样.对于 arxiv 和 yago,RIAIL 的索引空间比较大,主要是因为这两个数据集的平均度数较大,遍历后存在大量的非树边,有大量信息存储在结点的 hops 和 directs 中.但 RIAIL 在查询时只需索引,因此查询时 RIAIL 对内存的占用比 GRAIL 和 Ferrari 小.对于其他数据集,RIAIL 的索引大小基本是最优的.在大部分数据集上,RIAIL 的查询性能优于 GRAIL,原因与小的稀疏图一样.Ferrari-G 的查询性能优于 RIAIL,但 RIAIL 可以采用类似于 GRAIL 中的多趟标记方法来提高查询性能,并且因为 RIAIL 在查询时只需索引且对于小的稠密图,索引比原始图小很多,因此实际上 RIAIL 可以采用更多次数的标记来过滤查询.

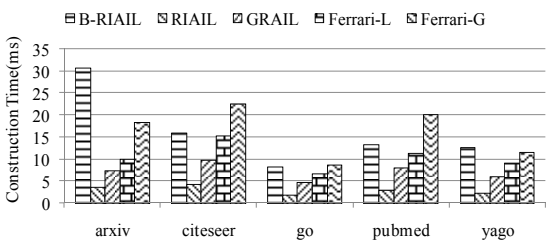


Fig.6 Index construction time of small dense graphs

图 6 小的稠密图索引创建时间

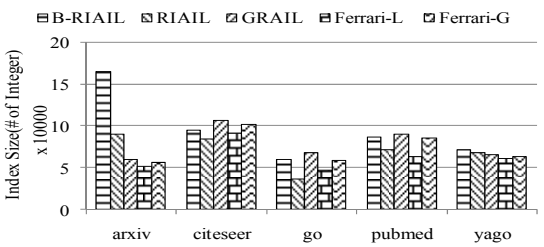


Fig.7 Index size of small dense graphs

图 7 小的稠密图索引大小

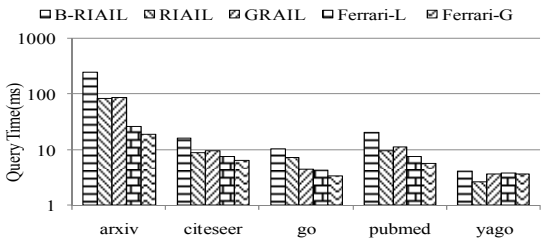


Fig.8 Query time of small dense graphs

图 8 小的稠密图查询时间

(3) 大的图数据

实验结果如图 9~图 11 所示.对于所有数据集,B-RIAIL 的索引创建时间是 RIAIL 的 2~3 倍.根据文献[13]中的实验,对于大部分稠密图,RIAIL 的索引大小小于 GRIPP,其原因与小的稠密图一样.

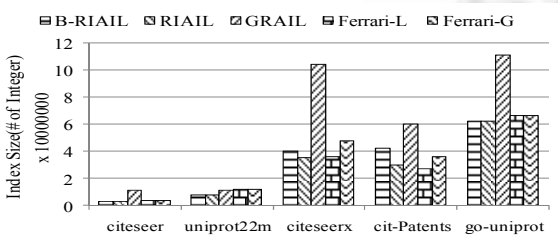


Fig.9 Index construction time of large graphs

图 9 大图索引创建时间

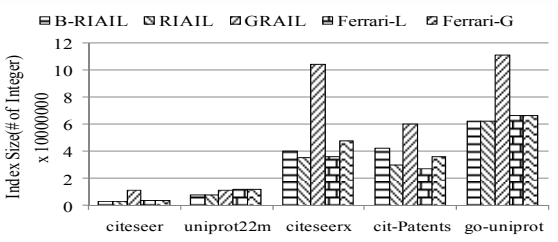


Fig.10 Index size of large graphs

图 10 大图索引大小

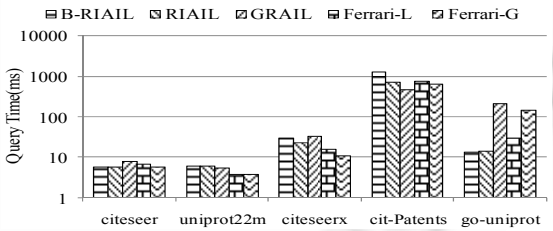


Fig.11 Query time of large graphs

图 11 大图索引查询时间

对于大的稀疏图(citeseer 和 uniprot22m),5 种算法都能处理.对于 citeseer,RIAIL 的查询性能最好.对于 uniprot22m,RIAIL 的查询性能与 GRAIL 接近.

在大的稠密图中,对于 cit-Patents,RIAIL 的查询性能与 Ferrari 接近.对于 citeseerx,RIAIL 的查询性能优于 GRAIL.RIAIL 可以采用小的稠密图中的改进方法来提高查询性能,而这时 GRAIL 和 Ferrari 却不能简单地通过增加结点标记的数量来改善查询性能,实际上,Ferrari 在标记 citeseerx 时就已经因为内存不能同时保存索引和原始图而进行磁盘 I/O 了.对于 go-uniprot,RIAIL 的查询性能最好,是 GRAIL 的 15 倍,是 Ferrari-L 和 Ferrari-G 的 2 倍和 10 倍.RIAIL 的查询性能优于 GRAIL,其原因与小的稀疏图一样.对于 Ferrari-L 和 Ferrari-G,因为内存不能同时保存索引和 go-unirpot,因此 Ferrari-L 和 Ferrari-G 的查询性能受到了磁盘 I/O 的影响.

(4) 人工生成的图数据

实验结果见表 8~表 10.对于两个数据集,B-RIAIL 和 RIAIL 都能创建索引,但 B-RIAIL 的索引创建时间大约是 RIAIL 的 2~4 倍,且 RIAIL 的索引大小明显小于 B-RIAIL.虽然 Ferrari-L 和 Ferrari-G 可以对 10m2x 创建

索引,但这时索引和原始图 10m2x 不能同时保存在内存中,查询时进行磁盘 I/O,导致查询不能在可接受的范围(-t 表示不能在 5 分钟内完成)内被处理.对于 10m5x 数据集,只有 RIAIL 可以处理,B-RIAIL,GRAIL,Ferrari-L 和 Ferrari-G 都不能在可接受的时间内处理(-m 表示不能在 5 分钟内生成索引).值得注意的是,GRAIL 针对这两个人工生成的数据集都不能处理,在生成索引时,GRAIL 就因内存不能同时保存索引和原始图导致索引创建不能在可接受的时间内完成.

Table 8 Index construction time of synthetic graphs (ms)

表 8 人工生成的图索引建立时间 (ms)

Data	B-RIAIL	RIAIL	GRAIL	Ferrari-L	Ferrari-G
10m2x	14 906	8 197.68	-t	117 935	118 310
10m5x	49 281.7	13 446.33	-t	-t	-t

Table 9 Index size of synthetic graphs(# of Integer)

表 9 人工生成的图索引大小(# of Integer)

Data	B-RIAIL	RIAIL	GRAIL	Ferrari-L	Ferrari-G
10m2x	66 400 380	57 238 009	-m	81 433 672	94 999 993
10m5x	196 823 223	89 050 907	-m	-m	-m

Table 10 Query time of synthetic graphs (ms)

表 10 人工生成的图查询时间 (ms)

Data	B-RIAIL	RIAIL	GRAIL	Ferrari-L	Ferrari-G
10m2x	47.01	44.21	-t	-t	-t
10m5x	-t	2 237.77	-t	-t	-t

7 结 论

可达性查询是图数据管理中的一个基本问题.随着各类应用的深入发展,图数据在类型和规模上都发生着巨大的变化,怎样适应这些变化,准确、快速、可扩展地处理可达性查询是一个很重要的问题.为此本文提出了一种处理可达性查询的索引算法,并在此基础上给出了查询算法.通过大量的实验说明了 RIAIL 算法的高效性和可扩展性.现实应用中的很多图数据是不断变化的,比如 Web 和社交网络.在今后的工作中,将深入研究动态环境下图数据的可达性查询,考虑改进 RIAIL 算法,以便能够处理动态图上的可达性查询.

致谢 感谢申老师、冯朔、李华和实验室师兄、师姐、师弟、师妹在论文写作过程中给予我的帮助,同时感谢审稿老师给本文提出的审稿意见和修改建议.

References:

[1] Aggarwal CC, Wang H. Managing and Mining Graph Data. Heidelberg: Springer-Verlag, 2010.

[2] Cormen TH, Leiserson CE, Rivest RL, *et al*. Introduction to Algorithms. 3rd ed., Cambridge: The MIT Press, 2009.

[3] Agrawal R, Borgida A, Jagadish HV. Efficient management of transitive relationships in large data and knowledge bases. In: Proc. of the 1989 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1989. 253−262.

[4] Cohen E, Halperin E, Kaplan H, *et al*. Reachability and distance queries via 2-hop labels. In: Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms. Philadelphia: ACM Press, 2002. 937−946.

[5] Wang H, He H, Yang J, *et al*. Dual labeling: Answering graph reachability queries in constant time. In: Proc. of the 22nd Int'l Conf. on Data Engineering. Washington: IEEE, 2006. 75−75.

[6] Cheng J, Yu J X, Lin X, *et al*. Fast computation of reachability labeling for large graphs. In: Proc. of the 10th Int'l Conf. on Advances in Database Technology. Berlin, Heidelberg: Springer-Verlag, 2006. 961−979.

[7] Cheng J, Yu J X, Lin X, *et al*. Fast computing reachability labelings for large graphs with high compression rate. In: Proc. of the 11th Int'l Conf. on Extending Database Technology: Advances in Database Technology. New York: ACM Press, 2008. 193−204.

[8] Jin R, Xiang Y, Ruan N, *et al*. 3-hop: A high-compression indexing scheme for reachability query. In: Proc. of the 2009 ACM SIGMOD Int'l Conf. on Management of data. New York: ACM Press, 2009. 813−826.

[9] Trißl S, Leser U. Fast and practical indexing and querying of very large graphs. In: Proc. of the 2007 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2007. 845−856.

[10] Jin R, Xiang Y, Ruan N, *et al*. Efficiently answering reachability queries on very large directed graphs. In: Proc. of the 2008 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2008. 595−608.

[11] Yildirim H, Chaoji V, Zaki MJ. Grail: Scalable reachability index for large graphs. Proc. of the VLDB Endowment, 2010,3(1-2): 276−284.

[12] Zhang Z, Yu J X, Qin L, *et al*. I/O cost minimization: reachability queries processing over massive graphs. In: Proc. of the 15th Int'l Conf. on Extending Database Technology. New York: ACM Press, 2012. 468−479.

[13] Yıldırım H, Chaoji V, Zaki MJ. GRAIL: A scalable index for reachability queries in very large graphs. The VLDB Journal—The Int'l Journal on Very Large Data Bases, 2012,21(4):509−534.

[14] Seufert S, Anand A, Bedathur S, *et al*. Ferrari: Flexible and efficient reachability range assignment for graph indexing. In: Proc. of the 2013 IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2013. 1009−1020.

[15] Dietz PF. Maintaining order in a linked list. In: Proc. of the 14th annual ACM Symp. on Theory of Computing. New York: ACM Press, 1982. 122−127.

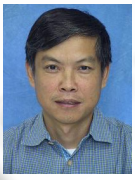
解宁(1988－),男,河北新乐人,硕士生,主要研究领域为图数据处理.
E-mail: andy.xning@gmail.com

寇月(1980－),女,博士,副教授,主要研究领域为实体识别,Web 数据管理.
E-mail: kouyue@ise.neu.edu.cn

申德荣(1964－),女,博士,教授,博士生导师,主要研究领域为 Web 数据处理,分布式数据库.
E-mail: shendr@mail.neu.edu.cn

聂铁铮(1980－),男,博士,副教授,主要研究领域为数据质量,数据集成.
E-mail: nietiezheng@ise.neu.edu.cn

冯朔(1989－),男,博士生,主要研究领域为图数据处理.
E-mail: fengshuo1989818@hotmail.com



于戈(1962－),男,博士,教授,博士生导师,主要研究领域为数据流,数据挖掘,分布式数据库.
E-mail: yuge@ise.neu.edu.cn