Journal of Computer Applications
计算机应用, 2020, 40(2): 426 - 433
ISSN 1001-9081
CODEN JYIIDU
2020-02-10
http://www.joca.cn

文章编号:1001-9081(2020)02-0426-08
DOI:10.11772/j.issn.1001-9081.2019081605

有向无环图上k步可达查询优化算法

杜　明[1],杨安平[1],周军锋[1*],陈子阳[2],杨　云[1]
(1. 东华大学 计算机科学与技术学院, 上海 201620; 2. 上海立信会计金融学院 信息管理学院, 上海 201620)
(*通信作者电子邮箱 zhoujf@dhu.edu.cn)

摘　要:k步可达查询用于在给定的有向无环图(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,提出一种基于部分点的双向最短路径索引来提升索引的可达信息覆盖率,并提出一组优化规则来减小索引规模;然后提出基于简化图的正反互逆拓扑索引来加速回答不可达查询;最后提出远距离优先的双向遍历策略来提高查询处理的效率。基于21个真实数据集(如引用网络、社交网络等)的实验结果表明,相比已有的高效方法PLL及BFSI-B,所提出的算法具有更小的索引规模和更快的查询响应速度。

关键词:有向无环图;k步可达性查询;hop点最短路径索引;双向互逆拓扑索引;双向遍历
中图分类号:TP311　　**文献标志码:**A

Optimized algorithm for k-step reachability queries on directed acyclic graphs

DU Ming[1], YANG Anping[1], ZHOU Junfeng[1*], CHEN Ziyang[2], YANG Yun[1]
(1. *School of Computer Science and Technology, Donghua University, Shanghai* 201620, *China*;
2. *School of Information Management, Shanghai Lixin University of Accounting and Finance, Shanghai* 201620, *China*)

Abstract: The k-step reachability query is used to answer whether there exists a path between two nodes with length no longer than k in a Directed Acyclic Graph (DAG). Concerning the problems of large index size and low query processing efficiency of existing approaches, a bi-directional shortest path index based on partial nodes was proposed to improve the coverage of reachable queries, and a set of optimization rules was proposed to reduce the index size. Then, a bi-directional reversed topological index was proposed to accelerate the unreachable queries answering based on the simplified graph. Finally, the farthest-node-first-visiting bi-traversal strategy was proposed to improve the efficiency of query processing. Experimental results on 21 real datasets, such as citation networks and social networks, show that compared with existing efficient approaches including PLL (Pruned Landmark Labeling) and BFSI-B (Breadth First Search Index-Bilateral), the proposed algorithm has smaller index size and higher query response speed.

Key words: Directed Acyclic Graph (DAG); k-step reachability query; hop point shortest path index; bi-directional reversed topological index; bi-traversal

0　引言

给定有向无环图(Directed Acyclic Graph, DAG)G,可达性查询$u?\rightarrow v$用于回答从顶点u出发是否存在一条路径可以到达顶点v。实际应用中,如社交网络、通信与传感器网络、生物网络、可扩展标记语言(eXtensible Markup Language, XML)数据、资源描述框架(Resource Description Framework, RDF)数据等,可达性查询可用于检测两点间是否存在特定关系。可达性查询处理是图数据管理与分析的基本操作之一,是研究者广泛关注的热点问题[1-2]。然而,实际中除了检测两点间是否存在特定关系,还需要考虑关系的强弱程度,如存在信号衰减的传感器网络,用户更感兴趣的是长度受限的可达性查询。传统的可达性查询只能回答两个顶点间是否可达,并不能确定在几步内可达。本文研究k步可达查询的高效处理问题。与基本的可达性查询相比,k步可达查询用于回答从u出发,是否存在长度在k步之内的路径到v。传统的可达查询可以看作$k=\infty$时的k步可达查询[3]。由于k步可达性查询体现着图中顶点对其他顶点的影响力,因而相对可达性查询而言,能提供更多的信息。

现有解决k步可达性查询的方法主要分为两类:Label-Only[4]和Label+G[5-7]。Label-Only类方法的主要思想是通过构建所有可达顶点间最短路径的索引,当判断从顶点u能否在k步内到达顶点v时,只需通过标签得到u和v之间的最短路径长度,然后比较从u到v的最短路径长度与k的大小关系。当最短距离大于k时,说明该查询不满足k步可达性,否则说明该查询满足k步可达性。Label+G类方法的主要思想是快速构建部分可达信息的索引,处理查询时,若索引可回答查询,则直接返回结果;否则需要通过图遍历来得到最终结果。相比较而言,Label-Only类方法的索引规模大、索引构建时间长,且当查询不可达时,处理效率低。而现有的Label+G类方法由于索引记录的可达信息少,查询处理时需要遍历的顶点较

收稿日期:2019-08-16;**修回日期:**2019-09-23;**录用日期:**2019-10-24。

作者简介:杜明(1975—),男,黑龙江虎林人,副教授,博士,主要研究方向:信息检索、数据分析和挖掘、自然语言处理; 杨安平(1994—),男,安徽马鞍山人,硕士研究生,主要研究方向:图形数据的查询处理和优化; 周军锋(1977—),男,陕西西安人,教授,博士,CCF会员,主要研究方向:信息检索、半结构化数据和图形数据的查询处理以及优化; 陈子阳(1973—),男,黑龙江五常人,教授,博士,CCF会员,主要研究方向:计算理论、查询处理、图数据优化; 杨云(1995—),女,安徽芜湖人,硕士研究生,主要研究方向:图形数据的查询处理和优化。

多,当满足条件的查询数量增多时,算法性能显著下降。

针对以上问题,本文提出一种求解k步可达查询的高效Label+G算法kRH。其基本思想是通过快速构建索引规模小、可达信息覆盖广的索引,从而可以在常量时间回答更多的查询,同时降低图遍历的代价。具体来说,本文的工作如下:

1)提出一种基于部分点的双向最短路径索引,用于在常量时间回答大部分k步可达查询,并提出三种启发式规则来减小索引规模。

2)提出基于简化图的正反互逆拓扑来提高不可达查询的处理效率。

3)提出远距离优先的双向搜索策略,并在此基础上提出高效的查询处理策略。

4)基于多个真实的数据集进行测试,实验结果显示,本文提出的方法比现有方法更高效。

1　背景知识及相关工作

1.1　数据模型及相关概念

本文处理有向无环图(DAG)上的k步可达查询。给定DAG,V表示顶点的集合,E表示边的集合。后续讨论中,用$(u,v)\in E$表示从u到v的边,$\delta(u,v)$表示u与v的最短路径,$Out(u)$表示u指向的顶点集,$In(u)$表示指向u的顶点集。$Out^*(u)$表示u可达的顶点集,$In^*(u)$表示可达u的顶点集。给定G中任意两个顶点u和v,若从u到v存在一条有向路径,则说明u可达v,否则说明u不可达v。若从u到v存在一条长度不大于k的路径,说明从u出发k步可达v,用$u\rightarrow_k v$表示;否则u不能在k步可达v,用$u\nrightarrow_k v$表示。

对于一个DAG来说,它体现顶点之间的偏序关系,通过拓扑排序,将顶点的偏序关系转变为全序关系,拓扑号即为全序关系的序号。顶点u、v的拓扑号分别为X_u和X_v,若存在$X_u>X_v$,u不可达v。

1.2　相关工作

1.2.1　可达查询

针对图论中的可达性查询,按照索引覆盖范围的大小,分为两大类:1)构建全索引的方式;2)构建部分索引的方式。全索引方式的主要思想是通过记录任意两个顶点之间的可达信息来处理可达查询。基于该思想,文献[8]采用传递闭包的方式记录任意两个顶点间的可达信息,该算法虽然能够在$O(1)$的时间复杂度内处理查询,缺点是空间复杂度是$O(|V|^2)$,实际中扩展性太差,无法应用于大图。文献[9]采用压缩传递闭包的方式,通过向上向下遍历给图中每个顶点v附加两个区间$L_{\mathrm{IN}}(v)$和$L_{\mathrm{OUT}}(v)$。当需要判断u与v是否可达时,仅需要判断u和v之间是否存在交集,若交集不为空,则说明u可达v;否则,u不可达v。部分索引方式的主要思想是通过记录部分顶点之间的可达信息配合遍历的方式来处理可达查询。基于该思想,GRIPP算法[10]给所有顶点附加一个区间,并通过区间包含来判断顶点间是否可达,若区间不包含,则需要通过遍历的方法才能判定。Grial算法[11]在GRIPP算法的基础上,采用不同的遍历方式给图中所有顶点附加多个区间,同样是采用区间包含判定是否可达,对于不能判断的查询,需要采用遍历的方式才能够判断。FELINE算法[12]通过给图中每个顶点附加两个整数,代表顶点在图中拓扑排序的序号,若顶点u的拓扑序号大于v的拓扑序号,则可以判定u不可达v。

1.2.2　Label-Only方法

Label-Only方法的思想是通过记录所有顶点对之间的最短路径长度来回答k步可达查询。基于该思想,一种简单的方法是直接记录所有顶点对之间的最短路径长度,该方法虽然可以$O(1)$时间回答k步可达查询,但空间复杂度为$O(|V|^2)$,在实际应用中可扩展性太差,无法处理大图。针对该问题,文献[9]提出一种压缩的最短路径索引PLL(Pruned Landmark Labeling)。其基本思想是为图中的每一个顶点v建立$L_{\mathrm{IN}}(v)$和$L_{\mathrm{OUT}}(v)$两个标签,其中$L_{\mathrm{IN}}(v)$标签用来记录顶点v与$In^*(v)$中部分顶点间的最短路径,$L_{\mathrm{OUT}}(v)$用来记录v与$Out^*(v)$中部分顶点间的最短路径。则u和v之间的最短路径$\delta(u,v)$的计算问题就转换为求解中间节点和u、v最短路径之和的最小值问题。当需要回答从u到v是否k步可达时,只需判断$L_{\mathrm{IN}}(v)$和$L_{\mathrm{OUT}}(v)$是否存在交集。若不存在交集,则说明两个顶点不满足k步可达;若存在交集,说明u通过交集中的任意一个顶点都可到达v。基于交集结果,可得到所有交集顶点与u和v之间的最短路径之和,则u和v之间的最短路径$\delta(u,v)$就等于最短路径之和中的最小值。如果$\delta(u,v)\leqslant k$,说明$u\rightarrow_k v$;否则,说明$u\nrightarrow_k v$。

该方法的问题在于两方面:1)索引规模大、索引时间长,原因在于需要记录完整的最短路径信息;2)对不可达查询的处理效率较低,原因是$L_{\mathrm{OUT}}(u)$与$L_{\mathrm{IN}}(v)$的交集为空意味着需要访问$L_{\mathrm{OUT}}(u)$与$L_{\mathrm{IN}}(v)$中的所有信息。

1.2.3　Label+G方法

由于通过直接遍历回答查询的效率太低,而Label-Only方法的索引时间长、索引规模大,研究者试图通过仅记录部分k步可达信息来在效率方面进行平衡。其基本思想是在线性或者近似线性时间复杂度内记录部分k步可达信息,用于减小遍历操作的搜索空间,根据其剪枝能力的不同,分为三种类型的方法。

第一类方法是直接使用传统的可达查询处理方法[10-11]来回答不可达查询,对于可达查询,通过直接遍历的方法检测是否满足k步的条件。该类方法虽然索引时间少,索引规模小,但当满足条件的查询数量增多时,效率异常低下。

第二类方法是构建可回答部分k步可达查询的索引来加快查询处理的速度。文献[5-7]基于顶点覆盖技术构建k步索引。其基本思想是首先求得原图的一个顶点覆盖集合,然后在顶点覆盖集上构建传递闭包作为索引,最后根据该索引来处理k步可达查询。这种方法的问题是索引只能回答固定k值的可达查询,扩展性较差。

第三类方法[3]采用FELINE算法[12]作为特定的剪枝条件,来快速回答不可达查询,对于其他查询,则提出基于广度优先生成树以及广度层作为剪枝条件来提高k步可达查询的处理效率。该方法为生成树的每个顶点附加一个区间,用于判断顶点之间的祖先后代关系。若$L(v)\subset L(u)$,说明v是u的后代,则查询可通过比较u和v之间的层数差来判断;若$L(v)\not\subset L(u)$,说明v不是u的后代,则需要递归判断u的后代中是否存在顶点p,满足$L(v)\subset L(p)$。该算法的缺点是生成树的区间覆盖率很低,因此,当满足条件的查询数量增多时,查询效率较低。

相比较而言,Label-Only方法虽然查询处理时无需遍历操作,但索引规模大、索引构建时间长,同时对不可达查询的处理效率较低;而Label+G方法虽然索引构建时间短、索引规模小,但查询处理时因需要遍历操作而显得效率较低,尤其是当满足条件的查询增多时,效率显著下降。本文针对该问题,提出一种增强的Label+G方法,在降低索引规模的前提下,通过

增强不可达查询和可达查询的剪枝效果来缩减遍历操作的搜索空间，提升k步可达查询的处理效率。

2 基于部分点的双向最短路径索引

2.1 双向最短距离索引的基本思想

本文提出基于部分出度与入度之和较大的顶点（简称hop点）建立双向最短路径索引来加速k步可达的判断，这里双向的意思是不仅记录a与其可达点的最短距离，而且记录a与可达a的点的最短距离。具体来说，类似于2hop索引[4]，本文为每个点v设定两个标签$L_{IN}(v)$和$L_{OUT}(v)$。和2hop索引的不同在于，两个标签中不仅仅记录了可达信息，而且记录了最短路径信息。其中前者$L_{IN}(v)$用于存储可达v及其与v的最短路径，后者$L_{OUT}(v)$存储v可达的点及其与v的最短距离。

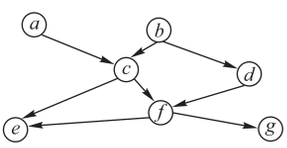

图1 有向无环图G的示例

Fig. 1 Example of DAG G

例如，对图1的G，假设顶点的处理顺序是$c \to a \to b \to d \to e \to f \to g$。当处理$c$时，其可达点为$c, e, f, g$，可达$c$的点为$a, b$。因此将元组$\langle c, dis\rangle$加入到$L_{IN}(u)$（其中$u \in \{c, e, f, g\}$）中，这里$dis$表示$c$到$u$的最短路径长度。类似地，将$\langle c, dis\rangle$加入$L_{OUT}(v)$中（其中$v \in \{a, b\}$），这里$dis$表示$v$和$c$的最短路径长度。表1是基于图1所有顶点构建的最短路径索引。

表1 基于所有顶点的最短路径索引

Tab. 1 Shortest path index based on all points

顶点	L_{IN}	L_{OUT}
a	$\{a,0\}$	$\{a,0\}\{c,1\}\{f,2\}\{e,2\}$ $\{g,3\}$
b	$\{b,0\}$	$\{b,0\}\{c,1\}\{d,1\}\{e,2\}$ $\{f,2\}\{g,3\}$
c	$\{a,1\}\{b,1\}\{c,0\}$	$\{c,0\}\{e,1\}\{f,1\}\{g,2\}$
d	$\{b,1\}\{d,0\}$	$\{d,0\}\{f,1\}\{g,2\}\{e,2\}$
e	$\{a,2\}\{b,2\}\{c,1\}\{d,2\}$ $\{f,1\}\{e,0\}$	$\{e,0\}$
f	$\{a,2\}\{b,2\}\{c,1\}\{d,1\}$ $\{f,0\}$	$\{f,0\}\{g,1\}\{e,1\}$
g	$\{a,3\}\{b,3\}\{c,2\}\{d,2\}$ $\{f,1\}\{g,0\}$	$\{g,0\}$

基于表1的索引，则给定任意两点a和b，判断a是否可在k步到达b，则只需比较$L_{OUT}(a)$和$L_{IN}(b)$：如果二者的交集为空，说明a不可达b；否则说明a可达b。如果a可达b，将a通过交集中每个点到达b的路径长度计算出来，取其最小值即为a和b的最短路径长度。如果该长度小于给定的k值，则说明a可以在k步可达b，否则a不能在k步到达b。

2.2 双向最短距离索引的优化

虽然该索引可回答所有查询，且无需在图上执行遍历操作，但其索引规模太大，且求解交集的代价太高。为此，提出仅构建部分点的双向路径索引来减小索引规模。原因在于：1）实际中的图存在少量度很大的点，相应地，其可达顶点也通常比较多，选择这样的点构建双向最短路径可以显著增加所维护的可达点对数量及通过这种点的最短路径长度。2）我们的实验表明，对一般图来说，要么32个点已经维护了足够多的可达信息，再增加顶点后，可达信息并没有明显增加；要么32个点仅维护少量可达信息，再增加这种点同样不会显著增加可达信息维护量。如图2所示，从实际数据上的测试结果可以看出：对于amaze数据集，即使$k=1$，双向索引的可达覆盖率也可接近100%；而对cit-Patents而言，即使k值增加到32，可达覆盖率依然非常小；另外两个数据集上，当k值增加到16以后，数据基本不再发生变化。

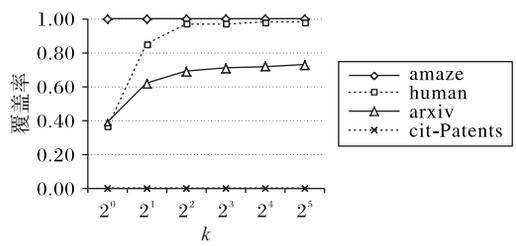

图2 k个hop点可达覆盖率

Fig. 2 Coverage of reachability for k hop nodes

基于以上观察，取32个点构建双向最短路径索引。假设对于图1所示的DAG G，本文取c和f两个点构建双向最短路径索引，如表2所示。显然，和表1的索引相比，表2的规模降低了。实际上，表2还可以进一步优化。

表2 基于c和f的最短路径索引

Tab. 2 Shortest path index based on c and f

顶点	L_{IN}	L_{OUT}
a	$\varnothing$	$\{c,1\}\{f,2\}$
b	$\varnothing$	$\{c,1\}\{f,2\}$
c	$\{c,0\}$	$\{c,0\}\{f,1\}$
d	$\varnothing$	$\{f,1\}$
e	$\{c,1\}\{f,1\}$	$\varnothing$
f	$\{c,1\}\{f,0\}$	$\{f,0\}$
g	$\{c,2\}\{f,1\}$	$\varnothing$

定理1 当基于顶点v构建双向最短路径索引时，如果向上（向下）遍历的过程中遇到了已处理的hop点u，则无需从u继续向上（向下）遍历。

证明 如图3所示，假设x可达u，当前处理的是hop点v，hop点u在v之前处理。当从v向上遍历时遇到了u。基于u得到的标签，假设求得x到u的距离是d_1，u到v的距离是d_2，则x通过u到v的距离是d_1+d_2。显然，从v向上遍历遇到u，所求u与v之间的距离仍然是d_2，因此即使在u不停止，所得x和v的距离仍然是d_1+d_2，因此，无需从u继续向上遍历。

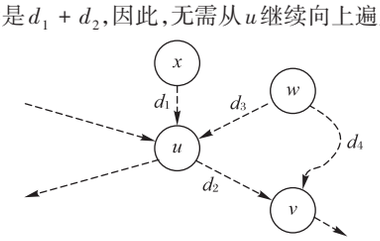

图3 hop点作用示意图

Fig. 3 Schematic diagram of hop nodes intension

定理2 当基于顶点v构建双向最短路径索引时，如果向上（向下）遍历的过程中通过非hop点遇到了w，且w通过u到v

的距离小于w通过非hop点到v的距离,则无需从w继续向上(向下)遍历。

证明　如图3所示,假设hop点u先处理,目前处理的是hop点v。假设从v向上遍历,通过非hop点到达w,路径长度是d_4。如果w和u的最短距离是d_3,且$d_3+d_2\leqslant d_4$,显然无需在$L_{\mathrm{OUT}}(w)$中记录其和v的距离d_4,因为d_4不是最短距离。因此,无需从w开始继续向上遍历。

基于定理1和定理2,在求解双向最短路径索引的过程中可有效减小索引规模,提高索引构建的速度。例如,对图1的G而言,假设选择两个hop点c和f来构建双向最短路径索引。先处理的是点c,当处理第二个点f时,当从其向上遍历遇到第一个hop点c时,可根据定理1立即停止遍历,当从非hop点遍历到b时,可根据定理2立即停止遍历。当处理完这两个点后,所得到的双向路径索引如表3所示。

表3　基于c和f的优化后的最短路径索引

Tab. 3　Optimized shortest path index based on c and f

顶点	L_{IN}	L_{OUT}
a	∅	$\{c,1\}$
b	∅	$\{c,1\}$
c	$\{c,0\}$	$\{c,0\}$
d	∅	$\{f,1\}$
e	$\{c,1\}\{f,1\}$	∅
f	$\{c,1\}\{f,0\}$	$\{f,0\}$
g	$\{c,2\}\{f,1\}$	∅

3　基于栈的正反互逆拓扑

可达是k步可达的必要条件,因此,若查询不可达,则必定k步不可达。基于此,本文使用拓扑序号为基础的方法来快速判断不可达。其基本思想是:给定两个顶点u和v,若u的拓扑号大于v的拓扑号,则u不可达v,因此u在k步内不可达v。

文献[3]通过使用互逆的两个拓扑号来增强不可达查询的判断能力。具体而言,给定第一个拓扑顺序,在构建逆序拓扑时,始终优先访问第一个拓扑号最大且所有入邻居均被访问的顶点。以此建立的拓扑称为第一遍拓扑的逆向拓扑。虽然实际中互逆拓扑可快速判断很多不可达查询,但仍然存在两方面的问题。

首先,在实际应用中,基于正向建立的两个互逆拓扑号仍然存在很多不可达查询无法判断的问题。针对该问题,本文提出构建正反互逆拓扑来过滤更多的不可达查询。其基本思想是:首先,删除已处理的hop点。原因是与hop点关联的最短路径均已处理完毕,后续的最短路径计算与其无关。删除hop点之后,数据图得以很大程度上的简化。其次,自顶向下、以深度优先的方式构建正向互逆拓扑。最后,沿着边的相反方向,以自底向上,以深度优先的方式构建反向互逆拓扑。

其次,文献[12]基于堆求解逆序拓扑,其时间复杂度为$O(|V|\log|V|)$。本文设计了高效算法在线性时间构建正反互逆拓扑。与以上工作相比,由于首先删除了hop点,因而所建立的正反互逆拓扑可以过滤更多的不可达查询。和文献[12]相比,本文提出的正反互逆拓扑索引具有线性的时间复杂度,同时能过滤更多的不可达查询。和文献[3]的方法相比,由于我们在求解正反互逆拓扑之前去除了hop点,数据图得以很大程度的简化,因而效率更高。

4　索引构建

4.1　基于hop点的最短距离索引构建

4.1.1　算法描述

给定DAG G,求解hop点的最短路径索引总体上分为3步。首先求解hop点,然后基于hop点执行广度优先遍历(Breadth First Search, BFS)建立L_{IN}索引,最后基于hop点执行反向BFS建立L_{OUT}索引。下面分别予以说明。

(1) 根据$(In(v)+1)*(Out(v)+1)$从大到小的顺序选择前32个顶点作为hop点。如算法1第1)行。

(2) 从hop点v开始执行BFS,对于遍历到的顶点u,若是通过标签计算得到了长度大于从v到u的路径长度,则更新$L_{\mathrm{IN}}(u)$并继续访问u可达的点;若是通过标签计算得到的长度小于从v点到u的路径长度,则不更新$L_{\mathrm{IN}}(u)$且不再从u出发继续BFS。如算法1第2)~12)行所示。

(3) 从hop点开始执行反向BFS,对于遍历到的顶点u,若是通过标签得到的长度大于从hop点到u的路径长度,则更新$L_{\mathrm{OUT}}(u)$并且访问u的父亲;若通过标签得到的长度小于从hop点到u的路径长度,则不更新$L_{\mathrm{OUT}}(u)$且不访问u的父亲。如算法1第13)行所示。

算法1　hop$(G=(V,E))$。

输入　$G=(V,E)$;

输出　hop点最短路径索引。

1)　选择32个hop点
2)　for each hop node v do
3)　　push v into Q
4)　/*从v开始正向遍历,求路径索引*/
5)　while Q is not empty do
6)　　pop u from Q
7)　　if $Q_U(v,u,L_{\mathrm{OUT}}(v),L_{\mathrm{IN}}(u))\leqslant Dis(v,u)$
8)　　　continue
9)　　$L_{\mathrm{IN}}(u)\leftarrow L_{\mathrm{IN}}(u)\cup\{(u,Dis(v,u))\}$
10)　　for each $p\in Out(u)$ do
11)　　　$Dis(v,p)\leftarrow Dis(v,u)+1$
12)　　　push p into Q
13)　反向遍历v,重复第3)~12)行,求路径索引

算法1用于求解基于hop顶点的最短路径,其中$Q_U(v,u,L_{\mathrm{OUT}}(v),L_{\mathrm{IN}}(u))$表示通过标签求解得到从$v$到$u$的最短路径长度,$Dis(v,u)$表示基于BFS得到从$v$到$u$路径长度,初始值为0,$Q$表示队列。算法1中第1)行按照规则选择度大的32个顶点作为hop点。第2)、3)行将hop点依次执行入队操作。当队列不空时,执行出队操作(第5)、6)行)。若通过标签求得的最短路径不大于遍历的路径长度,则停止访问其后代(第7)、8)行);否则,更新L_{IN}标签,并将其后代加入到Q中(第9)~12)行)。将hop点v再入队列,通过执行反向BFS构建L_{OUT}(第13)行)。

注意,算法1中需要在第7)行调用函数$Q_U(v,u,L_{\mathrm{OUT}}(v),L_{\mathrm{IN}}(u))$求解根据已处理hop点得到的路径长度。这一操作需要比较L_{OUT}和L_{IN},即执行集合求交集的操作。为了加快集合交集操作的处理,需要先对集合中的顶点按照编号,或者某种序号进行排序。

4.1.2　索引优化

若v和u之间通过hop点有路径相连,则函数$Q_U(v,u,L_{\mathrm{OUT}}(v),L_{\mathrm{IN}}(u))$进行集合交集操作的结果是经过hop点相连的路径长度,无需知道hop点是谁。因此,采用如下规则对算

法1进行优化。

规则1 所有顶点的L_{IN}或者L_{OUT}中无需存储顶点自身编号，只需存储代表顶点处理顺序的数字即可。

根据规则1，构建索引时，无需执行排序操作，对第i个处理的hop点，直接在相应顶点的标签中加入$\langle i, dis\rangle$即可，这里的i表示当前hop点是第i个被处理的hop点，dis是hop点和当前点的最短路径距离。

进一步，由于只选择了32个hop点，且实际中数据图的拓扑层数有限，因此无需使用两个整数来表示标签中的元组$\langle i, dis\rangle$。利用规则2来进一步减少索引规模。

规则2 将每个标签中的每个元组$\langle i, dis\rangle$的两个数字用一个整数的前半段和后半段表示，从而将索引规模降低一半。

算法1的时间复杂度是$O(|V|+|E|)$。原因是，每个点的遍历操作代价最坏情况下是$O(|V|+|E|)$，而本文算法处理了32个点，因此，时间复杂度仍是$O(|V|+|E|)$。

4.2 正反互逆拓扑构建

求解正反互逆的拓扑号索引总体上分为4步：首先删除32个hop点，然后求解正向的拓扑号X，接着基于X求其逆向拓扑号Y，最后在反向图上求解两个拓扑号索引M和N。

给定一个有向无环图G，可以通过以下步骤得到X：1)将G中入度为0的顶点入栈S。2)对栈中元素执行以下操作：(a)将S的栈顶元素v出栈，并标记v的拓扑顺序；(b)删除v及所有从v出发的边；(c)将v的孩子中入度为0的顶点插入S。3)重复步骤2)直到S为空，相应的顶点出栈顺序称为X。

给定X，同样基于栈可以得到其逆向拓扑顺序Y。基本操作方法如下：首先所有入度为0的顶点按照它们在X中从小到大的顺序进入栈S。对栈中元素执行以下操作：1)将S的栈顶元素v出栈，并标记v的逆向拓扑号；2)删除v及从v出发的边；3)将v的孩子中入度为0的顶点按照它们在X中从小到大的顺序入栈。执行上述步骤，直至栈空即可得到第二次拓扑顺序Y。

定理3 求解逆向拓扑时，栈顶元素和使用堆选择的拓扑号最大的元素为相同顶点。

证明 首先，使用大顶堆时，堆顶元素是当前堆中拓扑号最大的顶点。其次，使用栈时，本文首先将入度为0的顶点按照第一遍拓扑的顺序从小到大入栈，因此可以保证对初始入度为0的顶点，栈顶元素的拓扑号最大。最后，由于栈顶元素的拓扑号最大，其孩子的拓扑号必然也大于栈中元素的拓扑号。每处理一个栈顶元素，其孩子中可进一步拓扑的顶点也是按照第一遍拓扑的顺序从小到大入栈，这样可以保证栈顶元素始终具有最大的拓扑号。因而可以保证每次处理的都是可拓扑的顶点中拓扑号最大的。

算法2 genTopo($G=(V,E)$)。

输入 $G=(V,E)$;

输出 所有顶点v的四个拓扑序号X, Y, M, N。

1) 从G中删除32个hop点
2) Construct(G, X)
3) Construct(G, Y)
4) 将G的边反转，执行第1)~2)行，求解反向互逆拓扑

第2)、3)行都是调用的Construct函数，代码如下所示：

Function Construct($G=(V,E), H$)

1) 将入度为0的顶点按照特定顺序入栈S
2) $topoNum \leftarrow 0$
3) while $S \neq \varnothing$ do
4) $v \leftarrow pop(S)$
5) $H_V \leftarrow topoNum$
6) $topoNum \leftarrow topoNum + 1$
7) for each $u \in Out(v)$ do
8) $insize(u) \leftarrow insize(u) - 1$
9) if $insize(u) = 0$ then push u into S

在得到正向互逆的拓扑后，将图的边反转，在反向图上执行上述步骤，从而得到反向图上的拓扑号M和N。最终，这4个拓扑顺序构成本文提出的正反互逆的拓扑号索引。

算法2用于求解给定的DAG G的正反互逆拓扑序号，其中第2)行调用函数Construct求解第一个拓扑顺序X，第3)行求X的逆向拓扑Y，第4)行求解反向互逆拓扑M和N。函数Construct用于根据特定顺序得到一个拓扑序号，其中S为栈，H_v表示顶点的拓扑序号。$insize(v)$表示存储顶点v入度的临时数组。第1)行将图中入度为0的顶点入栈。从第3)~9)行的迭代中，每循环一次处理一个顶点。具体做法是，第4)~5)行元素出栈并设定其拓扑号，然后在第7)~9)行将其出邻居的入度减1，若入度为0，则将其入栈。需要注意的是，算法第1)行，对于第一和第三遍拓扑，特定顺序不代表任何含义，只要找到一个入度为0的顶点就将其入栈。第二遍和第四遍求的分别是第一、三遍的逆向拓扑，这时，特定顺序代表在第一边拓扑后，根据第一遍拓扑序号的升序将入度为0的顶点入栈，这样就能保证第一、二遍拓扑互逆，第三、四遍拓扑互逆。另外，第7)行按照存储顺序处理，当执行第一遍拓扑后，图的邻接表中顶点按照拓扑号升序排序，且第二遍处理的是基于拓扑排序后的邻接表进行处理。这么做是为了在保证互逆的基础上，避免排序操作。

对图1的G而言，在求解正反拓扑号之前，首先删除32个hop点，得到简化的图，之后执行算法2后，给每个顶点赋予4个拓扑号。

给定DAG $G=(V,E)$，删除32个hop点的操作可在线性时间$O(|V|+|E|)$内完成。由于函数Construct在执行过程中始终没有执行特定的排序操作，在每次执行函数Construct时，每个顶点v的处理代价是$O(|Out(v)|)$，因此函数Construct的时间复杂度是$O(|V|+|E|)$。算法2总共调用了4次函数Construct，因此算法2的时间复杂度是$O(|V|+|E|)$。

总的来看，加上正反互逆拓扑，本文的索引构建时间复杂度是$O(|V|+|E|)$。

5 查询

算法3用于判定查询的k步可达性。其中$Q_U(v, u, L_{OUT}(v), L_{IN}(u))$表示通过标签求解得到的最短路径长度。算法3的第1)、2)行采用hop点的最短路径索引判定该查询是否满足k步可达性。具体来说：若通过最短路径索引求得从u到v的最短路径不大于k，说明该查询满足k步可达；否则若最短路径大于k并且查询中的顶点存在hop点，说明该查询不满足k步可达性(第3)行)。若上述条件均不能判定查询的k步可达性，则在第4)、5)行采用正反互逆的拓扑号判定查询的可达性：若查询不可达，则说明该查询不满足k步可达；否则，在第6)~13)行以递归的方式执行遍历操作。

算法3 kRH ($u, v, k, G=(V,E)$)。

输入 $G=(V,E)$;

输出 true or false。

1) if $Q_U(v, u, L_{OUT}(v), L_{IN}(u)) \leqslant k$
2) return true

```
3)  if ∃u,v ∈ hops then return false
4)  if X_u > X_v or Y_u > Y_v or M_u < M_v or N_u < N_v
5)    return false
6)  if Out(u) ≤ In(v) then
7)    for each p in Out(u) do
8)      if kRH(p,v,k − 1)
9)        return true
10) else
11)   for each p in In(v) do
12)     if kRH(u,p,k − 1)
13)       return true
```

基于如下两个启发式来加速遍历的过程。

规则3　给定k步可达查询的两个端点u和v。若u的出度大于v的入度,则从v出发向上遍历找u;否则从u出发向下遍历找v。

规则4　当从v出发遍历时,优先选择与其拓扑层相差大的顶点w进行遍历。

规则4中拓扑层相差较大,说明v通过w可能更快到达目标点。

6　实验与结果分析

6.1　实验环境

根据相关工作分析可知,在Label-Only类算法中,PLL查询效率最高;在Label+G类算法中,根据文献[3],BFSI-B(Breadth First Search Index-Bilateral)算法比k-reach[5]更快,是Label+G类算法中查询效率最高的算法。因此,本文实验中用于比较的算法有BFSI-B[3]和PLL[9]。对于Label+G中第一类方法而言,例如GRAIL,因其并非专门处理k步可达查询,本文不再进行比较。所有算法都采用C++实现,并使用GCC7.3.0进行编译。所有算法使用相同的数据集和查询集,并在相同的平台上运行得到实验数据。机器配置是4 GB RAM的Intel Core i5-3230 CPU(8核,2.60 GHz)和Ubuntu18.04LTS。以索引规模、索引构建时间、查询时间作为评价指标来比较三种算法的性能。本文方法中的hop点个数是32。

6.2　实验数据集

实验数据集由21个真实的数据集组成,这些数据集都广泛地出现在图论的可达性查询研究[14-16]中,它们的统计信息如表4所示。其中arxiv、citeseer、pubmed来自引文献[14];cit系列的数据集来自文献[6],其非叶子顶点的平均出度为10到30;uniprot100m、uniprot150m来自Uniprot数据库的注释联合图[15],皆是UniProt的完整资源描述框架(Resource Description Framework, RDF)的子集;soc-LJ(soc-LiveJournall)是来自社会网络的有向无环图;wikiTalk是来自维基百科的有向无环图;web-Google是来自Google网页的有向无环图;dbpedia为知识图;web-uk是文献[16]收集的网络有向无环图。由表4可知,除了前5个数据集的规模较小之外,其余数据集规模较大;$|V|$表示给定有向无环图的顶点数量,$|E|$为边的数量,$|d|$表示顶点的平均度。对于每个数据集,本文使用随机生成的100万个查询进行测试,算法的运行时间为处理100万个查询的总时间。

6.3　索引大小及时间

表5和表6分别给出了不同方法的索引大小和索引构建时间。从索引大小来看,如表5所示,可以看出:1)在所有数据集上,本文方法构建的索引规模最小。2)对于规模较小的图而言,三种方法的索引规模相差不大。3)当图是稠密图时,PLL算法建立的索引规模最大。原因在于,PLL要构建所有点的双向索引,而BFSI-B的索引规模与图的顶点个数成正比,每个顶点对应8个整数。本文方法kRH为每个顶点设定4个拓扑号。虽然建立的是32个hop点的索引,但由于使用了提前终止条件以及每个hop点关联的顶点有限,平均到每个点,hop标签不会超过4个数字,因此索引规模比BFSI-B小。

表4　实验数据集统计信息

Tab. 4　Statistics of experimental datasets

数据集	$\|V\|$	$\|E\|$	$\|d\|$
human	38 811	39 576	1.01
citeseer	10 720	44 258	4.13
pubmed	9 000	40 028	4.44
yago	6 642	42 392	6.38
arxiv	6 000	66 707	11.12
Email-EuAll	231 000	223 004	0.97
uniprot100m	16 087 295	16 087 293	1.00
uniprot150m	25 037 600	25 037 598	1.00
WikiTalk	2 281 879	2 311 570	1.01
soc-LJ	971 232	1 024 140	1.05
web-Google	371 764	517 805	1.39
10cit-Patent	1 097 775	1 651 894	1.51
10citeseerx	770 539	1 501 126	1.95
citeseerx	6 540 401	15 011 260	2.30
dbpedia	3 365 623	7 989 191	2.37
govwild	8 022 880	23 652 610	2.95
cit-Patents	3 774 768	16 518 947	4.38
go_uniprot	6 967 956	34 769 339	4.99
10go-unipr	469 526	3 476 397	7.40
twitter	18 121 168	18 359 487	1.01
web-uk	22 753 644	38 184 039	1.68

表5　不同数据集上的索引大小　　单位:MB

Tab. 5　Index size on different datasets　　unit: MB

数据集	PLL	BFSI-B	kRH
human	0.91	1.18	0.39
citeseer	0.77	0.33	0.12
pubmed	0.70	0.27	0.11
yago	0.51	0.20	0.08
arxiv	2.68	0.18	0.07
Email-EuAll	5.22	7.05	3.25
uniprot100m	393.76	490.95	240.47
uniprot150m	635.14	764.09	342.04
WikiTalk	52.54	69.67	30.82
soc-LJ	22.73	29.64	11.82
web-Google	10.12	11.35	4.67
10cit-Patent	30.22	33.50	13.75
10citeseerx	39.28	23.52	8.76
citeseerx	276.14	199.60	90.80
dbpedia	121.90	102.71	46.36
govwild	398.22	244.84	116.42
cit-Patents	1 748.75	115.20	53.60
go_uniprot	532.72	212.64	197.32
10go-unipr	46.63	14.33	6.17
twitter	416.68	553.01	236.51
web-uk	967.76	694.39	307.19

在索引时间方面,如表6所示。相比PLL,kRH算法的索引构建时间更短,尤其当图的规模大且稠密时更加明显;相比BFSI-B,二者索引时间相差不大。

表 6 不同数据集上的索引时间 单位:ms

Tab. 6 Index time on different datasets unit: ms

数据集	PLL	BFSI-B	kRH
human	10.60	5.89	5.24
citeseer	22.24	3.09	2.96
pubmed	24.35	2.99	3.20
yago	7.45	2.61	2.55
arxiv	232.31	2.58	11.45
Email-EuAll	78.03	38.41	41.55
unprot100m	6 642.67	4 299.18	3 817.71
unprot150m	12 033.31	6 776.78	6 725.61
WikiTalk	617.26	369.50	318.81
soc-LJ	360.48	172.819	185.40
web-Google	223.31	99.04	101.70
10cit-Patent	964.10	614.03	420.48
10citeseerx	1 690.31	314.29	254.37
citeseerx	14 563.31	5 012.91	4 166.43
dbpedia	3 989.27	1 777.21	1 835.07
govwild	12 069.51	3 347.31	3 769.61
cit-Patents	476 502	8 040.33	6 065.75
go_uniprot	15 748.51	4 079.98	7 624.27
10go-unipr	1 453.04	328.76	584.98
twitter	6 727.98	3 712.88	3 664.62
web-uk	47 093.21	3 703.31	4 686.35

6.4 查询时间

表 7 展示了三种算法当 $k=3$ 时的查询处理时间。由表 7 可知,三种方法相比,本文算法所需时间最短。原因有两方面:一方面,和 PLL 相比,本文算法可以在常量时间处理不可达查询;另一方面,和 BFSI-B 相比,本文算法基于 4 个拓扑号,可在常量时间检测更多的不可达查询。由于 PLL 不能常量时间回答查询,因此在表 8 中和 BFSI-B 比较了常量时间内可判定的查询个数,可以看出本文算法可在常量时间内判定更多的查询,因而可以获得比 BFSI-B 更高的查询响应速度。

表 7 $k=3$ 时不同数据集上的查询时间 单位:ms

Tab. 7 Query time on different datasets when $k=3$ unit: ms

数据集	PLL	BFSI-B	kRH
human	54.04	42.36	22.49
citeseer	94.97	84.81	46.68
pubmed	63.44	61.99	38.49
yago	51.48	35.67	17.67
arxiv	371.15	336.50	145.22
Email-EuAll	86.57	57.71	37.03
unprot100m	71.09	60.83	11.49
unprot150m	65.43	62.15	11.45
WikiTalk	105.08	77.17	48.57
soc-LJ	153.13	124.11	76.81
web-Google	141.81	132.59	69.11
10cit-Patent	88.39	39.58	15.16
10citeseerx	143.41	74.84	54.21
citeseerx	174.43	126.10	97.17
dbpedia	149.32	131.11	72.29
govwild	172.26	127.73	97.14
cit-Patents	539.40	158.39	96.79
go_uniprot	184.28	54.32	36.62
10go-unipr	154.07	58.70	54.03
twitter	214.22	155.51	120.57
web-uk	248.95	242.83	207.51

6.5 不同 k 值的查询处理性能

图 4 分别展示了三种算法在不同特征的数据集上处理 100 万个查询时随 k 值变化的情况。可以看出,本文方法和 PLL 对 k 值的变化不敏感,而 BFSI-B 的性能受 k 值影响较大,随着 k 值的增加,其所需的查询处理时间增长较多。对于稀疏数据集来说,k 值的变化对于 BFSI-B 和 kRH 算法的查询时间影响不大;而对于稠密数据集(go_uniprot)来说,BFSI-B 和 kRH 算法的查询时间随着 k 的增大而增加,但是 BFSI-B 的增加幅度更大。这是因为 kRH 算法比 BFSI-B 覆盖程度更高。

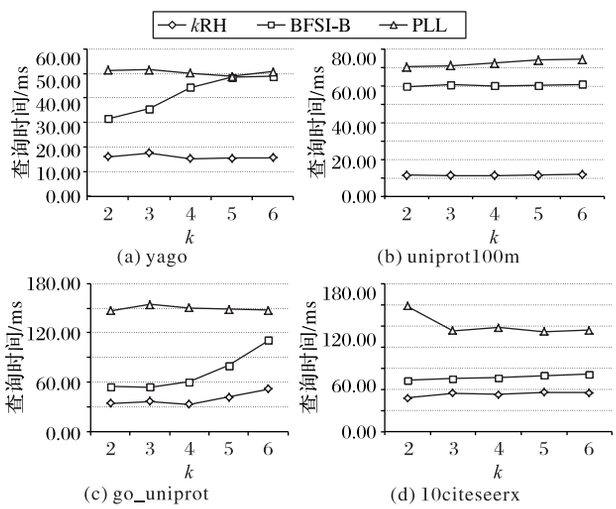

图 4 不同数据集上三种算法的查询时间对比

Fig. 4 Query time comparison of three algorithms on different datasets

表 8 $k=3$ 时常量时间内可判定的查询数量

Tab. 8 Query answers can be determined in constant time when $k=3$

数据集	BFSI-B	kRH
human	799 564	999 956
citeseer	794 836	911 257
pubmed	760 636	926 193
yago	711 253	984 771
arxiv	603 701	771 523
Email-EuAll	698 539	999 939
uniprot100m	808 389	999 995
uniprot150m	785 811	999 994
WikiTalk	882 736	999 853
soc-LJ	753 982	998 973
web-Google	754 241	986 717
10cit-Patent	847 771	998 752
10citeseerx	710 964	977 381
citeseerx	646 989	977 857
dbpedia	888 168	986 482
govwild	891 929	980 872
cit-Patents	717 408	998 286
go_uniprot	877 038	965 690
10go-unipr	846 093	932 241
twitter	854 191	999 701
web-uk	745 658	972 058

7 结语

针对已有 k 步可达查询方法存在的索引规模大、查询响应速度慢的问题,本文提出通过构建大度顶点的双向最短路径来提高可达查询的覆盖率,同时提出基于简化图的正反互

逆拓扑来提高常量时间内不可达查询的处理效率。本文还提出了一系列优化方法来减小索引规模、提高查询处理的效率。基于真实数据集的实验结果表明,本文算法具有较小的索引规模和更快的查询响应速度,同时具有较好的扩展性。最好情况下,在cit-Patents数据集上,与PLL算法相比,本文算法的索引规模和索引构建时间仅为PLL算法3%,查询时间仅为PLL算法的20%;和BFSI-B算法相比,本文算法的索引仅为BFSI-B算法索引规模、查询时间的一半。

参考文献 (References)

[1] ZHANG T, GAO Y, LI C, et al. Distributed reachability queries on massive graphs[C]// Proceedings of the 2019 International Conference on Database Systems for Advanced Applications, LNCS 11448. Cham: Springer, 2019: 406-410.

[2] SENGUPTA N, BAGCHI A, RAMANATH M, et al. Arrow: approximating reachability using random walks over Web-scale graphs[C]// Proceedings of the IEEE 35th International Conference on Data Engineering. Piscataway: IEEE, 2019:470-481.

[3] XIE X, YANG X, WANG X, et al. BFSI-B: an improved k-hop graph reachability queries for cyber-physical systems[J]. Information Fusion, 2017, 38:35-42.

[4] YANO Y, AKIBA T, IWATA Y, et al. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths[C]// Proceedings of the 22nd ACM International Conference on Information and Knowledge Management. New York: ACM, 2013: 1601-1606.

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

[6] CHENG J, SHANG Z, CHENG H, et al. K-reach: who is in your small world[J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1292-1303.

[7] CHENG J, KE Y, CHU S, et al. Efficient processing of distance queries in large graphs: a vertex cover approach[C]// Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2012: 457-468.

[8] 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. New York: ACM, 1989:253-262.

[9] AKIBA T, IWATA Y, YUICHI YOSHIDA Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling[C]// Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 349-360.

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

[11] YILDIRIM H, CHAOJI V, ZAKI M J. Grail: scalable reachability index for large graphs[J]. Proceedings of the VLDB Endowment, 2010, 3(1/2): 276-284.

[12] VELOSO R R, CERF L, MEIRA W, Jr, et al. Reachability queries in very large graphs: a fast refined online search approach[C]// Proceedings of the 17th International Conference on Extending DB Technology. Berling: Springer, 2014: 511-522.

[13] ZHOU J, ZHOU S, YU J X, et al. DAG reduction: fast answering reachability queries[C]// Proceedings of the 2017 ACM International Conference on Management of Data. New York: ACM, 2017: 375-390.

[14] BOLDI P, SANTINI M, VIGNA S. A large time-aware Web graph[J]. ACM SIGIR Forum, 2008, 42(2): 33-38.

[15] JIN R, RUAN N, DEY S, et al. Scarab: scaling reachability computation on large graphs[C]// Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2012: 169-180.

[16] van SCHAIK S, de MOOR O. A memory efficient reachability data structure through bit vector compression[C]// Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2011: 913-924.

DU Ming, born in 1975, Ph. D., associate professor. His research interests include information retrieval, data analysis and mining, natural language processing.

YANG Anping, born in 1994, M. S. candidate. His research interests include query processing and optimization of graph data.

ZHOU Junfeng, born in 1977, Ph. D., professor. His research interests include information retrieval, query processing and optimization of semi-structured data and graph data.

CHEN Ziyang, born in 1973, Ph. D., professor. His research interests include computation theory, query processing, optimization of graph data.

YANG Yun, born in 1995, M. S. candidate. Her research interests include query processing and optimization of graph data.