计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20131567
52(1):116-129,2015

大规模图数据可达性索引技术:现状与展望

富丽贞[1,2]　　孟小峰[2]
[1](中北大学软件学院　太原　030051)
[2](中国人民大学信息学院　北京　100872)
(fulizhen303@163.com)

Reachability Indexing for Large-Scale Graphs: Studies and Forecasts

Fu Lizhen[1,2] and Meng Xiaofeng[2]
[1](*School of Software, North University of China, Taiyuan* 030051)
[2](*School of Information, Renmin University of China, Beijing* 100872)

Abstract　With the rapid development of social networks, biology, ontology and other emerging areas, there are a lot of graph data in real applications. Reachability query is one of the fundamental queries in graphs. For small graphs, depth-first search (DFS) or reachability transitive closure can answer reachability query easily. However, as graphs become larger and larger, above two approaches are no longer feasible, because the query time of DFS is very long and the storage cost of reachability transitive closure is very high. Therefore, lots of reachability indices have been proposed. These approaches have been widely applied in many areas of computer science, such as software engineering, programming languages, distributed computing, social network analysis, biological network analysis, XML database, RDF databases, routing planning and other fields. In addition, reachability indices can also speed up other graph algorithms, such as the shortest path queries and sub-graph matching. In this survey, applications of reachability indexing schemes are introduced firstly. Secondly, according to supported graph size, supported graph type and supported query type, we provide taxonomy of existing works. Then, we provide a comparison of representative works. Finally, we discuss the disadvantages of existing scalable reachability indexing schemes and put forward some suggestions for future research works.

Key words　reachability; indexing; query processing; labeling; graph data

摘　要　随着社交网络、生物信息网、本体等新兴领域的飞速发展,在现实应用中涌现出大量的图数据.可达性查询是有向图上一类最基本的查询.当图的规模非常小时,利用深度优先遍历(depth-first search, DFS)或可达性传递闭包可以很容易处理可达性查询.但是,随着图的规模越变越大,由于DFS方法的查询效率太低而可达性传递闭包方法占用的存储空间太大,这2种方法不再适用.因此,许多可达性索引方法相继被提出.这些方法已经被广泛应用于多个计算机科学领域,如软件工程、编程语言、分布式计算、社交网络分析、生物网络分析、XML和RDF数据库、路由规划等领域.此外,可达性索引还可用于加速其他图算法,如最短路径查询和子图模式匹配.首先介绍了可达性索引的应用背景.接着,依据支持的数据规模、数据类型以及查询类别,将现有可达性索引工作进行了分类,并对代表性工作进行分类比较;最后,讨论了现有的大规模图数据可达性索引方法存在的问题,并指出了未来的研究方向.

收稿日期:2013-12-02;修回日期:2014-04-24
基金项目:国家自然科学基金项目(61379050, 91224008);国家"八六三"高技术研究发展计划基金项目(2013AA013204);高等学校博士学科点专项科研基金项目(20130004130001)

关键词 可达性;索引;查询处理;编码;图数据

中图法分类号 TP392

给定图上 2 个顶点 u 和 v,可达性查询(u?→v)回答这样一个问题:在有向图中是否存在一条从 u 到 v 的路径.可达性查询是有向图上一类最基本的查询.当图的规模非常小时,利用深度优先遍历(depth-first search, DFS)或可达性传递闭包很容易处理可达性查询.但是,随着现实应用中图的规模越来越大,因为 DFS 方法的查询效率太低而可达性传递闭包的空间代价太大,这 2 种方法不能满足需求.此时,可达性索引技术应运而生.它们已经广泛应用于多个计算机科学领域,如软件工程、社交网络、生物网络、编程语言、路由规划等领域.此外,可达性索引还能够加速图上其他算法,如最短路径和子图模式匹配.如 Dijkstra 算法利用可达性索引可以避免访问那些不能到达目标节点的节点,从而加速 Dijkstra 算法的执行.

现在我们已经进入大数据时代,数据增长的速度已经远远超出了人们的想象.近几年来,图的规模越来越大.例如,据文献[1]所述,Face Book 已经拥有 7.5 亿个用户,平均每个用户有 130 个朋友.这说明 Face Book 社交图有 7.5 亿个节点和大约 49 亿条边.在语义网中,已经出现元组数超过 10 亿的 RDF 图.随着实际应用中图的规模越变越大,图数据的索引和查询技术面临新的挑战.大规模图数据的可达性索引技术研究成为当前研究的热点.

1 应用场景

目前可达性索引已经被广泛应用于多个计算机科学领域,如软件工程、编程语言、分布式计算、社交网络分析、生物网络分析、XML[2]和 RDF[3]数据库、路由规划等领域.同时,可达性索引还能加速其他图算法[4-7].例如,Dijkstra[7]算法利用可达性索引避免遍历那些不能到达目标节点的点,进而加速 Dijkstra 算法.下面给出几个具体应用实例:

1)语义网.语义网的目标是为了让机器理解 Web 上信息.数据存储和转化的方式可以是 RDF 也可以是 XML.XML 文档的主体结构是树,如果存在 ID/IDF 关系,则需要表示为图.RDF 是由三元关系组成的,也需要表示为图.XQuery[8]和 SPARQL[9]分别是针对 XML 和 RDF 数据设计的查询语言.XQuery 是专门针对 XML 数据设计的查询语言.SPARQL 让用户使用类 SQL 语法来定义图模型.这 2 种查询语言都存在路径匹配,而可达性查询是路径匹配的核心操作.

2)本体.在知识表示系统中,本体定义了一个概念集合、概念属性以及在同一范畴中概念相互之间关系.本体利用这些来描述一个范畴或进行推理.在语义网中,推理引擎生成所有可能的结论并将这些推理得到的三元组加入到 RDF 数据集中.例如,如果类 C_1 是类 C_2 的子类,而 C_2 是类 C_3 的子类,则可以推出 C_1 是 C_3 的子类.可达性索引可以加速这个推理过程.

3)生物信息网.随着高吞吐量数据获取技术的发展,生物学家们收集到大量的异构图数据,如代谢路径数据、基因控制网络数据、信号传播网络数据等等.在这些数据集中,顶点代表蛋白质、基因或化合物,边表示它们之间的关系.例如,在基因控制网络中,查询"基因 A 是否直接或间接控制基因 B?"恰好对应于一个可达性查询.

4)社交网络.在社交网络中,每个用户用一个顶点表示.如果 2 个用户之间有联系,则用一条边将 2 个用户对应顶点连在一起.用户之间有多种关系,如父子关系、兄弟关系、姐妹关系以及雇用关系等等.用边上标签表示这些不同的关系.在社交网络中,许多查询都涉及到查询 2 个节点之间是否存在某种关系,这种查询实际上就是可达性查询.例如,我们想知道用户 A 是否是用户 B 的远亲.这时,我们需要查询在图中是否存在一条从 A 到 B 的路径且路径上每条边对应父母孩子关系、兄弟关系或姐妹关系.

5)复杂查询处理.可达性索引还能用于加速其他图上算法,如最短路径查询和子图匹配等等.例如,在文献[4]中,王宏志等人利用他们提出的 interval-based 的可达性索引方法来加速处理子图模式匹配.文献[5]和文献[6]利用可达性索引来快速处理最短路径查询.

2 可达性索引研究现状

迄今为止,为了加快有向图数据的可达性查询,大量的可达性索引方法相继被提出.这些方法旨在查询时间、索引规模以及构建时间 3 个方面取得平衡.同时这些索引方法在这 3 个方面的性能都位于

DFS 和可达性传递闭包 2 种方法之间. 本文从索引支持的数据规模、数据类型以及查询类别 3 个方面将现有可达性索引工作进行了归类. 数据规模分为 3 类:小规模数据(节点数在万级以下的图数据)、中等规模数据(节点数在百万级以下的图数据)和大规模数据(节点数在百万级及以上的图数据). 查询的类型分为无约束查询和带约束查询. 图数据类型分为静态数据(数据从不更新)和动态数据(数据经常会被更新). 目前,带约束的查询分为 2 类:边受限查询和距离受限查询. 依据支持最大图数据的规模,现有的可达性索引分为 3 类:小规模图数据的索引、中等规模图数据的索引以及大规模图数据的索引. 依据支持图数据的类型,可达性索引分为 2 类:动态索引(支持更新的动态索引)和静态索引(不支持更新的索引). 依据支持可达性查询的类别,现有可达性索引为 2 类:支持受限查询的索引和支持非受限查询的索引.

从以上 3 个不同角度,可以将可达性索引分为 12 类,如图 1 所示. 在图 1 中,每个菱形代表 1 种类别. 每个原点表示一个代表性可达性索引方法. 箭头表示当前可达性索引技术的研究趋势. 现有的图数据上可达性索引方法分布在其中 7 个类上,分别是:支持小规模图数据非受限查询的静态索引方法、支持小规模图数据非受限查询的动态索引方法、支持中等规模图数据非受限查询的静态索引方法、支持中等规模图数据受限查询的静态索引方法、支持中等规模图数据非受限查询的动态索引方法、支持大规模图数据非受限查询的静态索引方法、支持大规模图数据非受限查询的动态索引方法. 剩余 5 个类,目前尚没有相关工作,如空白的菱形所示.

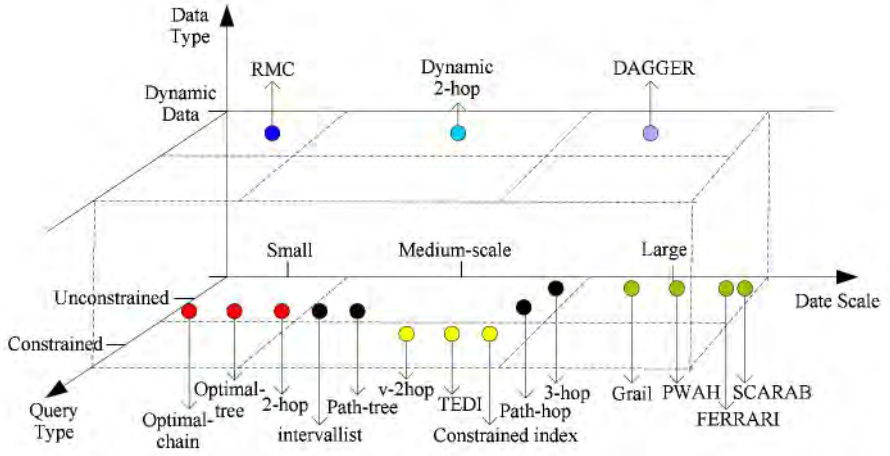


Fig. 1 Representative reachability labeling and trends.

图 1 可达性索引技术代表性成果及研究趋势

过去的工作主要针对中小规模图数据中不受限可达性查询处理问题. 近来,可达性索引研究热点集中在大规模图数据上可达性查询处理、动态图数据上可达性查询处理以及受限可达性查询处理. 下面,我们将详细介绍可达性索引的研究现状.

2.1 静态可达性索引

由于许多支持中等规模图数据索引方法都是由支持小规模图数据索引方法扩展而来,这些方法的实现原理相似. 同时,本文的重点在于大规模图数据索引研究. 因此,本文将以上两类方法合并在一起介绍.

现有的静态可达性索引分为 3 类:支持中小规模图数据非受限查询的静态索引方法、支持大规模图数据非受限查询的静态索引方法、支持中小型图数据受限查询的静态索引方法.

2.1.1 支持中小规模图数据非受限查询的静态索引

由于注重查询的效率以及索引规模的优化,虽然支持中小规模图数据非受限查询的静态索引方法的查询效率很高,索引的压缩率也高,但是它们索引构建十分复杂从而无法处理大规模图数据. 依据索引方法的实现原理,支持中小规模图数据非受限查询的静态索引方法可以分为 4 类:基于链的索引方法、基于树的索引方法、基于 Path-tree 的索引方法以及基于 hop 技术的索引方法. 下面我们将逐一介绍:

1) 基于链的索引方法

基于链的可达性索引方法基本思想:首先将一个有向无环图(directed acyclic graph, DAG)图划分为最少的不相交链,然后为图中每个节点分配一个链标号以及它在链中的序列号. 若 1 个节点能够到达另一条链中的其他节点,则只需记录它能到达

的序列号最小的节点以及其对应的链标号.给定2个节点A与B,想要知道在DAG图中A是否能到达B,只需检查A是否能到达B所对应的链以及所到节点的序列号是否小于B的序列号.如果答案为是,则A能够到达B,否则不能.假设节点个数为n且划分得到链个数为k,则该类方法的查询时间复杂度为$O(\log k)$,索引空间复杂度为$O(nk)$.Optimal-chain索引[10]和Cormen等人提出的方法[7]都属于这一类.其中,Optimal-chain索引是针对小规模图数据提出的,而文献[11]提出的索引方法可以用于中等规模图数据.文献[10]最先提出基于链的索引方法,但是计算最小链划分的时间复杂度为$O(n^3)$.例1给出一个Optimal-chain索引的例子.为了降低Optimal-chain索引创建时间复杂度,文献[11]提出了一个的$O(n^2+kn\sqrt{k})$算法.

例1. 如图2所示,Optimal-chain方法将图划分为3条链,得到的索引如图2(b)所示.图G上每个节点分配到一个编码,每个编码由链标号以及可达最小节点序号两部分组成.给定一个可达性查询v_0 ?→v_3,因为v_3在第2条链上且对应序号为3,v_0在第2条链可达最小节点序号为2且$2<3$,所以查询v_0 ?→v_3返回真,也就是v_0可以到达v_3.

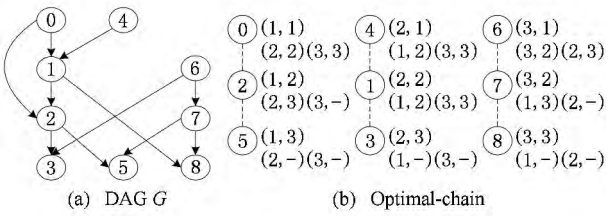


Fig. 2 Optimal-chain indexing.

图2 Optimal-chain索引

2) 基于树的索引方法

基于树的索引方法基本思想:首先找到一棵生成树并为每个节点分配一个区间编码,若节点A通过生成树可以到达节点B,则A对应的区间包含B对应的区间.接着,按照逆拓扑序,将节点对应的所有区间合并到其父节点中.最后每个节点对应一个区间集.在判断节点A是否可以到达节点B时,只需判断A的区间集合中是否存在一个区间包含B的第1个区间.若包含,则A能到达B;否则不能.

Optimal-tree[12], intervallist[13], interval-based[4]都属于这一类.Dual-Labeling[14], Label-SSPI[15], GRIPP[16]索引是扩展的Optimal-tree方法.其中,Optimal-tree和interval-based索引只能用于小规模图数据.Optimal-tree, intervallist, Dual-Labeling, Label-SSPI,GRIPP则可以用于中等规模图数据.

Optimal-tree索引方法是这一类中最具代表性工作.Optimal-tree方法从原图中找到的生成树是最优生成树,从而能够覆盖更多的可达性关系.例2给出一个Optimal-tree索引的例子.

例2. 给定一个DAG图,如图3(a)所示,Optimal-tree索引如图3(b)所示,其中实线部分对应的是最优生成树.给定一个可达性查询v_0 ?→v_5,因为v_0对应的区间编码([1,10])包含v_5的第1个区间([7, 8]),所以v_0 ?→v_5返回真.

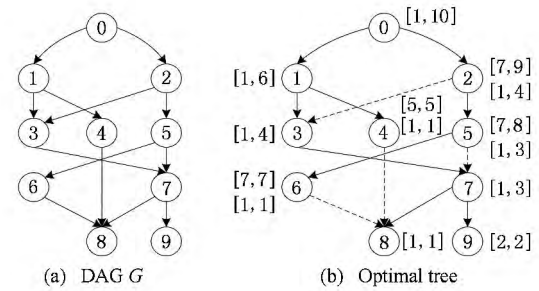


Fig. 3 Optimal-tree indexing.

图3 Optimal-tree索引

intervallist索引是一种基于区间的索引方法,实际上也是一种基于树的索引方法.但是,它所基于的生成树是通过深度优先遍历生成的.构建intervallist索引只需一次DFS遍历.所以,索引的构建效率比Optimal-tree方法高,有时能够处理百万级图数据.图4给出一个intervallist索引的例子.如图4所示,由实边组成的子图为DFS遍历得到的生成子树.interval-based方法是Optimal-tree方法的改进,该方法用一个整数ID来唯一标示1个节点,对应于Optimal-tree区间编码的第1值.该方法区间编码的生成过程与Optimal-tree相同.给定2个节点A和B,若B的ID属于A的区间集合的某个区间,则A能到达B;否则不能.该索引的构建效率不及intervallist方法.对于稀疏图而言,非树边

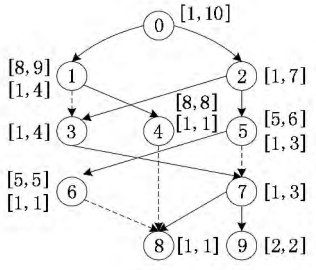


Fig. 4 intervallist indexing.

图4 intervallist索引

远远小于节点的个数. 为了提高稀疏图上基于树的索引方法的查询效率和降低索引的空间复杂度,文献[14]提出了 Dual-Labeling 索引. 该索引的空间复杂度降至 $O(n+t^2)$,同时查询时间复杂度降至 $O(1)$,其中 n 表示节点的个数,t 表示非树边个数. Label-SSPI 索引以及 GRIPP 索引的目标是最小化索引的构建时间和索引的规模. 它们的索引构建时间复杂度降低到 $O(m+n)$,索引的空间复杂度降低到 $O(m+n)$,但是却牺牲掉查询效率. 它们的最坏查询时间复杂度高达 $O(m-n)$. 这里,m 表示边数.

3) 基于 Path-tree 的索引方法

基于 Path-tree[17-18] 的索引方法基本思想:首先找到 1 个路径树生成子图(如图 5(b)所示),之后为每个节点分配 1 个三元组编码($I.begin$, $I.end$, X),利用编码能在常数时间内判断 2 个节点在路径树中是否可达. 最后为每个节点分配 1 个三元组编码集合 R^c,压缩记录所有通过不属于路径树的边可以到达的节点. 该方法可以处理中小规模图数据. 例 3 给出一个 Path-tree[17] 索引的例子. 对于非稠密图而言,该方法的索引查询效率非常高,但是索引的空间复杂度高达 $O(nk)$,其中 n 表示图中节点的个数,k 表示分解得到的路径个数.

例 3. 给定一个 DAG 图如图 5(a)所示,对应路径树如图 5(b)所示,对应的 Path-tree 索引如表 1 所示:

Table 1 Path-tree indexing

表 1 Path-tree 索引

Node	Label	R^c
1	(1,1,1)	(1,3,6), (2,2,4)
2	(1,3,3)	(1,4,5)
3	(1,1,2)	(1,3,6)
4	(1,3,6)	
5	(2,2,4)	(1,4,5)
6	(1,1,9)	(1,3,13)
7	(1,3,8)	
8	(2,2,7)	(1,3,11),(1,1,10)
9	(1,4,5)	
10	(1,3,11)	(1,1,10)
11	(1,3,13)	
12	(1,4,12)	
13	(1,1,10)	(1,3,13)
14	(1,1,14)	
15	(1,1,15)	

Path-tree 索引方法为每个节点分配一个三元组编码($I.begin$, $I.end$, X),编码前两位是对应路径的区间编码,最后一位对应节点在路径树(path tree)中序号. 给定两个编码 $L_1(a,b,c)$ 和 $L_2(d,e,f)$,若 $a<d<b$ 且 $c<f$,则称 L_2 被 L_1 包含. 同时,在路径树上 L_1 对应的节点可以到达 L_2 对应的节点. 给定两点 u 和 v,若 u 的三元组编码被 v 的 R^c 中一个三元组编码包含,则 v 可以到达 u. 如表 1 所示,v_8 的 R^c 中一个编码包含 v_{14} 对应的编码,所以可以判定 v_8 可以到达 v_{14}.

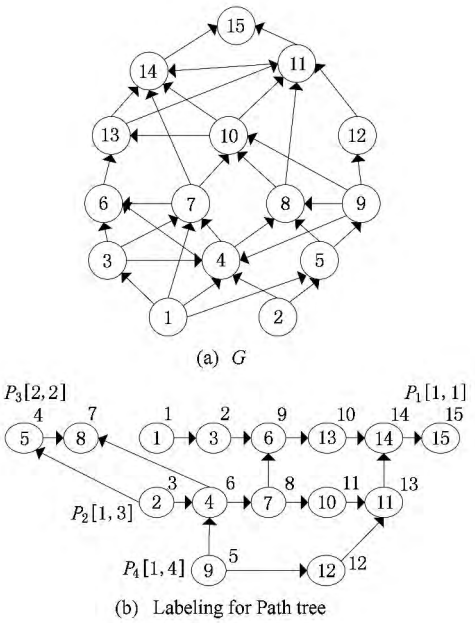


Fig. 5 Spanning subgraph; Path tree.

图 5 Path tree 生成子图

4) 基于 hop 技术的索引方法

基于 hop 技术的索引方法基本思想:为每个节点分配 in 和 out 两个编码,利用这些编码可以通过 2 跳或 3 跳来验证 2 点之间是否可达. 这类方法主要包括 2-hop 索引[19]、3-hop 索引[20] 以及 Path-hop 索引[21]. 其中,2-hop 只能用于处理小规模图数据,而 3-hop 和 Path-hop 索引可以用于处理中小规模图数据. 2-hop 方法通过 2 跳来处理查询,即先从源点跳到中间节点,再跳到目标节点. 3-hop 和 Path-hop 通过 3 跳来处理查询,即先从源点跳到链或树的入口,再从入口跳到链或树的出口,最后从出口跳到目标节点. 如图 6 所示. 其中,2-hop 索引和 3-hop 索引是最具代表性的 2 个工作.

2-hop 是最先提出的基于 hop 技术的索引方法. 它为 DAG 图上每个节点分配 2 个编码 in 和 out,其中 in 和 out 为 2 个整数集合. 给定 2 个节点

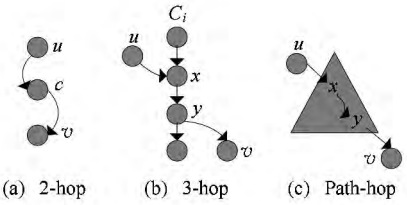
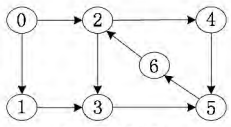


Fig. 6 hop indexing scheme.

图 6 hop 索引机制

v_1 和 v_2,若 $v_1.out$ 和 $v_2.in$ 有交集,则 v_1 可以到达 v_2. 图 7 给出一个 2-hop 索引的例子. 其中,图 7(a)为对应 DAG 图,图 7(b)为对应的编码. 2-hop 的索引构建时间复杂度为 $O(n^4)$. 文献[22-24]提出的索引方法是 2-hop 的变种. 尽管这些方法作了许多优化但是仍不能处理超过 100 万个节点的有向图.



(a) DAG G

Node	0	1	2	3	4	5	6
in			5	2,5	2,5	2,3	3,5
out	1,2,3	1,2,3	3,5	6,5	5	2,4,5	2

(b) 2-hop

Fig. 7 2-hop indexing.

图 7 2-hop 索引

为了处理稠密图数据,Jin 等人在文献[20]中提出了 3-hop 索引,该索引利用链分解作为中间结构来改进 2-hop 方法. 与 2-hop 类似,3-hop 索引为每个节点分配 2 个编码 in 和 out. 但是,查询处理方法与 2-hop 方法有所不同. 给定 2 个节点 v_1 和 v_2,若 $\exists X\in v_1.out$, $\exists Y\in v_2.in$, X 与 Y 在同一条链上且 $X<Y$,则 v_1 可以到达 v_2. 图 8 给出一个基于一条链的 3-hop 索引实例. 从图 8 可以看出,与 2-hop 相比,3-hop 规模更小. 而 Path-hop 索引是 3-hop 方法的扩展,它用树来代替链分解作为中间结构.

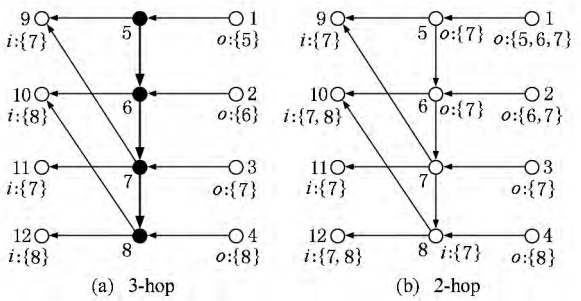


Fig. 8 3-hop indexing.

图 8 3-hop 索引

求最小基于 hop 的索引的问题可以转换成集合覆盖问题. 因此,该类索引方法的构建代价往往很高.

2.1.2 支持大规模图数据非受限查询的静态索引

现在的时代已经进入大数据时代,数据增长的速度已经远远超出了人们的想象. 而以前的可达性索引工作在处理大规模图数据时纷纷遇到瓶颈. 为了克服这个瓶颈,近年来先后提出一些针对大规模图数据的可达性查询处理方法,如基于 Map-reduce 的方法[25]和基于可达性索引的方法. 前一种方法是针对分布式数据提出的,该方法的查询代价还包括传输代价. 这里我们只讨论基于可达性索引的处理方法. 支持大规模图数据的可达性索引方法都是通过牺牲查询效率来换取索引的可用性. 这些方法有一个共同特点:对于某些查询会很快给出答案,而另外一些可能花费很长时间才能给出答案. 随着图的规模越来越大,这个问题会越来越显著. 如 Grail 索引方法,对于某些否定查询,它只需检查 2 个节点对应的编码就能给出答案,也就是在常数时间内可以回答查询. 而对于肯定查询则需要遍历原图,直到在原图中找到一条路径. 根据实现原理的不同,可以分为两类:基于压缩技术的索引方法和精炼在线索引方法:

1) 基于压缩技术索引方法

基于压缩技术的索引方法主要思想:利用位压缩技术压缩存储表示节点的可达传递闭包对应的位向量. 在判断节点间的可达性时,只需检查源节点的索引值在目标节点对应位置上是否为 1. 若为 1 则可达;否则不可达. 目前,该类方法只包括 PWAH[26]索引.

PWAH 索引是 1 种基于划分的词对齐混合压缩技术的索引方法. 该方法大大降低了索引的空间代价,从而可以处理大规模图数据. PWAH 支持 2,4,8 这 3 种划分. 图 9 给出一个 4 划分 PWAH 索引实例,对应 PWAH-4 编码用 16 B 压缩表示了 1 410 b 的位向量. 该方法首先将节点对应的传递闭包位向量划分为 15 b 的块. 共 3 种类型块:连续 1 块、连续 0 块和混合块. 然后,把 64 b 划分成 5 个

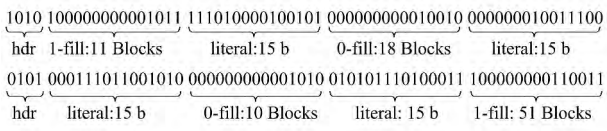


Fig. 9 PWAH-4 label.

图 9 PWAH-4 编码

部分来表示数据，hdr 为头部占 4 b，剩下 4 个部分用来表示数据. 1-fill 部分表示连续 1 块数量. 0-fill 部分表示连续 0 块数量，literal 部分表示混合块. 在 hdr 中，1 表示对应部分为 1-fill 或 0-fill(对应部分的首位是 1 则为 1-fill，0 则为 0-fill)，0 表示对应部分为 literal.

2) 精炼在线索引方法

精炼在线索引是一种辅助索引. 该类索引辅助 DFS 方法来完成可达性查询处理. 在搜索过程中，利用精炼在线索引对搜索空间进行剪枝，从而加快搜索过程. 代表性工作有 Grial[27-28]，SCARAB[29]，FERRARI[30].

Grail 方法多次随机深度优先遍历原图为每个节点分配 k 个区间. 在一次深度遍历，每个节点对应一个遍历序号. Grail 为每个节点分配一个区间 (min, max). 其中，min 表示节点在图中能到达的节点的最小的遍历序号. max 表示节点在图中能到达的节点的最大的遍历序号. 给定 2 个节点 v_1 和 v_2，对应区间分别为 Iv_1 和 Iv_2，Grail 方法能够保证：若 $Iv_1 \not\subset Iv_2$，则 v_2 一定不能到达 v_1. 但是，若 $Iv_1 \subset Iv_2$，则不能保证 v_2 可以到达 v_1. 给定一个查询 $v_1 ?\rightarrow v_2$，若 v_2 的区间编码中有一区间不包含于 v_2 的对应区间，则 v_1 不能到达 v_2. 否则，必须继续在线遍历原图，判断 v_1 的孩子节点是否能到达 v_2，直到确定不可达或遇到 v_2. 图 10 给出一个 2-Grail 索引的例子，其中每个节点的编码包含 2 个区间.

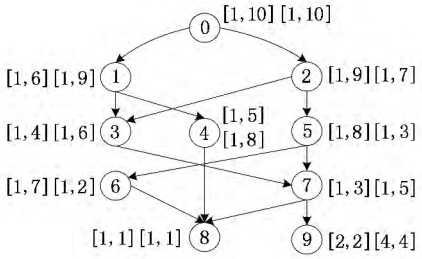


Fig. 10 Grail indexing.

图 10 Grail 索引

从图 10 可以看出，Grail 索引对于否定查询非常有利，但是对于肯定查询，查询效率却不及 DFS 方法. 为了提高肯定查询的效率，在 2013 年，文献[30]提出了 FERRARI 索引. 与 Grail 方法不同之处是，FERRARI 为每个节点分配 2 类区间：一种是精确的区间；另一种是近似区间. 利用精确区间，我们很快作出肯定回答，利用近似区间，我们可以很快作出否定回答. 如图 11 所示，I_i 表示精确区间，r_i 表示区间间隔. FERRARI 方法从中选出满足式(1)的 k 个最优区间. 在式(1)中，C 中区间一部分是精确区间 I_i，一部分由 I_i 和 r_j 合并得到的近似区间. 若 I 为精确区间，则 η_I 为 1，否则 η_I 为 0. $|I|$ 表示区间的长度.

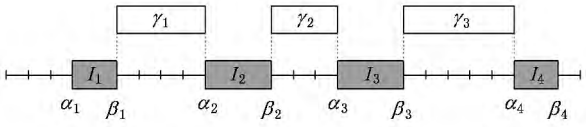
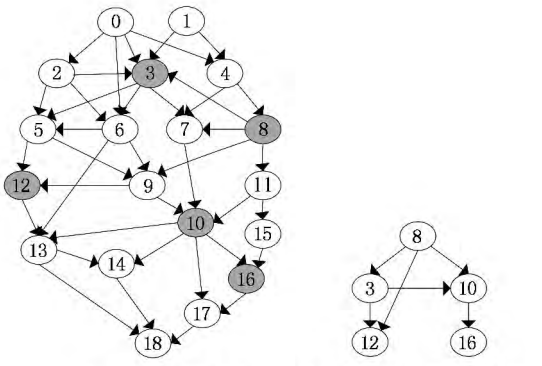


Fig. 11 The design principle of FERRARI.

图 11 FERRARI 索引设计原理

$$L_k^* = \arg\min_{C: L \subseteq_k C} \sum_{I \in C} (1 - \eta_I) |I|. \quad (1)$$

最新工作 SCARAB 则采用不同策略. 该方法的扩展性优于其他方法，即它可以处理更大规模的图数据. SCARAB 首先找到一个可达主干图 G^*，它能保证：给定任意 2 个节点 u_1 和 u_2 且从 u_1 到 u_2 的最短路径边数大于 k，G^* 上一定存在满足以下 2 个条件的点 v_1 与 v_2：1) 在 G 上，u_1 到 v_1 的最短路径边数小于 k，v_2 到 u_2 最短路径边数小于 k；2)在 G^* 中，$v_1 \rightarrow v_2$. 然后，在规模很小 G^* 上建立索引. 如图 12 所示，与原图 G 相比，对应的主干图 G^* 的节点数大幅度减少. 其中，k 为 2. 给定查询 $v_0 ?\rightarrow u_0$，SCARAB 的查询处理过程如下：首先，从 v_0/u_0 出发，向前/向后 BFS 遍历原图 l 步，$l \leqslant k$. 若在此过程中遇到 u_0/v_0，则返回真. 否则，将在向前遍历过程中遇到的所有主干图的节点放入集合 S_1 中，将在向后遍历过程的遇到的所有主干图的节点放入集合 S_2 中. 最后，利用在 G^* 建立的可达性索引判断 S_1 中是否存在一个节点可以到达 S_2 中一个节点. 若存在，则返回真，否则返回假.



(a) Original graph G (b) G's reachability backbone

Fig. 12 An example of reachability backbone.

图 12 可达主干图实例

2.1.3 支持中小型图数据受限查询的静态索引

上述所有工作都是针对一般的可达性计算问题.

它们要回答的问题是在图上是否存在一条从节点 A 到节点 B 的路径,也就是说,节点 A 是否能到达节点 B. 但是,在有些实际应用中,我们需要解决的问题是:从 A 到 B 是否存在一条满足某些条件的路径. 目前相关工作主要涉及到 2 种类型可达性查询:一类是距离受限的可达性查询;另一类是边的类型受限的可达性查询. 文献[5-6]属于第 1 类,文献[31]属于第 2 类.

1) 支持距离受限查询的索引方法

虽然 Dijkstra 算法与 Bellman-Ford 的 All Pairs Shortest Paths 算法可以很好地处理最短距离查询,但是由于 Dijkstra 算法的查询效率很低,而 All Pairs Shortest Paths 算法的空间代价很大,这些方法不能适合大规模图上最短路径查询. 为了加速最短路径查询,文献[5]提出一种 v-2hop 索引方法来解决距离受限可达性查询. 文献[6]提出一个基于树分解方法,称作 TEDI. 该方法在最短路径查询时间、索引规模以及索引构建时间 3 个方面都有一个数量级的提升.

2) 支持边受限查询的索引方法

许多真实的图数据集会为每条边设置一个标签,如在语义网以及生物信息网. 在这些场景中,用户提交的可达性查询可能对边上标签有约束,要求所有的标签都属于某个标签集合. 令人惊讶的是目前利用索引机制来解决这个问题的工作很少.

最近,Jin 等人在文献[31]通过扩展 BFS/DFS 以及 Full Transitive Closure 算法来处理边受限可达性查询. 此外,为了降低空间消耗以及提高查询效率,Jin 等人还提出一种基于树的索引方法,记作 Constrained Index.

2.2 动态可达性索引

目前,动态可达性索引的相关工作较少. 由于图数据结构复杂,动态维护可达性索引变得十分复杂. 早期关于动态可达性索引工作都是集中在理论分析上,包括文献[32-43]. 现有的动态态可达性索引分为 2 类:支持中小型图数据非受限查询的动态索引、支持大规模图数据非受限查询的动态索引.

2.2.1 支持中小型图数据非受限查询的动态索引

只支持插入操作的算法称为 Incremental 算法,只支持删除操作的算法称为 Decremental 算法,2 种操作都支持的算法称为 Fully Dynamic 算法. 在 1990 年以前的工作不是 Incremental 算法就是 Decremental 算法,他们的更新代价为 $O(n^2)$. 在这里,我们只讨论 Fully Dynamic 算法.

第 1 个 Fully Dynamic 算法是由文献[36]提出的,记作 RMC. 该方法的分摊更新时间复杂度为 $O(nm^{0.58}\log^2 n)$,分摊查询时间复杂度为 $O(n/\log n)$. 否定回答的错误概率为 $O(1/n^c)$,最坏情况下更新所需空间代价为 $O(n^2)$. 其中,n 表示节点个数,m 表示整个更新过程中 DAG 图包含的平均边数,c 为用户定义的常数.

RMC 方法是一种蒙特卡罗方法. 该方法的基本思想是随机选择一些特殊节点,并动态维护这些节点的可达传递闭包. 给定查询 $u \overset{?}{\rightarrow} v$,若存在一条从 u 到 v 的边,则返回真. 否则,检查是否存在一个特殊点 s,其中 u 能到达 s 且 s 能够到达 v. 若存在则返回真. 若以上两种情况都不满足,则返回否定回答. 文献[36]针对动态图数据上可达性查询处理只作了理论分析没有实现部分. 实际上,由于该方法所用到数据结构十分复杂,该方法很难实现. 因此,我们在此不作太多介绍.

文献[41]进一步将查询时间复杂度降低到 $O(1)$,更新时间复杂度降低到 $O(n^{2.2.6})$. 文献[35]再次将更新时间负责度降低到 $O(n^2)$,而对于只有删除操作的情形,分摊更新时间复杂度可以降低到 $O(n)$. Demetrescu 和 Italiano[36] 提出一个复杂度为 $O(n^{1.75})$的更新算法. 该方法对应的查询复杂度增加到 $O(n^{0.575})$. 算法将动态可达性问题归约到 0/1 矩阵上的多项式重估问题. 文献[32]提出一个新的 Fully Dynamic 算法. 其分摊更新代价为 $O(m+n\log n)$,最坏查询时间复杂度为 $O(n)$. 因为该方法需要动态维护强联通子图,其索引空间开销为 $O(m+n)$. 文献[33]针对 DAG 图提出了高效的更新算法. 在最坏情况下,在 $O(n^2)$时间内可以完成更新. 同时,在常数时间内可以处理查询. 该算法需要维护两点之间的路径集合,因此,其空间代价为 $O(n^2)$. 该方法还支持在同一节点上同时插入或删除多条边的情况.

以上几种方法都是动态维护节点间可达性传递闭包的方法,只能用于小规模图数据上可达性查询处理.

2010 年,文献[44]针对 DAG 图提出一种动态 2-hop 索引方法,记作 Dynamic 2-hop. 其目标旨在减少更新代价而不是最小化索引的规模. 所以,它比静态 2-hop 索引的规模大很多. Dynamic 2-hop 方法可以适用于中小型图数据.

Dynamic 2-hop 索引的基本思想是:找到一个具有节点分割属性的 2-hop 索引,从而提高索引更新

效率. 节点集 S 可以分割可达关系(u,v)当且仅当只有从图中移除所有 S 中节点,在 DAG 图上,u 才不能到达 v. 2-hop 索引具有节点分割属性当且仅当对于任意可达关系(u,v),$u.out \cap v.in$ 可以分割可达关系(u,v). 图 13 给出一个具有节点分割属性的 2-hop 索引实例:

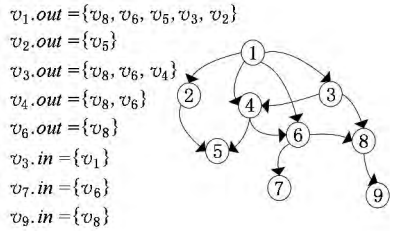


Fig. 13 An example of Dynamic 2-hop indexing.

图 13 Dynamic 2-hop 索引实例

该索引支持点的删除以及边的插入操作. 当然边删除操作和点插入操作可以转换为以上 2 种操作处理. 处理完这些操作之后,得到的 2-hop 方法仍然具有节点分割属性.

Dynamic 2-hop 方法处理点删除操作过程如下:

1) 令 X 为删除节点,从所有节点的 in 和 out 编码中删除 X;

2) 计算 X 的所有祖先节点 $A(X)$和所有后代节点 $D(X)$,按照第 1 个节点拓扑序降序和第 2 个节点拓扑序升序对 E_3 的元素进行排序,其中 $E_3=\{(a,d_a)\cup(a_d,d)\mid a\in A(X),d_a\in X.out\cap d(X)\}$;

3) 令$(a,\ d)$为 E_3 中一个元素,若 $d\in a.out$ 且 a 的所有孩子都不能到达 d(删除节点 X 后),则从 $a.out$ 删除 d. 若 $a\in d.in$ 且 a 不能到达 d 的所有父节点(删除节点 X 后),则从 $d.in$ 删除 a;

4) 按顺序处理 E_3 中下一个元素,执行步骤 3)直到处理完所有元素.

下面给出一个处理点删除操作的实例,如例 4 所示.

例 4. 给定一个图 G 和 Dynamic 2-hop 索引如图 13 所示,删除节点 v_6. 首先,从 $v_1.out$,$v_3.out$ 和 $v_4.out$ 中移除 v_6. 接着计算 $A(v_6)$,$D(v_6)$和 E_3,并按照步骤 2)规则排序. 其中,$A(v_6)=\{v_1,v_3,v_4\}$,$D(v_6)=\{v_7,v_8,v_9\}$,$E_3=\{(v_4,v_8),(v_3,v_8),(v_1,v_8)\}$. 最后,依序按照步骤 3)处理 E_3 中所有节点对:①处理(v_4,v_8). 因为满足步骤 3)中条件,所以从 $v_4.out$ 中移除 v_8. ②处理(v_3,v_8). 因为不满足步骤 3)中条件,所以保留 $v_3.out$ 中 v_8. ③处理(v_1,v_8). 因为不满足步骤 3)中条件,所以保留 $v_1.out$ 中 v_8.

Dynamic 2-hop 方法处理边插入操作过程如下:

1) 令$(x,\ y)$为插入边,若 x 为原图上点,y 为新插入点,则 $y.in=x.in\cup\{x\}$;

2) 若 x 为新插入的点,y 为原图上点,则 $x.out=y.out\cup\{y\}$;

3) 若 x 和 y 均为原图上点,则作如下处理:

①计算 x 的所有祖先节点 $A(x)$和 y 的所有后代节点 $D(y)$,$c_x=|\{a\mid x\notin a.out,\ a\in A(x)\cup\{d\mid x\notin d.in,d\in D(y)\}|$和 $c_y=|\{d\mid y\notin d.out,a\in A(x)\cup\{d\mid y\notin d.in,d\in D(y)\}|$. 若 $c_x<c_y$,则 $t=x$,否则,$t=y$;

② 对于 $A(x)$中节点 a,$a.out=a.out\cup\{t\}$;对于 $D(y)$中节点 d,$d.in=d.in\cup\{t\}$.

与删除操作相比可以看出插入操作非常容易处理.

文献[45]提出一种动态索引方法,但其更新处理效率远不及 Dynamic 2-hop 方法,这里我们不再详细讨论.

2.2.2 支持大规模图数据非受限查询动态索引

目前针对大规模动态图数据上的可达性查询处理工作非常少. 2013 年,Yildirim 等人提出了一种针对大规模图数据的轻量级动态索引方法,称作 DAGGER[46],它是目前唯一的大规模图数据上动态可达性索引工作. DAGGER 可以处理带环的有向图,其更新复杂度以及索引的空间复杂度均为 $O(n)$.

DAGGER 索引本质上是一种动态化的 Grail 索引. 在构建 DAGGER 过程时,首先将强连通子图(SCC)合并,然后构建区间编码. DAGGER 索引支持边插入操作和边删除操作. 在处理插入边(u,v)时,若形成新的 SCC c,则合并这些节点,再从 c 开始自底向上更新 DAGGER 索引. 否则,从 u 开始自底向上更新 DAGGER 索引. 同样,在处理删除边(u,v)时,若 SCC c 被分裂,则形成一条从 s 到 t 路径. 然后,从 t 开始自底向上更新 DAGGER 索引. 否则,从 u 开始自底向上更新 DAGGER 索引. 例 5 给出一个 DAGGER 实例以及处理边插入和删除的实例.

例 5. 给定一个有向图 G 如图 14(a)所示,其中圆圈中节点构成一个 SCC,合并所有 SCC 后得到的 DAG 图以及 DAGGER 索引$(k=1)$如图 14(b)所示. 插入边(T,A)后,生成由图 14 中双圈节点构成

新的 SCC. 合并这些节点到节点 v_3,而 v_3 的区间编码是 v_3 中节点的区间编码并集. 然后,从 v_3 出发由底向上更新 DAGGER 索引,即更新节点 v_K, v_J, v_2 和 v_R. 最终,更新后的 DAGGER 索引如图 15(a)所示. 在图 15(a)中图删除边 (L,P) 后,图 15(a)中 SCC v_3 被分裂成 3 个节点,如图 15(b)中双圈节点所示,形成一条从 v_3 到 v_L 的路径. 然后,从 v_L 出发自底向上更新索引,即依次更新 v_L, v_H 和 v_3 的区间编码. 最终,更新后的 DAGGER 索引如图 15(b)所示.

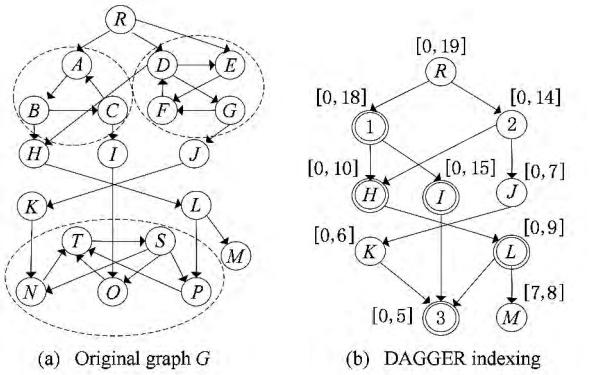


Fig. 14 An example of DAGGER indexing.

图 14 DAGGER 索引实例

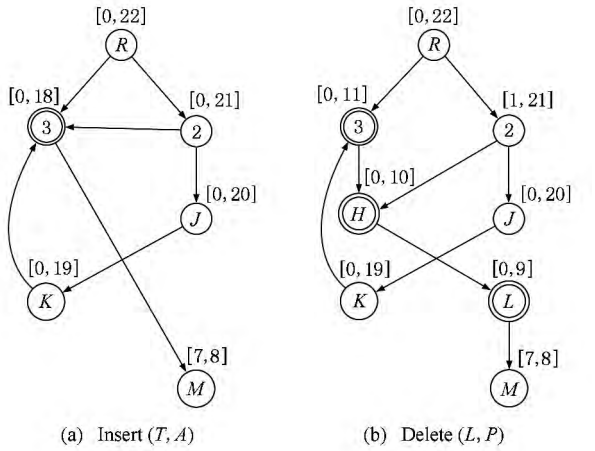


Fig. 15 An example of DAGGER updating.

图 15 DAGGER 索引更新实例

我们知道带有 ID/IDF 的 XML 数据是一种有序有向图,其主体结构是一个树. 因此,在更新 XML 数据时要维护这个结构. 但是,以上所述动态索引方法都是针对一般的图数据,不能有效地处理 XML 图数据. 虽然现在已经存在许多动态 XML 可达性索引方法[47-52],但是,这些方法都是针对树结构 XML 数据,不能用于图结构 XML 数据.

3 分类比较

下面我们针对已有的代表性可达性索引方法从复杂度上进行分类比较.

3.1 评价指标

如果设计索引的目标是加快查询处理,则查询效率是最重要的指标. 然而,在有些应用中索引的构建效率、索引的规模也是很重要的指标. 对于动态索引技术,索引的更新效率也是十分重要的指标. 那么,一个好的索引应该依据图的结构和类型以及应用的类型在这几个指标中取得平衡. 现有的大多数工作只关注查询效率和索引的规模而忽略索引构建效率. 主要原因是索引的构建效率对用户影响并不大. 但是,当图的规模越来越大时索引构建效率指标变得至关重要.

依据实际应用的需求,可达性索引的性能评价指标包含以下 4 个:

1) 查询效率——查询处理的时间;

2) 索引的规模——索引的空间代价;

3) 索引构建效率——构建索引所需时间;

4) 索引更新效率——索引更新时间,只针对动态可达性索引技术.

3.2 静态索引方法

我们从索引构建时间、查询时间和索引规模 3 个方面分析了支持中小型图数据可达性索引和支持大规模图数据可达性索引两类索引中代表性工作的复杂度,如表 2 所示,其中 k 表示链或路径的数量,k'表示用户指定常数,t 表示剩余边数,d 表示节点的平均出度,max 表示节点对应最大区间个数,avg 表示平均区间数,n 表示节点个数,n^* 表示主干图中节点个数,m 表示边数,$|Con(G)|$ 表示概要节点的个数,/ 表示不确定. 因为 SCARAB 索引规模与选取的内部索引机制有关.

从表 2 可以看出,在第 1 类中,GRIPP 的索引构建时间复杂度最小,Dual-Labeling 的查询时间复杂度最小,但是 Dual-Labeling 只适用于稀疏图. 对于一般的图而言,Intervallist 的查询效率最高. Path-tree 和 Optimal-chain 的索引规模最小. 在第 2 类中,Grail 的索引构建时间最少,Grail 和 FERRARI 的索引规模最小. SCARAB 索引的查询效率最高. 第 3 类支持受限查询索引方法,由于支持查询类别不同,所以不具备可比性.

Table 2 Comparison of Dynamic Indexing Schemes

表 2 静态索引性能对比

Category	Indexing Scheme	Construction Time	Query Time	Index Size
Small/Medium Scale	Optimal-chain[10]	$O(n^3)$	$O(\log k)$	$O(kn)$
	Optimal-tree[12]	$O(nm)$	$O(\log n)$	$O(n^2)$
	GRIPP[16]	$O(n+m)$	$O(m-n)$	$O(m+n)$
	Dual-Labeling[14]	$O(m+n+t^3)$	$O(1)$	$O(n+t^2)$
	intervallist[13]	$O(m\times max)$	$O(\log max)$	$O(n^2)$
	Path-tree[17]	$O(mk)$	$O(\log^2 k)$	$O(nk)$
	2-hop[19]	$O(n^4)$	$O(\sqrt{m})$	$O(n\sqrt{m})$
	3-hop[20]	$O(kn^2\lvert Con(G)\rvert)$	$O(\log n+k)$	$O(n\sqrt{m})$
Large Scale	Grail[27]	$O(k'(n+m))$	$O(n)$	$O(k'n)$
	PWAH[26]	$O(n^3)$	$O(n)$	$O(n^2)$
	SCARAB[15]	$O(n\varepsilon\log dd^{\varepsilon})$	$O(d^{\lceil\varepsilon\rceil}+d^{2\varepsilon}\log n^*)$	/
	FERRARI[30]	$O(m+nk'\log dk')$	$O(n)$	$O(k'n)$

3.3 动态索引方法

我们从索引构建时间、查询时间、更新时间和索引规模 4 个方面分析了动态可达性索引中代表性工作的复杂度,如表 3 所示,其中 m 表示边数,n 表节点个数. 从表 3 可以看出,RMC 的查询时间最少. DAGGER 在索引构建时间、更新时间以及索引规模 3 个指标上性能都最好. 虽然 RMC 的查询时间最少,但是索引更新代价、索引存储代价都很大,索引结构复杂,显然不能适用于大规模图数据.

Table 3 Comparison of Dynamic Indexing Schemes

表 3 动态索引性能对比

Indexing Scheme	Construction Time	QueryTime	Update Time	Index Size
RMC[34]	$O(mn)$	$O(1)$	$O(n^2)$	$O(n^2)$
Dynamic 2-hop[44]	$O(n^2(n+m))$	$O(\sqrt{m})$	$O(mn)/O(n)$	$>O(n\sqrt{m})$
DAGGER[46]	$O(n+m)$	$O(n)$	$O(n)$	$O(n)$

4 未来工作的展望

如今我们已经进入大数据时代,数据增长的速度已经远远超出了人们的想象. 尽管已有一些有效的支持大规模数据的可达性索引方法,但是仍存在一些没有解决的问题. 大规模数据上可达性查询处理将仍然是研究的热点. 结合存在的亟待解决的问题,我们认为以下几个方面是未来的研究方向.

4.1 大规模图数据热点查询处理

随着社交网络、语义网、XML 以及生物信息网等新兴领域的迅速发展,大规模图数据集不断涌现. 随着实际应用中图的规模越来越大,图数据索引和查询技术面临新的挑战. 在处理大规模图数据时,大部分已有可达性索引方法纷纷遇到瓶颈.

近年来, Yildirim 等人[27]、Seufert 等人[30]、Schaik 等人[26]、Jin 等人[29]先后提出一些有效的方法来克服这一瓶颈问题. 这些方法都是通过牺牲查询效率来换取索引的可扩展性. 然而,在实际应用查询负载中,往往有些节点经常被查询,而有些节点则很少或从来不会被查询. 例如,在论文引用数据集 Citeseer 中,用户经常会查询作者 A 是否受论文 B 的影响. 在此过程中,那些引用率很高的文章会频繁地被用户查询,而那些引用率低的或从没被引用过的文章往往无人问津. 在社交网络中,每个顶点表示一个用户. 若两个用户之间有联系,则用边相连,如亲属关系、朋友关系以及雇用关系等. 在许多实际应用中,用户常常会查询 A 是否和 B 有关系. 人们往往会关心与名人之间的关系. 对于一些无名小卒人们往往不予关心. 这里,我们把经常出现在查询当中的节点称作热点,而包含热点的查询称作热点查询. 对于热点查询,我们希望能很快返回查询结果,从而能够提高系统整体的查询性能,同时也能够即时响应用户的查询请求.

然而,已有的大规模图数据可达性索引方法不能针对热点进行优化.此外,现有方法的查询处理效率差异很大,对于有些查询可以很快返回结果,对于有些查询则需花费很长时间.以最新索引方法SCARAB为例,对于有些查询,在DFS遍历原图时就能返回结果;而对于有些查询,则需DFS遍历原图找到所有的主干节点,并在这些得到的点上进行$|S_1|\times|S_2|$次可达性查询才能返回结果.其中S_1表示所有查询源节点可以到达主干点集合,S_2表示所有可以到达目标节点的主干节点集合.随着图的规模越来越大该问题也越来越严重.因此,很可能出现这样的情况:热点查询的效率很低,而不是热点查询的查询效率却很高.

这里,如何依据热点的权重(该权重与热点出现频率相关)来设计可达性索引从而有效地支持热点查询是一个极具挑战性的问题.

4.2 大规模动态XML图数据上查询处理

随着XML作为一种通用的数据交换格式被广泛使用(特别是在万维网中获得空前成功),XML文档规模也越来越大.XML文档的管理和查询给研究学者带来新的挑战.已有大多数XML研究工作都将XML文档作为一棵树来处理.然而,在许多应用中,将XML文档作为有向图来处理更自然更合理.例如,文献出版XML文档中的章节和作者的关系,因为一个作者可以写多个章节,一个章节可能由多个作者完成,所以该XML文档更适合用图结构来描述.当然,图结构XML文档可以用树状模型来表示,通过多次复制元素节点来实现.但是,这将产生大量的冗余.例如,图16给出一个书籍XML文档片段对应的有向图,其中虚线表示ID/IDREF关系,实线表示元素包含关系.图16所示的书目信息如果用一个树形结构的XML表示,元素"作者"将会被多次复制.XML标准使用ID和IDREF类型以避免冗余.其中,ID表示文档元素唯一识辨名称,该属性唯一标识XML文档元素.IDREF类型容许在文档任何地方引用对应ID等于IDREF的元素.如果考虑ID/IDREF属性,XML数据应该用图模型来表示.不同于其他图数据(如生物网络、社会网络等),XML数据主体结构是一棵树且节点间是有序的,即文档序.因此,XML数据应该表示成为一个有序的有向图.为了提高XML数据的查询效率(例如XQuery查询),在XML数据库一个常用的方法是建立结构索引(即XML编码技术[45-50]).对于静态XML图数据,很容易扩展现有的针对树状XML数据的编码方法来处理.然而,当XML经常被更新时,维护索引变得十分费时.因为我们不仅需要维护文档的顺序,同时还需要维护节点间可达性关系.虽然已经存在一些XML动态编码技术,但是这些方法是树结构XML数据不能直接应用于XML图数据.

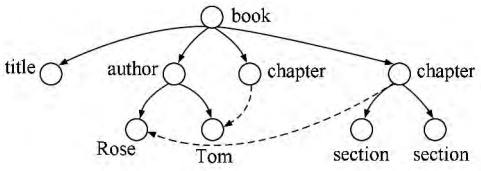


Fig. 16 An XML document with ID/IDREF.

图16 带有ID/IDREF的XML文档

这里,由于图数据结构复杂,如何设计能够适用于大规模XML图数据和易更新的可达性索引成为另一个极具挑战性问题.

4.3 大规模图数据受限查询处理

尽管图的规模越来越大,但是对于某些用户而言他们只关心其中一部分节点之间的可达性关系.例如,在生物信息学中,研究鸟类的学者只关心与鸟类相关的基因、蛋白质等之间的关系.研究哺乳类动物的学者只关心与哺乳类动物相关的基因、蛋白质等之间的关系.在文章引用数据库中,研究数据库的学者只关心与数据库相关的作者文章之间的关系,并不关心信息安全和人工智能等其他方面的文章和作者之间的关系.此外,在社交网络中,为了保护用户隐私,查询引擎只能提供一部分节点之间可达性关系查询功能.

近几年来尽管学术界已经提出一些有效处理大规模图数据上可达性查询的索引方法.但是,这些方法都没有考虑上面所述节点受限的查询问题.针对这类可达性查询,如何设计一种不泄漏用户隐私、查询时间不受原图规模的影响、索引的规模不受原图规模影响的可扩展可达性索引方法是一个极具挑战性的问题.

此外,目前针对边受限和距离受限可达性查询的可达性索引只能用于中小型图数据.对于大规模图数据,这些方法将会遭遇扩展瓶颈.那么如何设计一种支持大规模图数据受限查询的可达性索引是另一个挑战性的问题.

5 结 论

可达性查询是有向图上一个最基本的查询.为了快速处理可达性查询,可达性索引技术应运而生,

它们已经广泛应用于多个计算机科学领域.随着现实应用中图的规模越来越大,图数据上查询处理和索引方法面临新的挑战.大规模图数据的可达性索引技术成为研究热点.尽管目前针对大规模图数据已经提出一些有效的可达性索引方法,但是在实际应用中仍存在许多亟待解决的问题.本文讨论了可达性索引技术的应用场景;系统总结了现有的有向图上可达性索引技术;从支持的数据规模、数据类型以及查询类别3个不同角度将现有代表性可达性索引工作进行了归类分析;从查询时间、索引的规模、索引构建时间和索引更新时间4个方面对现有的代表性工作进行了分类比较;最后分析了现有的大规模图数据可达性索引技术存在的问题并给出3个极具挑战性的研究方向.

参 考 文 献

[1] Yildirim H. Scalable reachability indexing for very large Graphs [D]. Troy, NY: Computer Science, Rensselaer Polytechnic Institute, 2011

[2] Quin L. Extensible Markup Language XML (XML) [S]. Cambridge: The World Wide Web Consortium (W3C), 2013

[3] Hawke S, Herman I. Resource Description Framework (RDF) [S]. Cambridge: The World Wide Web Consortium (W3C), 2013

[4] Wang H, Li J, Luo J, et al. Hash-base subgraph query processing method for graph-structured XML documents [J]. Proceedings of the VLDB Endowment, 2008, 1(1): 478-489

[5] Cheng J, Yu J. On-line exact shortest distance query processing [C] //Proc of ACM EDBT09. New York: ACM, 2009: 481-492

[6] Wei F. TEDI: Efficient shortest path query answering on graphs [C] //Proc of ACM SIGMOD10. New York: ACM, 2010: 99-110

[7] Cormen T, Leiserson C, Rivest R, et al. Introduction to Algorithms [M]. Cambridge: MIT Press, 2001: 595-601

[8] Boag S, Chamberlin D. An XML Query Language (XQuery), Version 2.0 [S]. Cambridge: The World Wide Web Consortium (W3C), 2013

[9] Harris G. An Query Language for RDF (SPARQL), Version 1.1 [S]. Cambridge: The World Wide Web Consortium (W3C), 2011

[10] Jagadish H. A compression technique to materialize transitive closure [J]. ACM Trans on Database System, 1990, 15(4): 558-598

[11] Chen Y, Chen Y. An efficient algorithm for answering graph reachability queries [C] //Proc of IEEE ICDE08. Piscataway, NJ: IEEE, 2008: 893-902

[12] Agrawal R, Borgida A, Jagadish H. Efficient management of transitive relationships in large data and knowledge bases [C] //Proc of ACM SIGMOD'89. New York: ACM, 1989: 253-262

[13] Nuutila E. Efficient transitive closure computation in large digraphs [D]. Helsinki, Finland: Finnish Academy of Technology, Helsinki University of Technology, 1995

[14] Wang H, He H, Yang J, et al. Dual labeling: Answering graph reachability queries in constant time [C] //Proc of IEEE ICDE06. Piscataway, NJ: IEEE, 2006: 75-86

[15] Chen L, Gupta A, Kurul M. Stack-based algorithms for pattern matching on dags [C] // Proc of ACM VLDB05. New York: ACM, 2005: 493-504

[16] Trißl S, Leser U. Fast and practical indexing and querying of very large graphs [C] // Proc of ACM SIGMOD07. New York: ACM, 2007: 845-856

[17] Jin R, Xiang Y, Ruan N, et al. Efficiently answering reachability queries on very large directed graphs [C] //Proc of ACM SIGMOD08. New York: ACM, 2008: 595-608

[18] Jin R, Ruan N, Xiang Y, et al. Path-tree: An efficient reachability indexing scheme for large directed graphs [J]. ACM Trans on Database System, 2011, 36(1): 73-84

[19] Cohen E, Halperin E, Kaplan H, et al. Reachability and distance queries via 2-hop labels [C] //Proc of ACM SIAM02. New York: ACM, 2002: 937-946

[20] Jin R, Xiang Y, Ruan N, et al. 3-HOP: A high-compression indexing scheme for reachability query [C] //Proc of ACM SIGMOD'09. New York: ACM, 2009: 813-826

[21] Cai J, Poon C. Path-hop: Efficiently indexing large graphs for reachability queries [C] // Proc of ACM CIKM'10. New York: ACM, 2010: 119-128

[22] Cheng J, Yu J, Lin X, et al. Fast computation of reachability labeling for large graphs [C] //Proc of ACM EDBT'06. New York: ACM, 2006: 961-979

[23] Cheng J, Yu J, Lin X, et al. Fast computing reachability labelings for large graphs with high compression rate [C] // Proc of ACM EDBT08. New York: ACM, 2008: 193-204

[24] Schenkel R, Theobald A, Weikum G. HOPI: An efficient connection index for complex XML document collections [C] //Proc of ACM EDBT04. New York: ACM, 2004: 237-255

[25] Fan W, Wang X, Wu Y. Performance guarantees for distributed reachability queries [J]. Proceedings of the VLDB Endowment, 2012, 5(11): 1304-1315

[26] Schaik S, Moor O. A memory efficient reachability data structure through bit vector compression [C] //Proc of ACM SIGMOD11. New York: ACM, 2011: 913-924

[27] Yildirim H, Chaoji V, Zaki M. Grail: Scalable reachability index for large graphs [J]. Proceedings of the VLDB Endowment, 2010, 3(1): 276-284

[28] Yildirim H, Chaoji V, Zaki M. GRAIL: A scalable index for reachability queries in very large graphs [J]. Proceedings of the VLDB Endowment, 2012, 21(4): 509-534

[29] Jin R, Ruan N, Dey S, et al. SCARAB: Scaling reachability computation on large graphs// Proc of ACM SIGMOD12. New York: ACM, 2012: 169-180

[30] Seufert S, Anand A, Bedathur S, et al. FERRARI: Flexible and efficient reachability range assignment for graph indexing [C] //Proc of IEEE ICDE13. Piscataway, NJ: IEEE, 2013: 1009-1020

[31] Jin R, Hong H, Wang H, et al. Computing label-constraint reachability in graph databases [C] //Proc of ACM SIGMOD10. New York: ACM, 2010: 123-134

[32] Roditty L, Zwick U. A fully dynamic reachability algorithm for directedgraphs with an almost linear update time [C] // Proc of ACM STOC04. New York: ACM, 2004: 184-191

[33] King V, Sagert G. A fully dynamic algorithm for maintaining the transitive closure [J]. Journal of Computer System Science, 2002, 65(1): 150-167

[34] Demetrescu C, Italiano G. Fully dynamic transitive closure: Breaking through the o(n2) barrier [C] //Proc of IEEE FOCS00. Piscataway, NJ: IEEE, 2000: 381-389

[35] Henzinger M, King V. Fully dynamic biconnectivity and transitive closure [C] //Proc of IEEE FOCS95. Piscataway, NJ: IEEE, 1995: 664-672

[36] Demetrescu C, Italiano G. Dynamic shortest paths and transitive closure [J]. Journal of Discrete Algorithms, 2006, 4(3): 353-383

[37] Henzinger M, Poutra H. Certificates and fast algorithms for biconnectivity in fully-dynamic graphs [G] //LNCS 979: Proc of the 3rd Annual European Symp Corfu. Berlin: Springer, 1995: 171-148

[38] Henzinger M, King V. Randomized fully dynamic graph algorithms with polylogarithmic time per operation [J]. Journal of the ACM, 1999, 46: 516-527

[39] Henzinger M, Rao S, Gabow H. Computing vertex connectivity: New bounds from old techniques [C] //Proc of IEEE FOCS96. Piscataway, NJ: IEEE, 1996: 462-471

[40] King V. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs [C] //Proc of IEEE FOCS99. Piscataway, NJ: IEEE, 1999: 81-91

[41] King V, Sagert G. A fully dynamic algorithm for maintaining the transitive closure [C] //Proc of ACM STOC'99. New York: ACM, 1999: 492-498

[42] King V, Thorup M. A space saving trick for directed dynamic transitive closure and shortest path algorithms [G] //LNCS 2108: Proc of the 7th Annual Int Conf on COCOON 2001. Berlin: Springer, 2001: 268-277

[43] Roditty L. A faster and simpler fully dynamic transitive closure [C] //Proc of ACM SODA03. New York: ACM, 2003: 404-412

[44] Bramandia R, Choi B, Ng W. Incremental maintenance of 2-hop labeling of large graphs [J]. IEEE Trans on Knowledge and Data Engineering, 2010, 22(5): 682-698

[45] Liu Xuhui, Feng Jianhua, Hong Qin. An update-aware graph reachability query algorithm [J]. Journal of Natural Science, 2012, 34(10): 49-52 (in Chinese)
(刘旭辉,冯建华,洪亲. 一种支持更新的图可达性查询算法 [J]. 计算机科学, 2012, 34(10): 49-52)

[46] Yildirim H, Chaoji V, Zak M. DAGGER: A scalable index for reachability queries in large dynamic graphs [EB/OL]. [2013-01-06]. http://arxiv.org/abs/1301.0977

[47] Li C, Ling T, Hu M. Efficient updates in dynamic XML data: From binary string to quaternary string [J]. Proceedings of the VLDB Endowment, 2008, 17(3): 573-601

[48] Amagasa T, Yoshikawa M, Uemura S. QRS: A robust numbering scheme for XML documents [C] //Proc of IEEE ICDE03. Piscataway, NJ: IEEE, 2003: 705-707

[49] Schmidt A, Waas F, Kersten M, et al. XMark: A benchmark for XML data management [C] //Proc of ACM VLDB02. New York: ACM, 2002: 974-985

[50] Zhang C, Naughton J, Dewitt D, et al. On supporting containment queries in relational database management systems [C] //Proc of ACM SIGMOD01. New York: ACM, 2001: 425-436

[51] Xu L, Ling T, Wu H, Bao Z. DDE: From dewey to a fully dynamic XML labeling scheme [C] // Proc of ACM SIGMOD09. New York: ACM, 2009: 719-730

[52] Wang H, Park S, Fan W, et al. Vist: A dynamic index method for querying xml data by tree structures [C] // Proc of ACM SIGMOD03. New York: ACM, 2003: 110-121

Fu Lizhen, born in 1982. PhD, assistant professor at North University of China. Her Main research interests focus on XML database and graph database.

Meng Xiaofeng, born in 1964. Professor and PhD supervisor at Renmin University of China. Executive director of China Computer Federation. His main research interests include web data integration, cloud and mobile data management and flash databases.