网络出版时间：2017-08-19 00:16:26
网络出版地址：http://kns.cnki.net/kcms/detail/11.1826.TP.20170819.0016.006.html

面向大图的可达性查询处理算法

陈子阳[1),2)] 陈伟[2),3)] 李娜[2)] 周军锋[2),4)]

[1)](上海立信会计金融学院 信息管理学院，上海 201620)

[2)](燕山大学 信息科学与工程学院，河北 秦皇岛 066004)

[3)](河北环境工程学院 信息工程系，河北 秦皇岛 066102)

[4)](东华大学 计算机科学与技术学院，上海 201620)

摘 要 图的可达性查询处理是生物信息领域的热点问题之一，用于测定蛋白质交互网络中任意两个蛋白质分子间是否存在交互作用. 针对已有在线搜索算法在可达查询比例增大时效率下降明显及性能不稳定的问题，提出优化的 OPT-R 算法. 首先，提出最优生成树的概念，使得采用最优生成树的 OPT-R 算法可以在常量时间回答更多的可达查询. 同时提出基于栈的互逆拓扑顺序，使得 OPT-R 可以在常量时间回答更多的不可达查询. 提出相应的最优生成树及互逆拓扑顺序生成算法，并通过实验，基于 20 个不同规模的真实数据集从不同角度对算法的高效性进行了验证.

关键词 大图；有向无环图；可达性查询处理；最优生成树

中图法分类号 TP301

论文引用格式：

陈子阳，陈伟，李娜，周军锋，面向大图的可达性查询处理算法,2017, Vol.40,**在线出版号** No.128

CHEN Zi-Yang,CHEN Wei,LI Na,HOU Jun-Feng,Efficient Processing Algorithm for Reachability Queries based on Big Graph, 2017, Vol.40,Online Publishing No. 128

Efficient Processing Algorithm for Reachability Queries based on Big Graph

CHEN Zi-Yang[1),2)] CHEN Wei[2),3)] LI Na[2)] ZHOU Jun-Feng[2),4)]

[1)](School of Information Management, Shanghai Lixin University of Accounting and Finance, Shanghai 201620)

[2)](School of Information Science and Engineering, Yanshan University, Qinhuangdao, Hebei 066004)

[3)](Department of Information Engineering, Hebei University of Environmental Engineering, Qinhuangdao, Hebei 066102)

[4)](School of Computer Science and Technology, Donghua University, Shanghai 201620)

Abstract Given a directed graph, a reachability query asks whether there exists a path from a node to another one in the directed graph. Reachability query processing is one of the fundamental operations in graph data processing, and is one of the hot issues in the field of bioinformatics, which focuses on the study of the interaction between any two proteins in the protein interaction network. Considering that each node can reach every other node in a strongly connected component (SCC), and all SCCs can be identified in linear time, existing algorithms always assume that the input graph is a directed acyclic graph (DAG). The two kinds of extreme methods to answer a reachability query are (a) computing transitive closure and answering the query in

本课题得到国家自然科学基金(No. 61472339, 61572421, 61272124)资助.陈子阳，男，1973年生，博士，教授/博导，上海立信会计金融学院特聘教授，计算机学会(CCF)会员 (21309M)，主要研究领域为数据库理论与技术. E-mail: zychen@ysu.edu.cn. 陈伟(**通讯作者**)，女，1980年生，博士研究生，副教授，主要研究领域为图数据查询处理. E-mail: chenwei@ysu.edu.cn. 李娜，女，1990年生，硕士研究生，主要研究领域为图数据查询处理. E-mail: 597183565@qq.com. 周军锋，男，1977年生，博士，教授，计算机学会(CCF)会员 (16083M)，主要研究领域为XML和半结构化数据查询技术. E-mail: zhoujf@ysu.edu.cn.

constant time by affording expensive storing cost, and (b) traversing the DAG to get the final answer without any pre-processing cost of index construction. With the increase of graph data, the two kinds of extreme methods are not scalable in practice. Existing methods always make trade-off between the two extremes, such that to accelerate index construction, reduce the index size, and improve the performance of query processing as more as possible. Existing methods can be generally classified into two categories: (1) the Label-Only methods, which assign each node a label to store the complete reachable information, such that when answering the given query, we do not need to traverse the input graph, (2) the Online-Search methods, which also assign each node a label, but only store partial reachable information to achieve fast index construction speed. However, we may need to traverse the input graph when we cannot get the final answer by the node labels. As a comparison, the Label-Only methods are not scalable for big graphs, and by designing efficient pruning techniques, the Online-Search methods can achieve similar query performance, and therefore are more applicable in practice. However, the current Online-Search methods are not efficient when processing reachable queries. In fact, many reachability queries will return positive result, and the pruning techniques of existing Online-Search methods focus on answering negative queries, they may not be efficient for reachable queries. Considering that existing algorithms on reachability query processing are inefficient and unstable when the ratio of reachable queries increases, we propose an optimized algorithm, namely OPT-R, to accelerate reachability query processing. To make OPT-R work, we first propose optimal spanning tree, which contains the maximum number of reachable queries compared with other spanning trees, such that OPT-R can answer more reachable queries in constant time on average. The main idea behind the optimal spanning tree is that to make a spanning tree covering the maximal number of reachable node pairs, each node in the spanning tree should have the most number of ancestors. Based on this idea, we take the topological level of each node as its tree height, and construct a spanning tree in linear time, for which each node satisfies that its tree height equals its topological level in the given graph. The optimal spanning tree is used to accelerate testing of reachable queries. To accelerate the testing of unreachable queries, we propose the DFS based topological order, namely ST-order, and the reverse ST-order to help OPT-R answer more unreachable queries in constant time. We further propose efficient index construction algorithms to get the optimal spanning tree and the two ST-orders that are reverse to each other. We conduct extensive experimental study based on 20 real datasets. The experimental results show that OPT-R is more efficient in index construction and query processing.

Key words Big Graph; Directed Acyclic Graph; Reachability Query Processing; Optimal Spanning Tree

1 引言

生物信息学是应用信息科学的方法和技术研究和分析生物体细胞、组织、器官的生理、病理等问题的学科. 随着信息技术和生物学的迅猛发展, 产生了诸如蛋白质相互作用网络、基因调控网络、代谢路径等生物网络数据, 这些数据具有量大且复杂程度高的特点. 图能有效描述事物之间的复杂关系, 是生物计算中最常用的数据结构, 大量的生物计算问题最后都转化为基于图的计算问题[1,2], 其中的核心问题之一是如何高效回答蛋白质分子之间是否存在相互作用, 反映在图上即为可达性查询.

给定有向图 $\mathcal{G}=(\mathcal{V}, \mathcal{E})$, 可达性查询 $u?\rightarrow v$ 用于探查从 u 开始是否存在一条路径可以到达 v. 回答可达性查询是图数据处理中的核心操作之一, 除了单纯回答两点之间是否可达的问题之外, 很多其他的图操作最终都会转化为两点之间是否可达的问题. 例如, 有向图上的模式匹配问题需通过回答大量顶点对之间是否可达来确定最终答案[3,4], 可达查询处理效率的提升或下降会通过级联效应得到成倍放大, 显著影响上层问题的处理性能. 鉴于其本身的重要性, 可达性查询处理得到了持续关注和深入研究. 随着实际应用中图规模的不断增长, 高效处理可达性查询仍然是极具挑战性的问题.

考虑到强连通分量(Strongly Connected Component, SCC)中的任意两个顶点可以相互到达,

并且已经存在线性时间 $O(|\mathcal{V}|+|\mathcal{E}|)$找到 $\mathcal{G}$的强连通分量的算法[5], 现有方法关注的重点是: 如何快速回答和 $\mathcal{G}$对应的有向无环图(Directed Acyclic Graph, DAG) $G=(V, E)$上的两个顶点 u 和 v 是否可达.

回答可达性查询的两种极端方法[6,7]分别是: (1) 在求解传递闭包(Transitive Closure, TC)并负担高昂存储代价 $O(|V|^2)$的前提下在 $O(1)$时间给出答案; (2) 无需任何预处理的情况下从 u 出发, 通过深度优先搜索(Depth-First-Search, DFS)的方式判断 u 是否可达 v, 其代价是 $O(|V|+|E|)$. 随着图的规模不断变大, 两种极端方法因为扩展性差在实际中难以真正使用. 现有研究都是在以上两种极端情况之间进行平衡, 以便在尽可能的情况下加速索引构建, 降低索引空间, 同时加速查询处理的速度.

现有方法根据回答可达性查询时是否需要遍历原始图可以分为两类: (1) 标签类(Label-Only)[8-20]: 该类方法的特点是回答查询时**无需**访问原始图, 做法是先对给定的图进行预处理, 为其中每个顶点附加一个标签, 查询处理时只需比较两个顶点的标签即可返回最终结果, 其中的代表性工作包括 TF[16]、DL[17]以及 PLL[18]. (2) 在线搜索类(Online-Search)[6,7,21-24]: 该类方法的特点是回答查询时**有可能**需要访问原始图中的顶点. 和DFS方法不同, 在线搜索类方法也会对原始图进行预处理, 构建每个顶点的标签, 作用是记录部分可达性信息. 查询处理时, 这些标签用于减小搜索空间, 尽快得到结果, 其中的代表性工作包括 GRAIL[6,22]、FERRARI[23]、FELINE[7]以及 IP+[24]. 两类方法相比较, 一方面, 标签类方法的最大问题在于其较高的索引构建和存储代价. 由于实际中图的规模越来越大, 在机器内存容量受限的情况下, 这些方法尽管在某个方面可能表现很好, 但整体上依然存在无法处理的情况. 另一方面, 在线搜索类方法索引构建时间短, 索引空间小. 尽管查询处理的复杂度较高, 但是研究者通过设计高效的剪枝策略, 依然可以获得和标签类方法相当的查询响应速度, 因而在实际中具有更多的应用空间.

对于给定的查询 $u?\rightarrow v$, 现有在线搜索类方法主要通过快速判断顶点间的**不可达**来减小搜索空间, 如 GRAIL[6,22]、FERRARI[23]、FELINE[7]以及 IP+[24], 这些方法适合不可达查询的数量占绝对多数的情况[6]. 实际中, 由于查询点之间或多或少存在某种联系, 很多被处理的查询会返回肯定的结果[25], 因此, 以判断**不可达**来减小搜索空间的方式在实际应用中对系统性能的提升作用有限.

本文针对现有在线搜索类方法存在的问题进行研究, 提出索引构建时间和存储空间均为线性, 且查询响应更快的在线搜索方法来回答给定的可达性查询. 具体来说, 本文的贡献如下:

(1) 提出最优树的概念用于减小可达查询的搜索空间, 并证明其在获取可达信息方面相对于其他生成树的最优性.

(2) 提出基于栈的拓扑顺序及其逆序组成的双拓扑标签来减小不可达查询的搜索空间.

(3) 提出时间和空间均为线性的索引构建算法, 以及基于最优树和双拓扑标签的查询处理算法.

(4) 在20个真实数据集上通过实验对本文方法和已有方法进行了比较. 实验结果验证了本文提出方法的高效性.

本文的后续内容组织如下: 第 2 节对背景知识及相关工作进行介绍, 第 3 节介绍本文方法的基本思想, 第 4 节对算法进行了详细描述和分析, 第 5 节通过实验对本文提出的算法进行性能验证, 最后在第 6 节总结全文.

2 背景知识和相关工作

2.1 背景知识

对于给定的有向图 $\mathcal{G}=(\mathcal{V}, \mathcal{E})$, 和已有的研究[6-27]一样, 我们研究从 $\mathcal{G}$转换而来的 DAG $G=(V, E)$上的可达性查询处理问题, 其中 V 是顶点集, E 是边集. V 的一个顶点 u 代表了 $\mathcal{G}$中的一个 SCC S_u, E 的一条边(u, v)表示在 SCC S_u (S_v)中至少存在一个顶点 u'(v')满足$(u', v') \in \mathcal{E}$.

G 的拓扑排序是对具有偏序关系的顶点集 V 进行线性扩展的过程. 假定 x_v 是 v 的拓扑序号, 对于 G 的任意边(u, v), 拓扑顺序满足 $x_u<x_v$. 对 G 中所有顶点的邻接表进行拓扑排序的时间复杂度为 $O(|V|+|E|)$[28]. G 中顶点 u 的拓扑层指以 u 为终点的最长路径包含的边数.

后续讨论中, $in_G(u)=\{v|(v,u)\in E\}$表示 u 的入邻居, $out_G(u)=\{v|(u,v)\in E\}$表示 u 的出邻居. $in_G^*(u)$ 表示 G 中可达 u 的顶点集, 其中 $u\notin in_G^*(u)$. $out_G^*(u)$表示 G 中 u 可达的顶点集, 其中 $u\notin out_G^*(u)$. 简单起见, 称 $in_G(u)/out_G(u)/in_G^*(u)/out_G^*(u)$ 是 u 在图 G 中的父亲/孩子/祖先/后代. 类似的, 给定 G 的生成树 T, 称 $in_T(u)/out_T(u)/in_T^*(u)/out_T^*(u)$ 是 u 在 T 中的父

亲/孩子/祖先/后代.

表 1 是本文内容涉及到的重要符号及其意义.

表 1 本文所用符号及其意义

符号	意义
$G=(V, E)$	有向无环图, V 为顶点集, E 为边集
X, Y	G 的拓扑顺序
x_u	顶点 u 在 X 中的拓扑序号
$in_G(u)$ ($in_T(u)$)	u 在图 G(树 T)中的父亲
$in^*_G(u)$ ($in^*_T(u)$)	u 在图 G(树 T)中的祖先
$out_G(u)$ ($out_T(u)$)	u 在图 G(树 T)中的孩子
$out^*_G(u)$ ($out^*_T(u)$)	u 在图 G(树 T)中的后代

2.2 相关工作分析

现有的可达性查询算法可以分为两类, 一类是标签法, 即通过比较查询顶点的标签来回答查询; 另一类是在线搜索法, 其中顶点标签记录了部分可达性信息, 该方法结合了标签和 DFS, 在 DFS 的过程中通过标签来缩小搜索空间.

2.2.1 标签法

这类算法[8-20]主要通过压缩顶点的传递闭包以达到减小索引大小及查询时间的目的. 最近的相关工作包括 TF[16]、DL[17]、PLL[18].

TF[16]对给定的 DAG G 进行递归拓扑折叠来减少 2-hop 标签的计算开销. DL[17]和 PLL[18]用相同的思想计算 2-hop 标签. 对于这些方法而言, 每个顶点 u 的标签由两部分组成, 分别是 $L_{in}(u)$和 $L_{out}(u)$. $L_{in}(u)$ 中存储的是可达 u 的顶点集, $L_{out}(u)$ 中存储的是 u 可达的顶点集. 假设顶点按照某种顺序, 如$(|out_G(u)|+1)\times(|in_G(u)|+1)$, 降序排列, DL 和 PLL 在正向和反向 BFS (Breadth-First-Search, 广度优先搜索)过程中, 通过枚举, 把 u 加入 u 能到达的顶点 v 的 $L_{in}(v)$标签中, 同时将 u 加入能到达 u 的顶点 v 的 $L_{out}(v)$标签中. 在 BFS 过程中, DL 和 PLL 会在遇到每一个顶点 v 时, 通过测试 u 和 v 是否已经能够通过已有的标签可达来进行剪枝, 提前终止 BFS 的执行.

2.2.2 在线搜索法

这类算法[6,7,21-24]以 DFS 的方式来回答顶点 u 是否可达 v, 最近的相关工作包括 GRAIL[6,22]、FERRARI[23]、FELINE[7]、IP+[24]. 所有这些方法都给顶点设定标签, 并通过快速判断顶点的不可达关系来缩小搜索空间.

GRAIL[6,22]算法通过为每个顶点设定 k 个区间, $label(u)=\{I_{u1}, I_{u2}, ..., I_{uk}\}$, 每个区间可以覆盖其所有传递闭包中的顶点, 因而通过区间包含无法判定两个顶点可达, 但不包含可以判定顶点不可达. 给定查询 $u?\rightarrow v$, 如果存在 $i\in[1, k]$使得 $I_{vi}\not\subseteq I_{ui}$, 则 $u\nrightarrow v$, 否则 GRAIL 需要从 u 开始以 DFS 方式在遍历图的过程中判断 u 是否可达 v. 此外, GRAIL 还利用生成树的区间来加速可达查询的处理. 对于不能通过区间来判断可达关系的顶点, 作者还探讨了通过 Exception List 来记录可达关系从而回答查询可达性.

考虑到用区间精确表达可达性信息时, 区间个数不确定会导致存储空间增大的问题, FERRARI[23]尝试把一些精确区间合并为一个大的近似区间以减少存储空间. 和 GRAIL 类似, FERRARI 算法中每个顶点最多有 k 个区间, 但不同的是这 k 个区间中可能包含表达精确包含关系的区间. 给定查询 $u?\rightarrow v$, 如果 $label(u)$没有覆盖 v, 则 FERRARI 返回否定结果; 反之, 如果 v 被 $label(u)$的一个精确区间包含, 则 FERRARI 无需进一步遍历其它顶点, 直接返回肯定结果; 只有当 v 被 $label(u)$的近似区间覆盖, FERRARI 才会开始继续以 DFS 方式访问其它顶点. 和 GRAIL 类似, FERRARI 用拓扑号和种子顶点来加速查询处理.

FELINE[7]利用两个拓扑号 x_u 和 y_u 来标记顶点 u, 基于这两个拓扑序号, 对于给定的查询 $u?\rightarrow v$, 如果 $x_u\prec x_v$ 或者 $y_u\prec y_v$, 那么 FELINE 直接返回否定结果; 反之如果 $x_u<x_v$ 并且 $y_u<y_v$, FELINE 从 u 开始以 DFS 方式访问其它顶点. 在每次顶点的标签比较过程中, FELINE 也同时使用了生成树区间和拓扑层来加速过滤, 提升查询处理的性能.

最近的算法 IP+[24]基于独立置换的思想给每个顶点 u 设定两个标签 $L_{in}(u)$和 $L_{out}(u)$, 每个标签里最多含 k 个元素, 利用这两个标签可以快速回答部分不可达查询, 如果当前顶点的标签不能判断是否可达时, IP+需以 DFS 方式访问其它顶点. 和以上在线类方法类似, IP+也使用了拓扑层和出入度较大的顶点来加速查询处理的性能.

表 2 展示了最近一段时间提出的不同类型算法的复杂度对比, 其中 OPT-R 是本文提出的算法. 有关其它早期算法的对比见文献[24]. 由于文献[16]没有给出关于顶点和边数的复杂度分析, 我们没有在表 2 中展示 TF 的复杂度.

从表 2 可以看出, 标签类算法虽然查询响应的复杂度较低, 但由于其构建索引的复杂度太大, 导致在内存容量有限的前提下, 出现索引构建时间太长, 甚至无法处理的情况. 与之相比, 在线搜索类

算法的索引大小和索引时间的复杂度普遍较低, 其最大的问题是查询处理的复杂度较大. 实际上, 通过设计高效的剪枝技术, 现有的在线搜索算法, 如IP+, 完全可以做到在各个方面比标签类算法更高效. 然而, 现有的在线搜索类算法也存在自身的不足. 当可达查询的比例增大时, 现有的在线搜索类算法在回答可达性查询时的运行时间显著增加. 主要原因在于现有的在线类搜索算法对于可达查询的处理能力有限, 这也是本文主要解决的问题.

表 2　不同算法的复杂度比较

算法	索引大小	索引时间	查询时间
DL	$O(L\|V\|)$	$O(L\|V\|(\|V\|+\|E\|))$	$O(L)$
PLL	$O(L\|V\|)$	$O(L\|V\|(\|V\|+\|E\|))$	$O(L)$
GRAIL	$O(k\|V\|)$	$O(k(\|V\|+\|E\|))$	$O(\|V\|+\|E\|)$
FERRARI	$O((k+s)\|V\|)$	$O(k^2(\|V\|+\|E\|))$	$O(\|V\|+\|E\|)$
FELINE	$O(\|V\|)$	$O(\|E\|+\|V\|\log\|V\|)$	$O(\|V\|+\|E\|)$
IP+	$O((k+h)\|V\|)$	$O((k+h)(\|V\|+\|E\|))$	$O(kr^2\|V\|)$
OPT-R	$O(\|V\|)$	$O(\|V\|+\|E\|)$	$O(\|V\|+\|E\|)$

除了单纯求解有向无环图上的可达性查询, 文献[29]基于时态图研究时间受限的可达性查询问题, 文献[30]基于分布图研究集合中顶点间的可达性查询问题, 文献[31,32]则研究了长度受限的 k 步可达查询问题.

3 基本思想

现有的在线搜索类方法着眼于快速确定不可达查询; 对可达查询而言, 没有特别有效的处理方法. 实际中, 由于查询点之间或多或少存在某种联系, 很多被处理的查询会返回肯定的结果[25]. 当实际中可达查询占比较大的情况出现时, 现有在线搜索类算法的性能变差是可预期的. 针对该问题, 本文同时从可达和不可达两个角度设计高效判定方法, 加快在线搜索方法的查询响应速度. 具体来说, 通过设计最优树来获取最多的可达性信息, 从而加速可达查询的判定; 通过设计基于栈的互逆拓扑顺序来加速不可达查询的判定.

3.1 最优生成树及其性质

为了加速查询响应速度, 已有方法[7,22,33]基于 G 的生成树 $T=(V_T, E_T)$为每个顶点 u 指定一个区间 $I_u=[s_u, e_u]$, 用于判定 T 中顶点之间的祖先—后代关系. 不失一般性, 假设 s_u 表示遍历 T 时 u 的 DFS 序号, e_u 表示以 u 为根的子树 T_u 中 DFS 序号的最大值. 给定查询 $u?\rightarrow v$, 如果 $I_v\subset I_u$, 则表示在树 T 中, u 是 v 的祖先, 因而 $u\rightarrow v$. 显然 G 的任意生成树都可以根据两个顶点的区间包含关系判定二者是否一定可达, 且生成树顶点间的祖先—后代关系仅能表示 G 的一部分可达性信息. 问题是: 在所有生成树中是否存在包含最多可达信息的生成树? 如果存在, 该生成树有什么性质?

假设 $C(T)$表示树 T 包含的可达顶点对数量, 显然有如下公式成立.

$$C(T)=\sum_{u\in V_T}\left|out_T^*(u)\right| \tag{1}$$

则现在问题变为: 给定 DAG G, 求 G 的最优生成树 T_{opt}, 使得对于其他任意生成树 T 而言, $C(T)\le C(T_{opt})$ 成立.

根据公式(1), 为了得到最优生成树, 应该让生成树中的每个顶点的后代尽可能多, 从而保证 $C(T)$ 尽可能大. 由于顶点数量是固定的, 在生成树中增加一个顶点的后代就意味着减少了另一个顶点的后代数量. 换一个角度, 我们有如下公式成立.

$$\sum_{u\in V_T}\left|out_T^*(u)\right|=\sum_{u\in V_T}\left|in_T^*(u)\right| \tag{2}$$

根据公式(2), 生成树的可达顶点对数量可以通过顶点的祖先数量来表示. 由于在生成树中, 一个顶点祖先的增多不会让另一个顶点的祖先数量减少, 因此该最优化问题的等价问题是: 生成树中的顶点应该满足什么条件才能拥有最多祖先?

由于一个顶点的拓扑层表示了其它顶点到该顶点的最长路径的长度, 因此我们有如下定理.

定理 1. 给定 DAG G 及其生成树 T, T 是 G 的最优生成树当且仅当 T 中任意顶点的祖先数量等于其拓扑层.

证明: 给定 DAG G, 生成树 T 及其任一顶点 v, v 在 T 中的祖先数量为$|in_T^*(v)|$. 由于 G 中任一顶点 v 在其任一生成树中的祖先数量小于等于其拓扑层, 因此 T 是 G 的最优生成树(T 包含的可达顶点对数量最大)当且仅当 T 中任意顶点的祖先数量等于其拓扑层.

证毕.

例 1. 对图 1 所示的 DAG G 来说, 图 2 为 G 的两棵生成树, 其中 T_{opt} 为最优生成树, 满足其中每个顶点在树中的高度等于其在 G 中的拓扑层, T 并非按照最优树的要求来生成, 因而其中顶点的高度不一定等于其拓扑层. 比较而言, T 可用于回答 23

个可达查询，而最优树 T_{opt} 可回答 34 个可达查询. 显然，在查询处理的遍历过程中，在每一步，最优树可用于回答更多的可达查询.

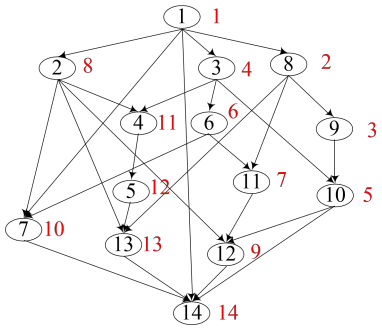


图 1 有向无环图 G，其中每个顶点用其 ST-order X 中的拓扑序号 x_v 表示，v 旁边的数字表示 v 在逆向 ST-order Y 中的拓扑序号 y_v.

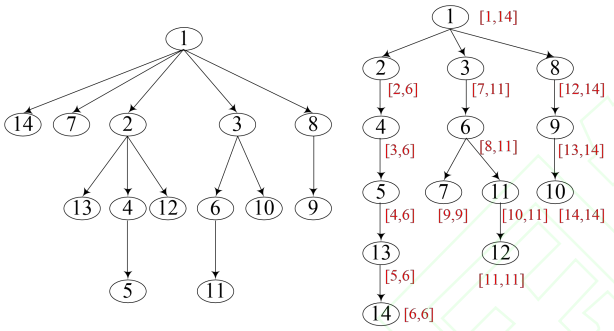


图 2 G 的生成树 T 和最优生成树 T_{opt}

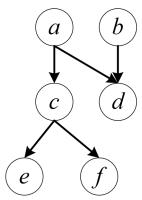
需要说明的是，给定 DAG G，最优树在 G 的所有生成树中可以涵盖最多的可达顶点对数量，该值可用公式(2)来得到，表示 G 的任意生成树能够覆盖的可达顶点对数量的上界. 从公式(2)可知，最优树能够覆盖的可达顶点对数量取决于树中每个顶点的祖先数量，树的高度越高(G 的拓扑层数越大)，最优树能够覆盖的可达顶点对数量越多，该值和 G 包含的可达顶点对的数量无关，只依赖于 G 的结构和拓扑层数.

3.2 基于栈的互逆拓扑顺序

一个 DAG 体现了顶点间的偏序关系，而拓扑排序是顶点偏序关系的全序化. 给定两个顶点 u, v 的拓扑顺序 x_u, x_v，虽然 $x_u<x_v \nRightarrow u\rightarrow v$，但 $x_u>x_v \Rightarrow u\nrightarrow v$，因而拓扑顺序常用于快速判定不可达查询. 为了尽可能多的识别顶点间的不可达关系，文献[7]使用两个拓扑顺序来增强不可达查询的过滤能力. 由于通过两个拓扑顺序来最大化不可达查询的识别量是 NP-hard 问题[34]，文献[7]通过逆转第一个拓扑顺序来得到第二个拓扑顺序，以便尽可能多的识别不可达查询. 虽然查询处理速度得到了一定程度的提高，但这种方法仍存在两方面的问题: (1) 两个拓扑顺序识别不可达查询的能力严重依赖于第一个拓扑顺序, (2) 第二个拓扑顺序求解的时间复杂度高，为 $O(|V|\log|V|)$. 为此，我们提出基于栈的互逆拓扑顺序来增强不可达查询的过滤效果.

直观上看，基于栈的拓扑顺序(stack-based topological order, ST-order)是在拓扑排序的过程中借助于栈来生成的拓扑顺序. 在生成第一个拓扑顺序 X 时，首先将所有入度为 0 的顶点入栈，每当处理完栈中的一个顶点，其可被拓扑访问的出邻居(指顶点的所有入邻居都已被访问)优先入栈. 这里第二个拓扑顺序 Y 同样基于栈来产生，二者互逆的意思指在生成 Y 的过程中拓扑访问任一顶点 u 时，在 X 中位于顶点 u 后面的可拓扑访问的顶点在 Y 中先被访问，从而使得不可达顶点对在 X 和 Y 中拓扑序号的大小关系尽可能相反，以此来尽可能多的识别不可达查询.

例 2. 考虑图 3 的 DAG G_1. 当使用互逆拓扑 W 和 Z 进行不可达判断时，对于查询 $b?\rightarrow c$ 而言，由于 c 的拓扑序号在 W 和 Z 中均位于 b 之后，因而根据 W 和 Z 无法判定 b 不可达 c. 基于相同的原因，对查询 $b?\rightarrow f$ 和 $b?\rightarrow e$ 而言，同样无法根据 W 和 Z 进行判定. 当使用基于栈的互逆拓扑 X 和 Y 处理以上 3 个查询时，虽然 b 在 Y 中的拓扑序号位于 c、f、e 的前面，但由于 b 在 X 中的拓扑序号位于 c、f、e 之后，因而可以直接判定 b 不可达 c、f、e.



(a) *DAG* G_1

W	a	b	c	d	e	f
Z	b	a	d	c	f	e
拓扑序号	1	2	3	4	5	6
X	a	c	e	f	b	d
Y	b	a	d	c	f	e

(b) 互逆拓扑比较

图 3 不同互逆拓扑示例，其中 W、Z、X、Y 表示四种拓扑顺序，W 和 Z 为互逆拓扑，X 和 Y 表示基于栈的互逆拓扑.

和文献[7]采用任意互逆拓扑顺序相比，ST-order X 和 Y 能在尽可能的情况下判定更多的不可达查询.

4 算法描述

4.1 索引构建

本文方法中, 每个顶点的标签由 3 部分构成: (1) 顶点在最优生成树上的区间, (2) 互逆的两个拓扑号, (3) 拓扑层. 相应的, 索引构建也由 3 步构成: (1) 求最优生成树并为其中的顶点设定区间, (2) 求解互逆的 ST-order, (3) 求解顶点的拓扑层. 由于拓扑层可在求解拓扑顺序时一并得到, 且被用于构建最优生成树区间, 下面首先介绍基于栈的互逆拓扑顺序生成算法, 然后介绍最优树区间的构建算法.

4.1.1 求解互逆 ST-order

求解互逆 ST-order 总体上分两步, 首先求解正向 ST-order X, 然后基于 X 求解逆向 ST-order Y. 拓扑层也一并在拓扑排序的过程中求得.

对于给定的一个 DAG G, 可以通过以下步骤得到 G 的 ST-order X: (1) 将 G 中入度为 0 的顶点插入栈 S 中; (2) 从 S 中移除一个顶点 v, 并标记 v 的出栈顺序, 即拓扑顺序, 然后删去 v 及所有从 v 出发的边, 并把 v 的孩子中入度变为 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 ST-order X.

给定 ST-order X, 其逆向拓扑顺序 Y, 同样基于栈得到. 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. 当从 S 中弹出一个顶点 u 时为 u 设定其逆向拓扑序号, 该序号等于 u 出栈的顺序. 之后, 删除 u 及其出边, 并将 u 的出邻居中入度为 0 的顶点按照它们在 X 中从小到大的顺序入栈. 以上操作不断循环, 直到栈空为止, 相应的顶点出栈顺序 Y 称为 X 的逆向拓扑顺序. 由于给定 Y 的情况下, 同样可得到 X, 因而称 X 和 Y 为基于栈的互逆拓扑顺序.

需要说明的是: 和一般的拓扑排序不同, ST-order 的求解基于栈. 在第一次拓扑排序过程中, 当栈顶元素出栈时, 会把栈顶元素写回其入邻居的邻接表中. 这样先访问的顶点就会先写回入邻居的邻接表. 当第一次拓扑排序结束时, 每个顶点的邻接表按拓扑顺序升序有序. 第二次拓扑排序求解 ST-order Y时, 只需将入度为 0 的顶点按照 X 拓扑升序入栈, 生成的拓扑序号就是 Y.

算法 1 用于求解给定 DAG G 的拓扑顺序, 其中 l_v 表示顶点 v 的拓扑层, x_v 表示顶点 v 在 G 的拓扑顺序 X 中的拓扑序号, *indeg*[·]表示存储顶点入度的临时数组, S 为栈. 算法 1 中第 3 行将入度为 0 的顶点依次入栈 S. 第 5 行开始以深度优先的方式拓扑访问栈中的顶点. 当栈不为空时, 从栈中弹出一个顶点 v(第 6 行), 并给 v 设定拓扑编号(第 7 行), 然后访问 v 的邻接表, 计算 v 的出邻居的拓扑层, 如果出邻居的入度减 1 后变为 0, 则该出邻居入栈(第 12 行). 算法 1 执行后得到 G 的 ST-order X, 且所有顶点的邻接表均按拓扑顺序升序有序. 算法运行结束后输出每个顶点 v 的拓扑序号 x_v 以及该顶点的拓扑层 l_v.

算法 1. TOPO(G=(V, E))算法

输入: G=(V, E)

输出: 每个顶点 v 的拓扑序号 x_v 及其拓扑层 l_v

```
1   FOR EACH (v∈V) DO
2       l_v ← 0; indeg[v] ← |in_G(v)|
3       IF (indeg[v] = 0) THEN push(S,v)
4   topologyNum ← 0
5   WHILE (S ≠ ∅) DO
6       v ← pop(S)
7       x_v ← topologyNum
8       topologyNum ← topologyNum+1
9       FOR EACH (w ∈ out_G(v)) DO
10          l_w ← max(l_w, l_v+1)
11          indeg[w] ← indeg[w]–1
12          IF (indeg[w] = 0) THEN push(S,w)
```

注意算法 1 执行后, 所有顶点的邻接表均按拓扑顺序升序有序. 因而基于 X 求 Y 时, 第 3 行入度为 0 的顶点的入栈顺序和求解 X 时的入栈顺序正好相反, 该顺序可通过在求解 X 时设立临时栈进行保存. 除此之外, 基于 X 求 Y 时, 算法 1 无需做任何修改, 相应的输出是每个顶点 v 的逆向拓扑序号 y_v.

例 3. 对图 3 中的 G_1 而言, 用算法 1 求 X 时, 顶点的入栈顺序为 *b*,*a*,*c*,*e*,*f*,*d*, 出栈顺序即拓扑顺序 X 为 *a*,*c*,*e*,*f*,*b*,*d*. 基于 X, 在求 Y 时, 入栈顺序为 *a*,*b*,*c*,*d*,*e*,*f*, 相应的出栈顺序及拓扑顺序 Y 为 *b*,*a*,*d*,*c*,*f*,*e*. 类似的, 对图 1 中的 G 而言, 应用算法 1, 可以得到 X, 其中每个顶点的拓扑序号为图 1 中该顶点的编号. 之后, 基于 X, 可得逆向 ST-order Y, 其中每个顶点在 Y 中的拓扑序号如图 1 中每个顶点旁边的数字所示. 在求解拓扑顺序的同时, 算法 1 也一并求得每个顶点的拓扑层. 拓扑层没有在图中标识出来, 但可从图 2 中 T_{opt} 看出. 例如, 顶点 v_1 的拓扑层是 0, 顶点 v_{14} 的拓扑层是 5.

4.1.2 构建最优生成树区间

最优树的构建基于定理 1. 给定 DAG G, 算法 2 以 DFS 的方式访问 G 中的顶点, 构造最优树, 同时对顶点设定相应的区间. 和正常的 DFS 不同, 处理完顶点 u 之后, 能从 u 扩展出来的树边对应的顶点需要满足拓扑层比 u 的拓扑层大 1.

算法 2 中 *visited*[·]是访问标志数组, *visitedNum* 是访问序号, S 为栈, I[·]是存储所有顶点区间的数组. 对于入度为 0 的顶点, 先标记被访问然后入栈(第 4-5 行), 并对该顶点的区间起始值 *start* 进行设定(即顶点的访问顺序, 第 6 行). 如果栈不空且栈顶元素有未被访问的出邻居, 且出邻居的拓扑层比栈顶元素大 1, 则访问该顶点(第 8-14 行); 否则如果栈顶元素的孩子都已访问则栈顶元素出栈, 并对出栈顶点元素的区间结束值 *end* 进行设定(第 16-17 行).

算法 2. OPT(G=(V, E))算法

输入: G=(V, E)

输出: 输出每个顶点 u 在最优生成树上的区间 I[u]

```
1   FOR EACH (u ∈ V) DO visited[u] ← FALSE
2   visitedNum ← 1
3   FOR EACH (u ∈ V and |in_G(u)| = 0) DO
4     visited[u] ← TRUE
5     push(S,u)
6     I[u].start ← visitedNum
7     visitedNum ← visitedNum+1
8     WHILE (S ≠ ∅) DO
9       v ← top(S)
10      IF (∃w∈out_G(v), visited[w]=FALSE∧l_w–l_v=1) THEN
11          visited[w] ← TRUE
12          push(S,w)
13          I[w].start ← visitedNum
14          visitedNum ← visitedNum+1
15      ELSE
16          v ← pop(S)
17          I[v].end ← visitedNum–1
```

例 4. 图 2 中的 T_{opt} 为根据算法 2 得到的图 1 中 G 的最优生成树, 其中每个顶点 v 的区间已标示在 v 旁边. 例如顶点 v_3 的区间为[7,11], 表示顶点 v_3 的 DFS 访问顺序为 7, 且以 v_3 为根的子树中, 顶点的最大 DFS 访问顺序为 11.

此外, 为了进一步增强可达查询的处理能力, 本文为每个给定的 DAG G 构建了反向最优生成树. 具体来说, 通过将 G 的边反向, 再应用算法 2 构建最优生成树, 并给树中每个顶点 u 设定相应的反向区间 I'[u]. 我们称边反向之前构造的最优树为正向最优树. 基于正反最优树中的顶点区间, 对于给定的查询 u?→v, 除了通过 I[v]⊂I[u]来确定 u 可达 v 之外, 也可通过 I'[u]⊂I'[v] 来确定 u 可达 v.

4.1.3 算法分析

定理 2. 给定 DAG G, 本文提出的索引构建算法的时间复杂度为 $O(|V|+|E|)$, 空间复杂度为 $O(|V|)$, 索引大小为 $O(|V|)$.

证明: 首先, 算法 1 在生成拓扑顺序 X 的过程中, 直接将当前访问顶点写回入邻居的邻接表中, 因此先访问顶点的拓扑号比后访问顶点的拓扑号小, 则入邻居的邻接表按拓扑号由小到大写入, 故最后每个顶点的邻接表就会变为拓扑升序. 在生成反向 ST-order Y 时, 算法 TOPO 无需和文献[7]一样使用优先队列, 只需顺序访问邻接表, 即可保证顶点按拓扑号由小到大进栈. 由于遍历图的时间复杂度为 $O(|V|+|E|)$, 所以生成两个互逆的拓扑顺序的时间复杂度也为 $O(|V|+|E|)$. 注意算法 1 在此过程中也同时得到了顶点的拓扑层. 算法 2 生成最优树只需对图进行一遍扫描, 相应的生成反向最优树的代价和正向最优树相同, 所以时间复杂度为 $O(|V|+|E|)$. 因此, 索引构建算法的时间复杂度为 $O(|V|+|E|)$.

其次, 算法 TOPO 以及 OPT 在执行过程中只需常量个和顶点数量规模一致的辅助空间, 因此其空间复杂度为为 $O(|V|)$.

由于每个顶点的标签大小为 5 个数字, 因此索引大小为 $O(|V|)$.

证毕.

和表 2 所示的现有方法相比, 本文提出的索引构建算法(用 OPT-R 表示)具有最小的时间和空间复杂度.

4.2 查询处理

基于第 3 节的讨论, 我们可以从正反两个方面来判定两个顶点之间是否可达, 如算法 3 所示. 对于给定的顶点对(u, v), 首先判断 u 和 v 在正反最优树上是否有祖先后代关系(第 1 行), 如有则 u 可达 v(第 2 行返回 TRUE); 否则在第 3 行判断 u 是否不可达 v, 如果是则返回 FALSE (第 7 行). 如果正反两方面都无法判定 u 是否可达 v, 则在第 4-6 行递归处理 u 的出邻居顶点, 直至得到最终的确定答案.

算法 3. OPT-R(u, v)算法

输入: G=(V, E)

输出: TRUE(u, v 可达) 或 FALSE(u, v 不可达)

```
1  IF (I[v] ⊂ I[u] or I'[u] ⊂ I'[v]) THEN
2     RETURN TRUE
3  IF (x_u<x_v and y_u<y_v and l_u<l_v) THEN
4     FOR EACH (w∈out_G(u)) DO
5        IF (OPT-R(w, v)) THEN
6           RETURN TRUE
7  RETURN FALSE
```

需要说明的是, OPT-R 属于在线搜索算法, 在最坏情况下会访问和 DFS 相同数量的顶点, 因而最坏情况下的查询代价依然是 $O(|V|+|E|)$. 然而, 借助于正反最优树区间、互逆拓扑顺序及拓扑层, OPT-R 可在常量时间判定大多数可达和不可达查询, 和现有的在线搜索类算法相比, 其优势体现在更高效的剪枝策略, 本文后续的实验结果也验证了 OPT-R 算法在回答可达性查询时的优势.

5 实验

5.1 实验环境

本文实验所用机器的基本配置为 AMD Athlon(tm) II X2 250 3000MHz CPU, 16 GB 内存, 操作系统为 Linux Ubuntu 12.04.4.

用于比较的算法包括最近提出的 FELINE[7]、GRAIL[6,22]、TF[16]、IP+[24]以及本文提出的 OPT-R 算法. 所有算法均采用 C++语言实现, 通过 G++ 4.6.3 编译器进行编译. 除本文算法, 已有算法的源代码均由文章作者提供.

5.2 数据集

本文所用数据集为 20 个真实数据集. 这些数据集经常出现在近期的研究工作中[6,7,22,24], 用于测试算法性能. 表 3 列出了本文所用数据集及相关统计信息, 其中 Human、Anthra、Agrocyc、Ecoo、Vchocyc 和 Mtbrv 来自 Ecocyc①, 都是描述基因组、转录调控、转运蛋白和代谢途径的数据集; Go②和 go_uniprot③为基因本体数据集; Xmark 和 Nasa 为 XML 文档; Arxiv④、citeseerx⑤、10citeseerx⑤、05cit-Patent⑥、10cit-Patent⑥为引用网络数据集; Dbpedia⑦为知识图; WikiTalk⑥是来自维基百科的有向无环图; LJ⑥是来自社会网络的有向无环图; web-Google⑥是来自 Google 网页的有向无环图; web-uk 是[35]收集的网络有向无环图. 其中 Human、Anthra、Agrocyc、Ecco、Vchocyc、Mtbrv、Xmark、Nasa、WikiTalk、LJ 以及 web-Google 是将原始有向图中的强联通分量进行压缩后得到的有向无环图.

表 3 数据集统计信息

数据集	$\|V\|$	$\|E\|$	平均度	顶点平均传递闭包大小
Human	38,811	39,576	1.02	10
Anthra	12,499	13,104	1.05	13
Agrocyc	12,684	13,408	1.06	14
Ecoo	12,620	13,350	1.06	15
Vchocyc	9,491	10,143	1.07	15
Mtbrv	9,602	10,245	1.07	15
Xmark	6,080	7,025	1.16	58
Nasa	5,605	6,537	1.17	20
Go	6,793	13,361	1.97	12
Arxiv	6,000	66,707	11.12	822
WikiTalk	2,281,879	2,311,570	1.01	18,522
LJ	971,232	1,024,140	1.05	206,908
web-Google	371,764	517,805	1.39	55,055
10cit-Patent	1,097,775	1,651,894	1.50	3
10citeseerx	770,539	1,501,126	1.95	25
05cit-Patent	1,671,488	3,303,789	1.98	6
citeseerx	6,540,401	15,011,260	2.30	15,457
Dbpedia	3,365,623	7,989,191	2.37	83,660
go_uniprot	6,967,956	34,769,339	4.99	18
web-uk	22,753,644	38,184,039	1.68	3,417,930

5.3 性能比较及分析

本文实验的评价指标包括：(1) 查询响应时间; (2) 构建索引的时间及索引大小. 生成不同可达比例的 100 万个查询的做法是: 首先求得每个数据集的可达点对, 然后通过随机算法选择相应比例的可

① ecocyc.org

② www.geneontology.org

③ www.uniprot.org

④ arxiv.org

⑤ citeseerx.ist.psu.edu

⑥ snap.stanford.edu

⑦ dbpedia.org

达查询, 剩余不可达查询通过随机算法从相应数据集的顶点对中抽取, 若抽取的查询是不可达查询, 则保留, 否则重新抽取, 直到选中 100 万个查询为止. 当运行时间超过 24 小时或超过内存限制(16GB)时在结果中用“-”表示. 以下实验中 GRAIL 算法的参数 k=5, IP$^+$算法的参数 k=5, h=5, μ=100, 这些参数值均为相应文章建议的最优参数.

5.3.1 查询时间

可达性查询处理作为图的基本操作之一, 其查询性能的变化会因级联放大效应对上层应用问题的处理能力产生显著影响. 表 4 和表 5 为可达比例为 20%、40%、60%和 80%的查询时间(给定可达查询比例，加粗数字表示相应的查询时间的最小值). 通过对表 4 和表 5 的实验结果进行比较分析, 可知本文提出的 OPT-R 算法相对来说总体效率最高, 且表现最稳定. 在所有给出的 20 个数据集上, OPT-R 在 17 个数据集上表现最好. 在剩余的 3 个数据集上, OPT-R 的运行时间和最好结果相比, 差别不大. 同时, OPT-R 对可达查询比例的变化不敏感, 查询处理性能没有剧烈变化的情况出现, 表现更稳定.

相比较而言, 已有的在线搜索算法中, 由于 DFS 和 BFS 没有使用任何剪枝策略, 其性能取决于相应数据集中顶点的平均传递闭包的大小 (表 3 第 5 列). 当顶点的平均传递闭包变大时, 二者的查询处理性能迅速变差, 例如对于 WikiTalk、LJ、web-Google、citeseerx、Dbpedia 以及 web-uk 数据集, DFS 和 BFS 都不能在有限时间内处理完相应的查询. 与 DFS 和 BFS 相比, 虽然 GRAIL 可以在 web-Google、citeseerx 以及 Dbpedia 上得到结果, 但依然不能在有限时间内从 WikiTalk、LJ 以及 web-uk 数据集上处理完所有查询, 原因在于其剪枝能力有限. 和 GRAIL 相比, IP$^+$和 FELINE 的主要优势在于快速回答不可达查询, 因而查询处理的效率得到了明显提升, 但和本文提出的 OPT-R 算法相比, 同样存在查询效率相对较差, 且性能不够稳定的问题. 例如, 在 Xmark 数据集上, 当可达查询比例从 20%增加到 80%时, IP$^+$和 FELINE 的查询处理时间均增加了 78ms, 而 OPT-R 增加了 16ms. 对 Arxiv 数据集而言, 二者的查询处理时间分别增加了 319ms 和 128ms, 而 OPT-R 增加了 58ms. 对 web-uk 数据集而言, 相同条件下, IP$^+$和 FELINE 的查询处理时间分别增加了 1,315ms 和 1,277ms, 而 OPT-R 增加了 312ms. 由于本文的编码方案和 FELINE 最相似, 这里仅通过表 6 对 FELINE 和 OPT-R 在常量时间内回答可达查询和不可达查询时的数量(即无需遍历图即可回答的查询数量)进行比对来说明二者性能差异的原因. 表 6 的第 2-4 (5-7) 列展示的是当可达查询比例为 20% (80%) 时, OPT-R 比 FELINE 在 3 个数据集上常量时间内多回答的查询数量, 该结果说明 OPT-R 在可达查询和不可达查询两方面使用的剪枝策略比 FELINE 更高效, 因而相对来说整体效率更高, 且表现更稳定. 值得一提的是, 在表 6 所示的 3 个数据集上, 和 FELINE 所使用的生成树相比, OPT-R 所使用的最优树能够覆盖的可达查询分别是 FELINE 的 1.5 倍、14.3 倍和 157.1 倍, 因而可以常量时间回答的可达查询数量显著多于 FELINE(见表 6). 和 FELINE 相比, OPT-R 在所有数据集上都更高效, 对任意可达查询比例而言, OPT-R 在至少 12 个数据集上查询处理速度比 FELINE 快 2 倍以上.

表 4 可达比例为 20%和 40%的查询时间(ms)

数据集	20%可达查询							40%可达查询						
	DFS	BFS	GRAIL	IP$^+$	FELINE	OPT-R	TF	DFS	BFS	GRAIL	IP$^+$	FELINE	OPT-R	TF
Human	487	107	1,225	26	36	***24***	86	858	167	2,624	47	50	***30***	125
Anthra	276	92	668	26	38	***17***	67	421	116	1,289	41	51	***21***	94
Agrocyc	303	98	669	26	38	***17***	75	447	124	1,210	41	56	***22***	114
Ecoo	303	104	651	26	39	***17***	35	454	129	1,245	43	56	***21***	35
Vchocyc	302	103	660	26	34	***16***	29	435	120	1,366	44	48	***21***	30
Mtbrv	314	108	610	26	33	***16***	21	459	129	1,183	41	47	***21***	23
Xmark	697	319	434	54	116	***24***	43	776	288	792	78	134	***30***	51
Nasa	384	184	315	52	76	***21***	33	390	156	528	80	97	***25***	35
Go	373	176	432	64	239	50	***37***	396	154	735	96	267	57	***40***
Arxiv	84,009	20,529	1,402	324	876	***321***	502	71,765	15,842	1,837	414	881	***351***	458
WikiTalk	-	-	-	108	159	***90***	113	-	-	-	172	199	***127***	146

LJ	-	-	-	120	184	***109***	114	-	-	-	189	284	155	***137***
web-Google	-	-	561,428	147	205	***119***	141	-	-	1,691,458	218	256	***152***	155
10cit-Patent	439	250	1,055	208	281	***157***	229	497	268	1,614	316	367	***207***	254
10citeseerx	5,590	1,427	1,910	***276***	670	278	409	7,769	1,180	3,542	436	848	***371***	572
05cit-Patent	1,118	574	1,254	***281***	474	286	298	1,190	554	1,993	426	611	369	***334***
citeseerx	-	-	123,544	393	791	***341***	587	-	-	366,572	634	1,224	***499***	813
Dbpedia	-	-	673,346	285	354	***198***	955	-	-	2,023,913	425	495	***274***	1,672
go_uniprot	1,315	724	1,192	243	359	***196***	7,580	1,181	663	1,629	423	582	***314***	14,753
web-uk	-	-	-	702	913	***290***	-	-	-	-	1,188	1,326	***420***	-

表 5 可达比例为 60%和 80%的查询时间(ms)

数据集	60%可达查询							80%可达查询						
	DFS	BFS	GRAIL	IP+	FELINE	OPT-R	TF	DFS	BFS	GRAIL	IP+	FELINE	OPT-R	TF
Human	1,189	227	2,624	56	62	***33***	171	1,601	292	3,522	73	74	***35***	212
Anthra	571	142	1,289	57	66	***25***	126	715	168	1,616	71	82	***29***	159
Agrocyc	598	149	1,210	57	72	***27***	156	754	177	1,543	73	88	***32***	199
Ecoo	604	155	1,245	58	70	***26***	37	751	180	1,561	73	97	***31***	39
Vchocyc	568	141	1,366	57	64	***25***	33	700	159	1,629	73	80	***30***	35
Mtbrv	603	150	1,183	58	59	***21***	22	746	171	1,492	73	75	***21***	22
Xmark	847	249	792	108	160	***35***	63	919	213	986	132	194	***40***	76
Nasa	399	127	528	108	106	***29***	38	407	99	627	140	133	***32***	42
Go	424	133	735	134	261	64	***43***	448	111	888	169	325	71	***48***
Arxiv	60,221	11,186	1,837	588	902	***371***	420	48,486	6,439	2,099	643	1,004	***379***	381
WikiTalk	-	-	-	237	238	***165***	172	-	-	-	302	273	***200***	203
LJ	-	-	-	258	301	207	***158***	-	-	-	328	341	248	***184***
web-Google	-	-	1,691,458	290	304	***160***	169	-	-	2,267,895	363	352	***181***	187
10cit-Patent	556	286	1,614	425	447	***256***	288	617	303	1,902	533	525	***307***	321
10citeseerx	10,004	960	3,542	603	1,027	***448***	701	11,968	696	4,364	760	1,225	***528***	863
05cit-Patent	1,270	538	1,993	568	747	454	***357***	1,340	518	2,349	711	871	541	***393***
citeseerx	-	-	366,572	878	1,642	***645***	935	-	-	489,971	1,131	2,054	***803***	1,127
Dbpedia	-	-	2,023,913	561	632	***347***	2,397	-	-	2,706,411	701	760	***423***	3,108
go_uniprot	1,043	602	1,629	603	790	***431***	22,525	908	540	1,858	765	1,006	***549***	29,994
web-uk	-	-	-	1,659	1,755	***526***	-	-	-	-	2,055	2,190	***602***	-

表 6 常量时间内 OPT-R 比 FELINE 多回答的查询数量

数据集	查询数量差(20%可达)			查询数量差(80%可达)		
	不可达	可达	总数	不可达	可达	总数
Xmark	12,151	62,778	74,929	2,987	251,413	254,400
Arxiv	8,261	51,475	59,736	2,243	205,780	208,023
web-uk	44,542	83,802	128,344	11,085	362,501	373,586

和标签类方法 TF 相比, TF 虽然在 Go、LJ 以及 05cit-Patent 数据集上表现最好, 但其性能随着数据集的变化表现不够稳定, 当可达查询比例为 20%时, TF 在 Dbpedia 和 go_uniprot 数据集上所需时间分别是 OPT-R 的 4.8 倍和 38.9 倍. 当可达查询比例为 80%时, TF 所需时间分别是 OPT-R 的 7.3 倍和 54.6 倍. 同时, TF 在 web-uk 上无法处理所有的查询.

5.3.2 索引构建时间及索引大小

表 7 展示了不同算法在 20 个数据集上构建索引的时间和索引大小. 在索引构建时间方面, 如表 7 所示, 除了 WikiTalk 和 LJ 两个数据集, 本文提出的 OPT-R 算法构建索引的时间都是最短的, 而且在一些数据集中优势明显: 例如对于 Arxiv 数据集, OPT-R 的构建索引时间比 TF 快 2,558 倍, 比 GRAIL 快 10 倍; 对于 web-uk 数据集, OPT-R 的构建索引时

间比 GRAIL 快 30 倍，比 IP+快 4.8 倍，而 TF 在此数据集上不能构建索引.

在索引大小方面, FELINE 在 14 个数据集上取得了最小值，和 FELINE 相比, OPT-R 同时使用了正反两个最优树区间，因而每个顶点的标签比 FELINE 多两个数字，因而索引大小比 FELINE 大 40%. 从表 7 可以看出，在线类算法的索引大小都相差不大，和图中顶点的数量成线性相关关系. 相比较而言，标签类算法 TF 的索引大小极其不稳定. 例如在 Arxiv 和 citeseerx 数据集上索引过大，且不能对 web-uk 构建索引.

表 7 索引构建时间及索引大小

数据集	索引构建时间(ms)					索引大小(MB)				
	GRAIL	IP+	FELINE	OPT-R	TF	GRAIL	IP+	FELINE	OPT-R	TF
Human	121.46	22.65	14.12	***5.47***	74.81	1.48	1.04	0.74	1.04	***0.58***
Anthra	29.12	6.60	4.29	***1.30***	31.55	0.48	0.34	***0.24***	0.33	0.30
Agrocyc	29.10	6.79	4.40	***1.24***	18.28	0.48	0.34	***0.24***	0.34	0.37
Ecoo	29.48	6.94	4.39	***1.26***	20.04	0.48	0.34	***0.24***	0.34	0.38
Vchocyc	21.06	5.09	3.22	***0.82***	30.34	0.36	0.26	***0.18***	0.25	0.31
Mtbrv	21.70	5.28	3.20	***0.82***	10.37	0.37	0.26	***0.18***	0.26	0.07
Xmark	13.62	3.63	2.46	***0.65***	12.43	0.23	0.19	***0.12***	0.16	***0.12***
Nasa	12.21	3.15	2.20	***0.66***	11.73	0.21	0.15	***0.11***	0.15	0.12
Go	17.91	5.55	3.72	***1.20***	21.45	0.26	0.19	***0.13***	0.18	0.18
Arxiv	32	19	6	***3***	7,673	0.23	0.26	***0.11***	0.16	14.66
WikiTalk	***87***	1,629	1,006	361	2,379	87	95	44	61	***9***
LJ	***37***	719	467	169	1,146	37	28	19	26	***4***
web-Google	1,748	385	251	***113***	918	14	10	***7***	10	27
10cit-Patent	5,276	1,498	1,094	***713***	3,152	42	20	21	29	***8***
10citeseerx	2,793	999	557	***365***	4,717	29	18	***15***	21	45
05cit-Patent	9,755	3,019	2,133	***1,720***	7,920	64	35	32	45	***26***
citeseerx	33,840	10,951	6,474	***5,496***	102,444	249	126	***125***	175	1,523
Dbpedia	23,292	5,094	3,280	***2,047***	15,680	128	81	64	90	***52***
go_uniprot	67,664	11,968	7,196	***3,839***	69,428	266	158	***133***	186	431
web-uk	147,831	23,368	12,473	***4,852***	-	868	729	***434***	608	-

综合来看, OPT-R 算法同时从判断可达和不可达两方面加速查询的处理，不但运行速度快，而且对可达查询比例的变化不敏感，查询处理性能表现更稳定. 同时, OPT-R 因其在索引构建方面具有线性的时间复杂度，在索引构建方面表现最佳. 在索引大小方面，因单个顶点标签中的数据量稍多于 FELINE，因而 OPT-R 的索引规模比 FELINE 稍大. 考虑到实际中不断增长的内存空间和 OPT-R 具有线性的空间复杂度，可以认为 OPT-R 相对于 FELINE 的索引增长量是可承受的.

6 结论

针对已有的在线搜索算法在可达查询比例增大时效率下降明显及性能不稳定的问题，本文提出优化的 OPT-R 算法. OPT-R 算法的优势体现在两方面. 首先，通过构建最优生成树, OPT-R 可以在常量时间回答更多的可达查询. 其次，通过使用基于栈的互逆拓扑顺序, OPT-R 可以在常量时间回答更多的不可达查询. OPT-R 具有线性的索引构建时间复杂度 $O(|V|+|E|)$,线性的空间复杂度 $O(|V|)$以及线性的索引规模 $O(|V|)$. 实验结果显示，在不降低索引时间且索引规模增长不大的情况下，对所有 20 个数

据集和不同比例的可达查询而言, OPT-R 在 17 个数据集上表现最好. 同时, OPT-R 对可达查询比例的变化不敏感, 查询处理性能相对来说没有剧烈变化的情况出现, 表现更稳定. 当查询中可达查询比例增加时, OPT-R 的优势更明显.

参 考 文 献

[1] Zhang Y, Liu Y B, Xiong G, Jia Y, Liu P, Guo L. Survey on succinct representation of graph data. Journal of Software, 2014, 25(9): 1937−1952 (in Chinese)
(张宇,刘燕兵,熊刚,贾焰,刘萍,郭莉. 图数据表示与压缩技术综述. 软件学报, 2014, 25(9): 1937−1952)

[2] Ding Y, Zhang Y, Li Z H, Wang Y. Research and advances on graph data mining. Journal of Computer Applications, 2012, 32(1): 182–190 (in Chinese)
(丁悦,张阳,李战怀,王勇. 图数据挖掘技术的研究与进展. 计算机应用, 2012, 32(1): 182–190)

[3] Chen L, Gupta A, Kurul M E. Stack-based algorithms for pattern matching on DAGs//Proceedings of International Conference on Very Large Data Bases (VLDB). Trondheim, Norway, 2005: 493–504

[4] Cheng J F, Yu J X, Ding B L. Cost-based query optimization for multi reachability joins//Proceedings of 12th International Conference on Database Systems for Advanced Applications (DASFAA). Bangkok, Thailand, 2007: 18–30

[5] Tarjan R E. Depth-First search and linear graph algorithms. SIAM Journal on Computing, 1972, 1(2):146–160

[6] Yildirim H, Chaoji V, Zaki M J. GRAIL: scalable reachability index for large graphs. The Proceedings of the VLDB Endowment (PVLDB), 2010, 3(1):276–284

[7] Veloso R R, Cerf L, Junior W M, Zaki M J. Reachability queries in very large graphs: a fast refined online search approach//Proceedings of 17th International Conference on Extending Database Technology (EDBT). Athens, Greece ,2014: 511–522

[8] Jagadish H V. A compression technique to materialize transitive closure. ACM Transactions on Database Systems (TODS), 1990, 15(4):558–598

[9] Chen Y J, Chen Y B. An efficient algorithm for answering graph reachability queries//Proceedings of IEEE International Conference on Data Engineering (ICDE). Cancun, Mexico, 2008: 893–902

[10] Jin R M, Xiang Y, Ruan N, Wang H X. Efficiently answering reachability queries on very large directed graphs//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Vancouver, Canada, 2008: 595–608

[11] Jin R M, Ruan N, Xiang Y, Wang H X. Path-tree: an efficient reachability indexing scheme for large directed graphs. ACM Transactions on Database Systems (TODS), 2011, 36(1): 7:1–7:44

[12] Agrawal R, Borgida A, Jagadish H V. Efficient management of transitive relationships in large sata and knowledge bases//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Portland, USA, 1989: 253–262

[13] Wang H X, He H, Yang J, Yu P S, Yu J X. Dual labeling: answering graph reachability queries in constant time//Proceedings of IEEE International Conference on Data Engineering (ICDE). Atlanta, USA, 2006: 75

[14] Cheng J F, Yu J X, Lin X M, Wang H X, Yu P S. Fast computation of reachability labeling for large graphs//Proceedings of 10th International Conference on Extending Database Technology (EDBT). Munich, Germany, 2006: 961–979

[15] Jin R M, Xiang Y, Ruan N, Fuhry D. 3-HOP: a high-compression indexing scheme for reachability query//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Providence, USA, 2009: 813–826

[16] Cheng J, Huang S L, Wu H X, Fu W C. TF-Label: a topological-folding labeling scheme for reachability querying in a large graph//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). New York, USA, 2013: 193–204

[17] Jin R M, Wang G. Simple, fast, and scalable reachability oracle. The Proceedings of the VLDB Endowment (PVLDB), 2013, 6(14):1978–1989

[18] Yano Y, Akiba T, Iwata Y, Yoshida Y. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths//Proceedings of The Conference on Information and Knowledge Management (CIKM). San Francisco, USA, 2013:1601–1606

[19] Schaik S J V, Moor O D. A memory efficient reachability data structure through bit vector compression//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Athens, Greece, 2011: 913–924

[20] Chen Y J, Chen Y B. Decomposing DAGs into spanning trees: a new way to compress transitive closures//Proceedings of IEEE International Conference on Data Engineering (ICDE). Hannover, Germany, 2011: 1007–1018

[21] Silke T, Leser U. Fast and practical indexing and querying of very large graphs//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Beijing, China, 2007: 845–856

[22] Yildirim H, Chaoji V, Zaki M J. GRAIL: a scalable index for reachability queries in very large graphs. The International Journal on Very Large Data Bases (VLDB Journal), 2012, 21(4):509–534

[23] Seufert S, Anand A, Bedathur S J, Weikum G. FERRARI: flexible and efficient reachability range assignment for graph indexing//Proceedings of IEEE International Conference on Data Engineering (ICDE). Brisbane, Australia, 2013: 1009–1020

[24] Wei H, Yu J X, Lu C, Jin R M. Reachability querying: an independent permutation labeling approach. The Proceedings of the

VLDB Endowment (PVLDB), 2014, 7(12):1191–1202

[25] Jin R M, Ruan N, Dey S, Yu J X. SCARAB: scaling reachability computation on large graphs//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Scottsdale, USA, 2012: 169–180

[26] Cheng J, Shang Z C, Cheng H, Wang H X, Yu J X. K-Reach: who is in your small world. The Proceedings of the VLDB Endowment (PVLDB), 2012, 5(11):1292–1303

[27] Zhu A D, Lin W Q, Wang S B, Xiao X K. Reachability queries on large dynamic graphs: a total order approach//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Snowbird, USA, 2014: 1323–1334

[28] Simon K. An improved algorithm for transitive closure on acyclic digraphs. Theoretical Computer Science (TCS), 1988, 58:325–346

[29] Wu H H, Huang Y Z, Cheng J, Li J F, Ke Y P. Reachability and time-based path queries in temporal graphs//Proceedings of IEEE International Conference on Data Engineering (ICDE). Helsinki, Finland, 2016: 145–156

[30] Gurajada S, Theobald M. Distributed set reachability//Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD). Francisco, USA, 2016: 1247–1261

[31] Cheng J, Shang Z C, Cheng H, Wang H X, Yu J X. Efficient processing of k-hop reachability queries. The International Journal on Very Large Data Bases (VLDB Journal), 2014, 23(2): 227–252

[32] Zhou J F, Chen W, Fei C P, Chen Z Y. BiRch: a bidirectional search algorithm for k-step reachability queries. Journal on Communications, 2015, 36(8): 50−60 (in Chinese)
(周军锋, 陈伟, 费春苹, 陈子阳. BiRch:一种处理k步可达性查询的双向搜索算法. 通信学报, 2015, 36(8): 50−60)

[33] Jiang H F, Wang W, Lu H J, Yu J X. Holistic twig joins on indexed XML documents//Proceedings of 29th International Conference on Very Large Data Bases (VLDB). Berlin, Germany, 2003: 273–284

[34] Evgenios M K, Ioannis G T. Overloaded orthogonal drawings//Proceedings of Graph Drawing- 19th International Symposium. Eindhoven, The Netherlands, 2011: 242–253

[35] Boldi P, Santini M, Vigna S. A large time-aware web graph. SIGIR Forum, 2008, 42(2):33–38

CHEN Zi-Yang, born in 1973, Ph.D., professor, Ph.D. supervisor, Distinguished Professor of Shanghai Lixin University of Accounting and Finance. His research interests include database theory and techniques.

CHEN Wei, born in 1980, Ph.D. candidate, associate professor. Her research interests include query processing on graph data.

LI Na, born in 1990, M.S. candidate. Her research interests include query processing on graph data.

ZHOU Jun-Feng, born in 1977, Ph.D., professor. His research interests include query processing on XML and semi-structured data.

Background

Given a directed graph, a reachability query $u?\rightarrow v$ asks whether a node v is reachable from a node u. Answering reachability queries is one of the fundamental graph operations and has been extensively studied. Its applications include biological networks, social networks, the Semantic Web, ontology, transportation networks, program workflows, etc. Due to its importance and the emergence of large graphs, it is still a challenging task for reachability queries to be answered faster with less index size and index construction time offline.

Considering that existing algorithms on reachability query processing are inefficient when the number of reachable queries increases, we proposed an optimized algorithm, namely OPT-R, to accelerate reachability query processing. We first propose optimal spanning tree, which contains the maximum number of reachable queries compared with other spanning trees, such that OPT-R can answer more reachable queries in constant time on average. The main idea behind the optimal spanning tree is that to make a spanning tree covering the maximal number of reachable node pairs, then each node in the spanning tree should have the most number of ancestors. Based on this idea, we take the topological level of each node as its tree height, and construct a spanning tree in linear time, for which each node satisfies that its tree height equals its topological level in the given graph. We then proposed the *stack*-based topological order, namely ST-Order, and the *reverse* ST-Order to help OPT-R answer more unreachable queries in constant time. We further proposed efficient index construction algorithms to get the optimal spanning tree and the two ST-Orders that are reverse to each other. We conducted extensive experimental study based on 20 real datasets. The

experimental results show that OPT-R is more efficient in index construction and query processing.

This research was partially supported by the grants from National Science and Technology Major Project (No. 61472339, 61572421, 61272124).