软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(3):515−536 [doi: 10.13328/j.cnki.jos.005692]

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

基于规则的最短路径查询算法*

李忠飞[1,2,3], 杨雅君[1,2,3], 王 鑫[1,3]

[1](天津大学 智能与计算学部,天津 300354)

[2](数字出版技术国家重点实验室,北京 100871)

[3](天津市认知计算与应用重点实验室,天津 300354)

通讯作者: 王鑫, E-mail: wangx@tju.edu.cn

摘 要: 最短路径查询是图数据管理中非常重要的一类问题.研究了基于规则的最短路径查询,它是一类特殊的最短路径查询问题.给定起点和终点,基于规则的最短路径查询是指找到一条从起点到终点的最短路径,使得此路径经过用户指定点集中的所有点,并且某些点的访问顺序满足一定的偏序规则.该问题被证明是一个 NP-hard 问题.目前已有的工作侧重于空间数据集(两点之间的最短距离用欧氏距离表示)上基于规则的最短路径问题,它采用穷举的方式列出所有满足规则的路径,然后选择长度最小的路径作为问题的解.然而在实际的道路交通网中,两点之间的距离等于两点之间的最短路径的长度,它往往大于两点之间的欧氏距离;此外,采用穷举的方式会造成大量重复的计算.因此,设计了一种前向搜索算法以及一些优化技术来求解该问题.最后,在不同的真实数据集上设计了大量的实验来验证算法的有效性.实验结果表明,该算法可以快速给出问题的解,而且算法的效率在很大程度上超过了现有的算法.

关键词: 图数据;最短路径;规则;最优子排列;分层收缩

中图法分类号: TP311

中文引用格式: 李忠飞,杨雅君,王鑫.基于规则的最短路径查询算法.软件学报,2019,30(3):515−536. http://www.jos.org.cn/1000-9825/5692.htm

英文引用格式: Li ZF, Yang YJ, Wang X. Rule based shortest path query algorithm. Ruan Jian Xue Bao/Journal of Software, 2019,30(3):515−536 (in Chinese). http://www.jos.org.cn/1000-9825/5692.htm

Rule Based Shortest Path Query Algorithm

LI Zhong-Fei[1,2,3], YANG Ya-Jun[1,2,3], WANG Xin[1,3]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300354, China)

[2](State Key Laboratory of Digital Publishing Technology, Beijing 100871, China)

[3](Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin 300354, China)

Abstract: The shortest path query is very important in the related applications of graph data. The problem studied in this work is the rule-based shortest path query, which is a special kind of shortest path problem. Given the starting vertex and the ending vertex, the rule-based shortest path query is that finding a shortest path from the starting vertex to the ending vertex, which passes through all vertices in the user-specified set, and the access order of some vertices meets certain partial order rules. It has proved that this problem is NP-hard problem. The existing work focuses on the rule-based shortest path problem on spatial datasets (the shortest distance between

* 基金项目: 国家自然科学基金(61402323, 61572353, U1736103); 数字出版技术国家重点实验室开放课题; 天津市自然科学基金(17JCYBJC15400)

Foundation item: National Natural Science Foundation of China (61402323, 61572353, U1736103); Opening Project of State Key Laboratory of Digital Publishing Technology; Natural Science Foundation of Tianjin (17JCYBJC15400)

本文由智能数据管理与分析技术专刊特约编辑樊文飞教授、王国仁教授、王朝坤副教授推荐.

收稿时间: 2018-07-19; 修改时间: 2018-09-20; 采用时间: 2018-11-01

two vertices is expressed by Euclidean distance), which exhaustively lists all paths those satisfy the rule, and selects the path with the smallest length as the solution to the problem. However, in an actual road network, the distance between two vertices is equal to the length of the shortest path between two vertices, which is often greater than the Euclidean distance between two vertices. In addition, using an exhaustive approach always results in a lot of repetitive calculations. Based on this, this study designs a forward search algorithm and some optimization techniques to solve such problems. Finally, this study designs a large number of experiments on different real datasets to verify the effectiveness of the algorithms. The experimental results show that the algorithms described in this paper can quickly give the solution to the problem, and the efficiency of the algorithms greatly exceeds the existing algorithms.

Key words: graph data; shortest path; rule; optimal sub-permutation; contraction hierarchy

近年来,随着信息技术和互联网的飞速发展,图数据作为一种重要的数据模型变得愈加重要.在很多领域,图数据刻画了不同实体之间的相互关系,例如社交网络[1,2]、道路交通网[3]、生物信息网[4]、计算机网络[5]和Web网络[6]等.

其中,最短路径查询是图数据管理中的一类非常重要的研究问题.在社交网络中,每个人都构成了图中的一个顶点,人与人之间的联系则形成了边.社交网络中的最短路径是网络顶点影响力评判的重要因素,小世界网络中,直径的计算一般也是通过计算最长最短路径得到的[7];在Web网络中,数据转发时每个路由器使用路由协议和链路状态信息来识别从自己到其他路由器的最短路径,网络拓扑通常随时间而变化,因此,一个高效的最短路径算法在路由计算中显得尤为重要[8];在道路交通网中,经常需要计算两个地点之间的最短路径,有时因为一些特殊的需求还要计算最小的前 k 条最短路径[9−11].

在实际应用中,用户往往需要查询带有约束条件的最短路径.例如,某游客去一座城市旅行,他计划首先前往某餐馆就餐,然后分别参观景点 A~景点 C,最后返回酒店.特别地,该游客计划在参观景点 B 之前先参观景点 A.因此,如何设计一条满足用户约束且代价最小的观光路径成为一个重要的问题.在该例中,道路交通网被建模为一张图 $G(V,E)$,则上述问题可形式化描述为:给定起点 v_s 和终点 v_e,寻找一条从 v_s 到 v_e 的最短路径,使得此路径经过用户指定点集 $V_s \subseteq V$ 中的所有点,且某些点的访问要满足一定的先后顺序.在本文中,该问题被称为基于规则的最短路径查询问题.

目前,存在少量工作研究了最优路径查询问题optimal route queries(ORQ).ORQ将图 G 中全部顶点划分为多个不同的类别,用户查询时,给出起点 v_s 和一个访问顺序图 G_Q.这里,G_Q 是一个有向无环图,G_Q 中的每个点都对应于 G 中的一个类别信息,G_Q 中每条有向边(c,c')表示 G 中类别 c 优先于类别 c'访问.查询结果是一条以 v_s 为起点且满足 G_Q 的最优路径.例如,某游客计划去餐馆、酒吧和电影院,每个类别都有多个具体的地点可供选择,而且游客希望在去酒吧之前要先去餐馆吃饭,因此,他需要制定一条路线使得在满足约束条件的前提下路线的总距离最短.目前,解决ORQ问题的精确算法的基本思想是:首先计算 G_Q 中所有类别的全排列(每个类别在排列中的顺序代表此类的访问顺序),并删除访问顺序不满足约束条件的排列;然后对于每一个排列,依次从排列的每个类中选择一个点构成一条路径;最终枚举出满足约束条件的所有路径,并返回具有最短距离的路径.然而这些方法在面对基于规则的最短路径问题时,主要存在以下两个缺点:(1) 针对基于规则的路径查询问题,对图 G 进行划分所得的每个类仅包含一个顶点,即任意两点对应的类都不相同,已有的算法求解此类问题相当于枚举出所有满足约束条件的排列,然而大部分排列对于求解最短路径来说是冗余的;(2) 目前已有的解决ORQ问题的算法面向的是空间数据库,这些算法均通过计算两点之间的欧氏距离加速查询,然而本文所解决的问题是基于普通的图模型,两点间的最短距离即为最短路径的长度,而存储全部顶点对之间的最短距离空间开销过高.因此,本文设计了一种前向扩展算法来快速求解基于规则的最短路径问题,其主要思想是,尽可能早地过滤掉不能构成最短路径的子路径.真实数据集上的实验结果证明了本文提出的算法的效率远远优于传统ORQ算法.

本文的主要贡献如下:

(1) 提出了广义规则树的概念,设计了生成广义规则树的算法,并利用广义规则树来判断算法是否找到一条基于规则的最短路径;

(2) 设计了基于最优子路径的前向扩展算法,该算法可以快速求解基于规则的最短路径问题,并设计了前

向扩展算法的改进算法——基于最短优先策略的前向扩展算法;

(3) 在真实的数据集上设计了大量的实验,并与已有的性能最好的算法比较,实验结果验证了本文算法的有效性.

本文第 1 节给出基于规则的最短路径查询问题的形式化定义,并证明此类问题是 NP-hard 问题.第 2 节介绍广义规则树的概念,并设计生成广义规则树的算法.第 3 节介绍一种图数据的预处理技术,可以用来快速求解两点之间的最短路径.第 4 节和第 5 节分别介绍前向扩展算法以及基于最短优先策略的前向扩展算法,并分别对它们的算法复杂度进行分析.第 6 节在真实的数据集上验证本文算法的高效性.第 7 节介绍相关工作.最后一节对全文进行总结.

1 问题定义

有向加权图可以表示为 $G(V,E,w)$(简称 G),V 表示图 G 中全部顶点的集合,E 表示全部边的集合.任意一条有向边 $e\in E$ 可以表示为 $e=(v_i,v_j)$,其中,$v_i,v_j\in V$,e 称为 v_i 的出边或者 v_j 的入边,$v_j(v_i)$被称为 $v_i(v_j)$的出边(入边)邻居.w 是一个权重函数,并且对图 G 中的每条边都赋予一个非负的权重,本文使用 $w_{i,j}$ 来表示有向边$(v_i,v_j)\in E$ 的权重,即 $w_{i,j}=w(v_i,v_j)$.图 G 中的路径 p 是一个顶点序列,即 $p=(v_1,v_2,\ldots v_k)$,其中,$(v_i,v_{i+1})(1\leqslant i\leqslant k-1)$是图 G 中的一条有向边,路径 p 的权重用 $w(p)$表示,表示 p 中全部有向边的权重之和,即 $w(p)=\sum_{1\leqslant i\leqslant k-1} w_{i,i+1}$.路径 p 是一条简单路径当且仅当 p 中没有重复的顶点.对于无向图,一条无向边(v_i,v_j)等价于两条有向边(v_i,v_j)和(v_j,v_i),因此,本文提出的方法也可应用到无向图上的同类问题.

本文主要研究基于规则的最短路径查询问题,首先介绍什么是路径查询规则,然后给出基于规则的路径定义.本文中,路径规则用$\mathcal{R}$表示,被定义为二元组$\mathcal{R}=(I,R)$,包含两个元素.

(i) I 是 V 的一个顶点子集,即 $I\subseteq V$;

(ii) R 是一组偏序关系$\langle v_i,v_j\rangle$的集合,其中,$v_i,v_j\in I$.$\langle v_i,v_j\rangle$表示 $v_i\prec v_j$.

需要注意的是:偏序关系集 R 中不存在环,即 R 中不存在一个偏序关系子集$\{\langle v_1,v_2\rangle,\ldots,\langle v_{k-1},v_k\rangle,\langle v_k,v_1\rangle\}$,其中,顶点 $v_1,v_2,\ldots,v_k\in I$.下面给出基于规则的路径定义.

定义 1.1(基于规则的路径). 给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}=(I,R)$,如果路径 p 满足以下两个条件,则被称为由 v_s 到 v_e 的基于规则$\mathcal{R}$的路径.

(1) 对每一个顶点 $v_i\in I$,都有 $v_i\in p$;

(2) 对每一个偏序关系$\langle v_i,v_j\rangle\in R$(或者 $v_i\prec v_j$),路径 p 中都存在一个从 v_i 到 v_j 的子路径 $v_i\rightsquigarrow v_j$.

在定义 1.1 中,条件(1)是指:若路径 p 基于规则$\mathcal{R}=(I,R)$,则 p 需要包含 I 中的所有点;条件(2)是指:对 R 中的任意偏序关系 $v_i\prec v_j$,在路径 p 中都存在一条从顶点 v_i 到顶点 v_j 的子路径.

给定图 G 中的两个顶点 v_s,v_e 以及规则$\mathcal{R}$,从 v_s 到 v_e 的基于规则$\mathcal{R}$的最短路径表示为 $p^*_{s,e,\mathcal{R}}$,是指从 v_s 到 v_e 的基于规则$\mathcal{R}$的所有路径中权重 $w(p)$最小的路径.表 1 列出了本文常用的符号.

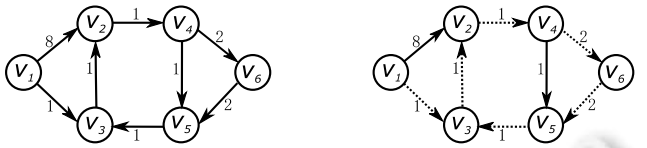
Table 1 List of notations

表 1 符号列表

符号	含义	符号	含义
$G(V,E,w)$	有向加权图	v_s,v_e	起点,终点
$w_{i,j},w(p)$	边(v_i,v_j)的权重,路径 p 的权重	$\mathcal{R}$	规则
$p^*_{s,e,\mathcal{R}}$	基于规则$\mathcal{R}$的最短路径	I_R	规则点集
$T_{\mathcal{R}}$	广义规则树	π	点集的排列
$p\|_\pi$	排列π对应的路径	V_{π_l}	子排列π_l对应的点集

例 1.1:给定有向图 $G(V,E,w)$,如图 1(a)所示,已知规则$\mathcal{R}=(I,R)$,$I=\{v_2,v_4,v_5,v_6\}$,$R=\{v_2\prec v_4,v_2\prec v_5\}$,其中,起点为 v_1,

终点为 v_3,可得此图基于规则$\mathcal{R}$的最短路径为 $p^*_{s,e,\mathcal{R}}=(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$,如图 1(b)中的虚线箭头所示.



(a) 有向图 (b) 路径

Fig.1 Rule-based shortest path

图 1 基于规则的最短路径

定理 1.1. 基于规则的最短路径查询问题是一个 NP-hard 问题.

证明:本文通过将其归约到哈密尔顿路问题(NPC 问题)证明.给定一个有向图 G,令 v_s 和 v_e 分别表示起点和终点.G 中每条边的权重都设为 1,规则$\mathcal{R}$=(I,R)被设置为 $I=V-\{v_s,v_e\}$,$R=\varnothing$.显然,图 G 中存在一条从 v_s 到 v_e 的哈密尔顿路当且仅当从 v_s 到 v_e 的基于规则$\mathcal{R}$的最短路径的长度为$|V|-1$($|V|$表示 V 中顶点的个数),而且对于一条给定的路径,不能在多项式时间验证它是否是基于规则的最短路径,因此,基于规则的最短路径问题是一个 NP-hard 问题.证毕. □

2 广义规则树

若 p 为一条基于规则$\mathcal{R}$的路径,由第 1 节可知:在规则$\mathcal{R}$中,偏序关系集 R 中的偏序关系 $v_i\prec v_j(v_i,v_j\in I)$表示在路径 p 中存在一条从 v_i 到 v_j 的子路径.显然,对 I 中的任意一点 v_i,p 中存在一条从起点 v_s 到 v_i 的子路径,也存在一条 v_i 到终点 v_e 子路径.由于 v_s 和 v_e 分别为 p 的起点和终点,因此集合 $I_R=I\cup\{v_s,v_e\}$中的任意点都在基于规则$\mathcal{R}$的路径 p 中,称集合 I_R 为规则点集,I_R 中的点称为规则点.本节提出了广义规则树的概念,广义规则树并非真正的树,其将规则点集 I_R 中的全部顶点组织成类似树的结构,并通过树中的祖先后代关系来表示两个规则点之间的偏序关系,即:若 $v_i\prec v_j$,则在广义规则树中,结点 v_i 是结点 v_j 的祖先结点.与普通树区别在于,广义规则树中的结点可能具有多个父亲结点.利用广义规则树,本文提出的算法可以高效判断搜索到的路径 p 是否是一条满足规则的路径.下面首先给出广义规则树的定义,然后给出广义规则树的生成算法.

定义 2.1(广义规则树). 满足规则$\mathcal{R}$的广义规则树是一棵由规则点集 I_R 中全部顶点构成的广义树,记为 $T_{\mathcal{R}}$,其满足以下两个条件.

(1) 起点 v_s 为根结点、终点 v_e 为唯一叶结点;

(2) 对 $T_{\mathcal{R}}$ 中除根结点和叶结点外的任意两个结点 v_i,v_j,若 v_i 是 v_j 的一个父亲结点,当且仅当 $v_i\prec v_j$;且不存在另外一个顶点 $v_k\in I$,使得 $v_i\prec v_k$ 和 $v_k\prec v_j$ 同时成立.

广义规则树满足以下两个性质.

性质 2.1. 给定起点 v_s、终点 v_e 和规则$\mathcal{R}$,广义规则树 $T_{\mathcal{R}}$ 存在且唯一.

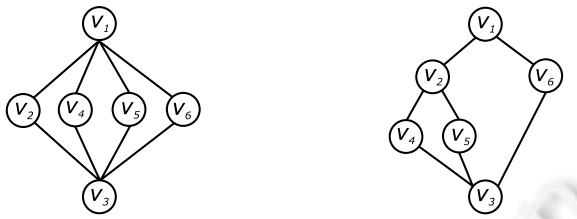
证明:当给定起点 v_s 和终点 v_e,$T_{\mathcal{R}}$ 的根和唯一叶结点即已确定.给定规则$\mathcal{R}$,$T_{\mathcal{R}}$ 中结点的父亲-孩子关系也已确定,故 $T_{\mathcal{R}}$ 存在且唯一.证毕. □

性质 2.2. R 中蕴含的任意一个偏序关系都对应广义规则树中的一对祖先后代结点.

证明:若 R 中蕴含 $v_i\prec v_j$,由广义规则树定义中的条件(2)可知:在 $T_{\mathcal{R}}$ 中,v_i 是 v_j 的祖先结点.证毕. □

显然,在广义规则树中,起点是所有其他点的祖先结点,终点是所有其他点的后代结点.规则点 $v_i\in I_R$ 的全部父结点集合表示为 S_i^p,子结点集合表示为 S_i^c,祖先结点集合表示为 S_i^a,后代结点集合表示为 S_i^d.例 2.1 给出了广义规则树的示例.

例 2.1:给定有向图 G,如图 1(a)所示,其中,起点为 v_1,终点为 v_3,规则$\mathcal{R}$=(I,R),$I=\{v_2,v_4,v_5,v_6\}$,若 $R=\varnothing$,则基于规则$\mathcal{R}$的广义规则树如图 2(a)所示;若 $R=\{v_2\prec v_4,v_2\prec v_5\}$,此基于规则$\mathcal{R}$的广义规则树如图 2(b)所示.



(a) (b)

Fig.2 Generalized rule tree

图 2 广义规则树

下文中,规则点 $v_i \in I_R$ 的父结点、子结点、祖先结点和后代结点均指广义规则树中对应树结点 v_i 的父结点、子结点、祖先结点和后代结点.例如在图 2(b)中,规则点 v_2 的父结点集合 $S_2^p=\{v_1\}$、子结点集合 $S_2^c=\{v_4,v_5\}$、祖先结点集合 $S_2^a=\{v_1\}$、后代结点集合 $S_2^d=\{v_3,v_4,v_5\}$.

算法 1 展示了构造广义规则树的伪代码.初始阶段,算法将所有的规则点初始化一棵广义树(第 1 行),广义树的根设为起点 v_s,唯一叶结点设为终点 v_e,I 中的所有顶点都互为兄弟结点,且它们的父结点和祖先结点都设为 v_s,子结点和后代结点都设为 v_e,因此,v_s 的子结点集合和 v_e 的父结点集合均为 I.算法 1 的主要思想是:依次处理 R 中的每一对偏序关系,并逐步构造广义规则树.在每次迭代中,对每个$\langle v_i,v_j\rangle \in R$,算法判断在 $T_{\mathcal{R}}$ 中 v_i 是否是 v_j 的祖先:若是,则忽略$\langle v_i,v_j\rangle$并跳出本轮迭代(第 3 行~第 3 行);否则,根据$\langle v_i,v_j\rangle$更新广义规则树 $T_{\mathcal{R}}$.更新过程如下.

- 首先,对 $T_{\mathcal{R}}$ 中 v_i 的每个子结点 $v_k \in S_i^c$,若 v_k 是 v_j 的后代结点,则将 v_k 从 S_i^c 中删除(第 6 行~第 10 行);对 $T_{\mathcal{R}}$ 中 v_j 的每个父结点 $v_k \in S_j^p$,若 v_k 是 v_i 的祖先结点,则将 v_k 从 S_j^p 中删除(第 11 行~第 15 行);
- 然后,将 v_i 加入 v_j 的父结点集合,将 v_j 加入 v_i 的子结点集合(第 16 行);
- 最后,对 v_j 的每个祖先结点 v_k,将 v_k 的后代结点集合更新为 $S_k^d \cup S_j^d$ (第 17 行~第 19 行);对 v_i 的每个祖先结点 v_k,将 v_k 的祖先结点集合更新为 $S_k^a \cup S_i^a$ (第 20 行~第 22 行).当 R 中的全部偏序关系处理完毕,算法结束并返回 $T_{\mathcal{R}}$.

算法 1. *GeneralizedRuleTree*(G,v_s,v_e,$\mathcal{R}$).

输入:图 G,起点 v_s,终点 v_e,规则$\mathcal{R}$=(I,R);

输出:广义规则树 $T_{\mathcal{R}}$.

1. 用所有的规则点初始化广义树,即:广义树的根结点为 v_s,叶结点为 v_e,I 中的所有顶点都互为兄弟结点,并且它们的父结点和祖先结点都设为 v_s,子结点和后代结点都设为 v_e;
2. **for each** $\langle v_i,v_j\rangle \in R$ **do**
3. **if** $v_i \in S_j^a$ **then**
4. **continue**;
5. **end if**
6. **for each** $v_k \in S_i^c$ **do**
7. **if** $v_k \in S_j^d$ **then**
8. $S_i^c \leftarrow S_i^c-\{v_k\}, S_k^p \leftarrow S_k^p-\{v_i\}$;
9. **end if**
10. **end for**
11. **for each** $v_k \in S_j^p$ **do**
12. **if** $v_k \in S_i^a$ **then**
13. $S_j^p \leftarrow S_j^p-\{v_k\}, S_k^c \leftarrow S_k^c-\{v_j\}$;

14. **end if**
15. **end for**
16. $S_j^p \leftarrow S_j^p \cup \{v_i\}, S_i^c \leftarrow S_i^c \cup \{v_j\}$;
17. **for each** $v_k \in S_j^a$ **do**
18. $S_k^d \leftarrow S_k^d \cup S_j^d$
19. **end for**
20. **for each** $v_k \in S_i^d$ **do**
21. $S_k^a \leftarrow S_k^a \cup S_i^a$
22. **end**
23. **end for**
24. **return** $T_{\mathcal{R}}$

例 2.2:接例 1.1,利用算法 1 构造其对应的广义规则树.首先,初始化广义规则树,结果如图 3(a)所示;当偏序关系 $v_2 \prec v_4$ 插入到广义树中后,结点 v_2 的子结点集合修改为 $\{v_4\}$,结点 v_4 的父结点集合修改为 $\{v_2\}$,结点 v_1 的子结点集合修改为 $\{v_2,v_5,v_6\}$,结点 v_3 的父结点集合修改为 $\{v_4,v_5,v_6\}$,结果如图 3(b)所示;当偏序关系 $v_2 \prec v_5$ 插入到广义树中后,结点 v_2 的子结点集合修改为 $\{v_4,v_5\}$,结点 v_5 的父结点集合修改为 $\{v_2\}$,结点 v_1 的子结点集合修改为 $\{v_2,v_6\}$,结果如图 3(c)所示.

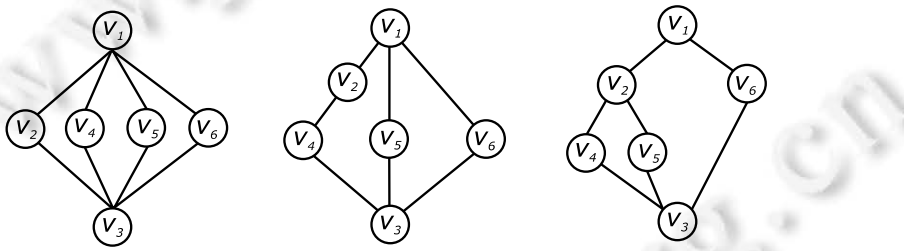


Fig.3 Construction of generalized rule tree

图 3 广义规则树的构造过程

广义规则树 $T_{\mathcal{R}}$ 中,结点 v_i 被某路径 p 合法访问是指:若 v_i 为根结点,则 p 包含 v_i;否则,p 包含 v_i 且 v_i 在 $T_{\mathcal{R}}$ 中的所有父结点都已被 p 中从起点 v_s 到 v_i 的子路径合法访问.定理 2.1 给出了路径 p 是否满足规则 $\mathcal{R}$ 的判断条件.

定理 2.1. 路径 p 为一条从起点 v_s 到终点 v_e 的路径,若终点 v_e 被路径 p 合法访问,则路径 p 为一条基于规则 $\mathcal{R}$ 的路径.

证明:由规则点被路径合法访问的定义可知,集合 I 中的点都在路径 p 中.下面证明偏序关系集 R 中的所有偏序关系均已满足.假设存在一个偏序关系 $\langle v_i,v_j\rangle \in R$,使得 p 不满足此偏序关系,即 p 中不存在一条从 v_i 到 v_j 的子路径,因此,规则点 v_j 没有被合法访问且 v_j 的所有子结点也没有被合法访问.由于终点 v_e 是所有规则点的后代结点,可得终点 v_e 没有被路径 p 合法访问.这与已知条件矛盾.因此,所有的偏序关系均已满足.综上可得,路径 p 为一条基于规则 $\mathcal{R}$ 的路径.证毕. □

由定理 2.1 可知:可以依据一个查询的终点是否被一条路径合法访问,来判断此路径是否为基于规则的路径.在下文介绍的前向扩展算法中,本文依据定理 2.1 来判断是否已找到一条基于规则的路径.

3 分层收缩技术

分层收缩(contraction hierarchies,简称 CH)[12]是一种简单有效的图数据预处理技术,它可以提高最短路径

的查询效率.CH本质上是预先存储一些最短路径信息,从而实现加速最短路径查找的目的.本文采用CH技术进行预处理,以使算法能更高效地计算两点之间的最短路径.给定图 $G(V,E,w)$,CH 首先给 V 中所有顶点分配一个序号,任意两个顶点的序号都不相同;然后依据序号的大小,按照从小到大的顺序对点进行收缩.顶点 v_i 被收缩是指:把 v_i 以及与 v_i 相连的边从 G 中删除,并且添加一些新的边到 G 中.具体地,对每一对 v_i 的入边(v_j,v_i)和出边(v_i,v_k),如果路径(v_j,v_i,v_k)是连接v_j和v_k的唯一的最短路径,则往G中添加边(v_j,v_k),并且将边(v_j,v_k)的权重设为(v_j,v_i)和(v_i,v_k)的权重之和.所有顶点收缩完成之后,CH将原图G分为两个新图,分别为前向图G_u和后向图G_d;然后,CH利用 G_u 和 G_d 计算最短路径.CH技术的具体细节见文献[12].

4 FE算法

本节提出前向扩展算法(forward expanding,简称 FE)来求解基于规则$\mathcal{R}$的最短路径.接下来,首先介绍最优子排列,然后详细介绍FE算法,最后分析了算法的时间和空间复杂度.

4.1 最优子排列

给定有向图$G(V,E,w)$、规则$\mathcal{R}=(I,R)$以及起点v_s和终点v_e,可得规则点集I_R,令r表示I_R中点的个数,即$r=|I_R|$.I_R的一个排列π是一个点序列 $v_1v_2\ldots v_r$,对任意点 $v_i(1\leqslant i\leqslant r)$都有 $v_i\in I_R$,且排列π中不包含重复的点.显然,一个规则点集 I_R 总共有 $r!$个排列.对于一个排列π,本文用规则点在π中的次序来表示此规则点在路径 p 中的合法访问顺序,即:若$v_i\in I_R$在π中处于第 i 个位置,则在 p 中,v_i 是第 i 个被合法访问的.给定排列$\pi=v_1v_2\ldots v_r$,如果路径 p 满足下面3个条件,称 p 为一条对应于π的路径,记为 $p|_\pi$.

(1) 对π中任意点 v_i,p 中都包含点 v_i;

(2) 若π中点 v_i 的位置在 v_j 之前,则 p 中包含一条从 v_i 到 v_j 的子路径;

(3) 路径 p 的起点和终点分别为 v_1 和 v_r.

给定路径 $p|_\pi$,它可以被π中的所有点划分为 $r-1$ 条子路径 $v_1\rightsquigarrow v_2,v_2\rightsquigarrow v_3,\ldots,v_{r-1}\rightsquigarrow v_r$.每条子路径 $v_i\rightsquigarrow v_{i+1}(1\leqslant i\leqslant r-1)$都被称为 $p|_\pi$的一个路径片段.本文使用 $S_{p|\pi}$来表示 $p|_\pi$的所有路径片段.

在例 1.1 中,路径 $p^*_{s,e,\mathcal{R}}=(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$ 是一个对应于排列$\pi=v_1v_2v_4v_6v_5v_3$ 的路径,此时,$S_{p|\pi}=\{(v_1,v_3,v_2),(v_2,v_4),(v_4,v_6),(v_6,v_5),(v_5,v_3)\}$.下面给出基于规则的排列定义.

定义 4.1(基于规则的排列). 给定图$G(V,E,w)$,起点为v_s,终点为v_e,规则$\mathcal{R}=(I,R)$,π是规则点集I_R的一个排列.已知条件:(1) π起始位置和末尾位置的顶点分别为v_s和v_e;(2) 对每一个偏序关系$\langle v_i,v_j\rangle\in R$(或者 $v_i\prec v_j$),π中 v_i 的位置都在 v_j 之前.若排列π满足上述两个条件,则称π为从 v_s 到 v_e 的基于规则$\mathcal{R}$的排列.所有基于规则$\mathcal{R}$的排列构成的集合记为 $\Pi_{v_s,v_e,\mathcal{R}}$,即,$\Pi_{v_s,v_e,\mathcal{R}}$ 中所有排列都以 v_s 为起始位置顶点、v_e 为末尾位置顶点且满足规则$\mathcal{R}$.

由定义 4.1 可知:若π为一个基于规则$\mathcal{R}$的排列,则 $p|_\pi$为一个基于规则$\mathcal{R}$的路径.如果路径 p 是对应于排列π的路径,且 p 中的所有路径片段均为最短路径,本文称 p 为对应于排列π的最短路径,记为 $p^*|_\pi$.由此可以得到下面的定理:

定理 4.1. 给定图$G(V,E,w)$,起点为v_s,终点为v_e,规则$\mathcal{R}=(I,R)$,若p^*是I_R的所有基于规则的排列所对应的$p^*|_\pi$中权重最小的一条路径,则p^*即为基于规则$\mathcal{R}$的最短路径.

证明:给定I_R的一个基于规则$\mathcal{R}$的排列π,因为$p^*|_\pi$中的所有路径片段都是最短路径,因此$p^*|_\pi$的权重不大于π所对应的每一条路径$p|_\pi$的权重;又因为p^*是所有基于规则的排列所对应的$p^*|_\pi$中权重最小的一条路径,而且所有的$p^*|_\pi$都是基于规则$\mathcal{R}$的路径,因此,p^*为基于规则$\mathcal{R}$的最短路径.证毕. □

由定理 4.1 可知:对于一个基于规则的排列π,只有当π所对应的路径$p|_\pi$中的所有路径片段都为最短路径时,即 $p|_\pi$为 $p^*|_\pi$,此路径才有可能成为基于规则的最短路径.因此,本文后续部分所提到的任意排列π所对应的路径都特指$p^*|_\pi$,即所有的路径片段都为最短路径.

对于点集$V_s\subseteq V$,可以逐步扩展点序列的长度来得到V_s的排列,其中,点序列中的每个点都属于V_s,且点序列中不包含重复的顶点.如果点序列中包含l个顶点,本文称此点序列为V_s的l-子排列(特别地,本文规定:规则点集

I_R的所有子排列均以起点 v_s作为起始位置的顶点),记为π_l,其中,$1\leqslant l\leqslant r(r=|V_s|)$.包含$\pi_l$中 l 个顶点的点集合称为子排列π_l 对应的点集,记为 V_{π_l} .给定一个排列π,$\pi\oplus v$ 表示π扩展一个顶点后形成的新排列,这称为排列的扩展操作,其中,$\oplus$是一个连接符,即,把点 v 添加到π的末尾.若 π_{l_1} 和 π_{l_2} 分别为 V_s 的 l_1-子排列和 l_2-子排列,已知条件:(1) $l_1\leqslant l_2$;(2) π_{l_2} 经由 π_{l_1} 扩展 0 个或多个顶点得到.如果 π_{l_1} 和 π_{l_2} 满足条件(1)和条件(2),则称 π_{l_1} 为 π_{l_2} 的前缀.例如,v_1v_2 是 $v_1v_2v_3v_4$ 的前缀.下面本文给出子排列满足的规则的定义.

定义 4.2(子排列满足的规则$\mathcal{R}'=(\boldsymbol{I}',\boldsymbol{R}')$). 给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}=(I,R)$,π_l 为 I_R 的 l-子排列$(1\leqslant l\leqslant|I_R|)$且$\pi_l$起始位置的顶点为 v_s.$\mathcal{R}'=(I',R')$,其中,$I'\subseteq I,R'\subseteq R$,已知条件:(1) π_l 是 I'的一个排列;(2) 对任意偏序关系$\langle v_i,v_j\rangle\in R'$,都有 $v_i,v_j\in I'$且π_l中 v_i 的位置在 v_j 之前;(3) 不存在偏序关系$\langle v_i,v_j\rangle\in R-R'$,使得 $v_i,v_j\in I'$.若规则$\mathcal{R}'$满足上述 3 个条件,则称$\mathcal{R}'$为子排列π_l满足的规则.

利用定义 4.2,可以很容易地给出最优子排列定理.

定理 4.2. 给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}=(I,R)$,若 I_R 的排列π对应的路径 $p|_\pi$为基于规则$\mathcal{R}$的最短路径,π_l 为π的包含 $l(1\leqslant l\leqslant|I_R|)$个顶点的前缀,$\pi_l$ 满足的规则为$\mathcal{R}'=(I',R')$,v^*是π_l 末尾位置的顶点,对任意排列 $\pi_l'\in\Pi_{v_s,v^*,\mathcal{R}'}$,都有 $w(p|_{\pi_l'})\geqslant w(p|_{\pi_l})$,本文称$\pi_l$为 I_R 的一个最优子排列.

证明:假设存在一个 $\pi_l'\in\Pi_{v_s,v^*,\mathcal{R}'}$ 且 π_l' 不等于π_l,使得 $w(p|_{\pi_l'})<w(p|_{\pi_l})$,将$\pi$的前缀$\pi_l$ 替换为 π_l' 得到 I_R 的一个新的排列π',显然,π'同样为基于规则$\mathcal{R}$的排列且 $w(p|_{\pi'})<w(p|_\pi)$.这与 $p|_\pi$为基于规则$\mathcal{R}$的最短路径矛盾,假设不成立.因此,对任意的排列 $\pi_l'\in\Pi_{v_s,v^*,\mathcal{R}'}$,都有 $w(p|_{\pi_l'})\geqslant w(p|_{\pi_l})$.证毕. □

给定有向图 $G(V,E,w)$如图 1(a)所示,已知规则$\mathcal{R}=(I,R)$,$I=\{v_2,v_4,v_5,v_6\}$,$R=\varnothing$,起点为 v_1,终点为 v_3,可得排列$\pi=v_1v_2v_4v_6v_5v_3$对应的路径 $p|_\pi=(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$为基于规则$\mathcal{R}$的最短路径,$\pi_4=v_1v_2v_4v_6$为$\pi$的包含 4 个顶点的前缀,$v_6$ 是π_4 末尾位置的顶点,π_4 满足的规则为$\mathcal{R}'=(I',R')$,其中,$I'=\{v_1,v_2,v_4,v_6\}$,$R'=\varnothing$, $\Pi_{v_1,v_4,\mathcal{R}'}=\{\pi_4=v_1v_2v_4v_6,\pi_4'=v_1v_4v_2v_6\}$,$\pi_4$和 π_4' 分别对应的路径为 $p|_{\pi_4}=(v_1,v_3,v_2,v_4,v_6)$ 和 $p|_{\pi_4'}=(v_1,v_3,v_2,v_4,v_5,v_3,v_2,v_4,v_6)$.显然,

$$w(p|_{\pi_4})<w(p|_{\pi_4'}).$$

推论 4.1. 给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}=(I,R)$,已知 I_R的长度为 $l(1\leqslant l\leqslant|I_R|)$的一个子排列$\pi_l$,$\pi_l$起始位置的顶点为 v_s、末尾位置的顶点为 v^*,π_l满足的规则为$\mathcal{R}'=(I',R')$,则扩展操作得到的子排列集合 $\Pi_{v_s,v^*,\mathcal{R}'}$中,存在一个子排列 π_l',使得对任意的 $\pi_l''\in\Pi_{v_s,v^*,\mathcal{R}'}$,都有 $w(p|_{\pi_l'})\leqslant w(p|_{\pi_l''})$,$I_R$ 的排列π对应的路径 $p|_\pi$为基于规则$\mathcal{R}$的最短路径,若π的长度为 l 的前缀属于 $\Pi_{v_s,v^*,\mathcal{R}'}$,将$\pi$的长度为 l 的前缀替换为 π_l' 得到新的排列π',则 $w(p|_{\pi'})=w(p|_\pi)$,本文称 π_l' 为 I_R 的一个最优子排列.

推论 4.1 的结论是显然的.因此,对于 $\Pi_{v_s,v^*,\mathcal{R}'}$ 中的排列,只需保留对应路径具有最小权重的任意一个排列,即只需保留最优子排列.基于推论 4.1,可以设计一个前向扩展算法来求解基于规则$\mathcal{R}$的最短路径,在第 4.2 节,本文将详细介绍此算法.

4.2 前向扩展算法

本节提出求解基于规则$\mathcal{R}$的最短路径的前向扩展算法,算法的主要思想是:从规则点集 I_R 的 1-子排列逐步进行扩展操作直至 I_R 的 r-子排列$(r=|I_R|)$,并利用最优子排列对生成的子排列进行过滤,最终返回 I_R 的 r-子排列对应的路径中权重最小的那条路径.

前向扩展算法的伪代码见算法 2.

算法 2. $FE(G,v_s,v_e,\mathcal{R})$.

输入:图 G,起点 v_s,终点 v_e,规则$\mathcal{R}=(I,R)$;

输出: $p_{s,e,\mathcal{R}}^*$.

1. $T_{\mathcal{R}}\leftarrow GeneralizedRuleTree(G,v_s,v_e,\mathcal{R})$;
2. 利用 NN 算法得到路径 p_g;

3. $\theta \leftarrow w(p_g)$;
4. $\pi_1 \leftarrow v_s$;
5. $R_1 \leftarrow \{p|_{\pi_1}\}$;
6. **for** l=2 **to** r=$|I_R|$ **do**
7. $R_l \leftarrow \varnothing$;
8. 依据点集 $V_{\pi_{l-1}}$ 将 R_{l-1} 划分为不同的集合;
9. **for each** $v \in I_R - \{v_s\}$ **do**
10. **for each** 划分 R_{l-1} 得到的集合 R'_{l-1} **do**
11. $R' \leftarrow \varnothing$;
12. **for each** $p|_{\pi_{l-1}} \in R'_{l-1}$ **do**
13. **if** π_{l-1} 不包含 v 且 v 的所有父结点均已被 $p|_{\pi_{l-1}}$ 合法访问 **then**
14. $\pi_l \leftarrow \pi_{l-1} \oplus v$;
15. $R' \leftarrow R' \cup \{p|_{\pi_l}\}$;
16. **end if**
17. **end for**
18. 查找 R'中具有最小权重的路径,记为 $p^*|_{\pi_l}$;
19. **if** $w(p^*|_{\pi_l}) \leqslant \theta$ **then**
20. $R_l \leftarrow R_l \cup \{p^*|_{\pi_l}\}$;
21. **end if**
22. **end for**
23. **end for**
24. **end for**
25. 查找 R_r 中具有最小权重的路径,记为 $p^*_{s,e,\mathcal{R}}$;
26. **return** $p^*_{s,e,\mathcal{R}}$;

给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}$=(I,R),FE 首先调用算法 1 得到广义规则树 $T_{\mathcal{R}}$ (第 1 行);然后,利用下文相关工作部分介绍的启发式算法 NN 求解得到基于规则$\mathcal{R}$的路径 p_g,令θ等于 p_g 的权重(第 2 行、第 2 行);第 4 行~第 24 行,FE 逐步扩展规则点集 I_R 的 1-子排列直至 I_R 的 r-子排列(r=$|I_R|$).在构建 I_R 的 1-子排列时,因为只有π_1=v_s 这一个 1 子排列满足规则,因此,FE 将 R_1 初始化为{v_s}(第 4 行、第 5 行);第 6 行~第 24 行,FE 循环得到 I_R 的 2-子排列至 r-子排列.下面将详细介绍 FE 求解 I_R 的 l-子排列的过程,其中,2≤l≤r.

FE 用 R_l 来存储 I_R 的 l 子排列所对应的路径:首先,将 R_l 初始化为空集(第 7 行);然后,FE 对长度为 l–1 的子排列进行扩展操作,并利用最优子排列理论过滤不可能成为基于规则的最短路径所对应排列的前缀.依据子排列 π_{l-1} 对应的点集 $V_{\pi_{l-1}}$,FE 将 R_{l-1} 划分为不同的集合,即属于同一集合里的路径拥有相同的点集 $V_{\pi_{l-1}}$,属于不同集合的路径拥有不同的点集 $V_{\pi_{l-1}}$ (第 8 行).接下来,FE 依次遍历规则点集 I_R 中除 v_s 之外的其他所有点(第 9 行),并对每一个 $v \in I_R - \{v_s\}$做如下操作:遍历划分 R_{l-1} 得到的每一个集合 R'_{l-1},R'_{l-1} 中的每一条路径都与一个长度为 l–1 的子排列所对应,对于 R'_{l-1} 中的每一条路径 $p|_{\pi_{l-1}}$ 及相应的π_{l-1},如果π_{l-1}中不包含点 v 且 v 的所有父结点均已被 $p|_{\pi_{l-1}}$ 合法访问,那么将 v 添加到π_{l-1}的末尾构成长度为 l 的子排列π_l并且生成相应的路径 $p|_{\pi_l}$,若 R'_{l-1} 中的每一条路径均已完成上述操作,则从新生成的路径中选择一条权重最小的路径,记为 $p^*|_{\pi_l}$,$p^*|_{\pi_l}$ 即为一个最优子排列,如果 $p^*|_{\pi_l}$ 的权重不大于θ,则将 $p^*|_{\pi_l}$ 添加到 R_l 中(第 10 行~第 22 行).

FE 求得 I_R 的 r-子排列所对应的路径之后,由于 I_R 的 r 子排列所对应的路径的最后一个顶点为终点 v_e,由定理 2.1 可知,I_R 的 r 子排列所对应的路径都为基于规则$\mathcal{R}$的路径,显然,基于规则$\mathcal{R}$的最短路径即为 I_R 的 r 子排列

所对应的路径中权重最小的一条.第 25 行、第 26 行,FE 首先遍历 R_r 得到具有最小权重的路径,然后返回此路径作为基于规则$\mathcal{R}$的最短路径.

例 4.1:接例 1.1,利用 FE 算法求解基于规则$\mathcal{R}$的最短路径的过程见表 2,其中,π_l 表示一个子排列,$|\pi_l|$表示子排列中包含顶点的个数,V_{π_l} 表示π_l对应的点集,v^*表示π_l的最后一个顶点,$p|_{\pi_l}$ 表示π_l所对应的路径.用贪心算法求解得到基于规则$\mathcal{R}$的路径 $p_g=(v_1,v_3,v_2,v_4,v_5,v_3,v_2,v_4,v_6,v_5,v_3)$,$\theta=w(p_g)=12$.加删除线的子排列表示被最优子排列过滤掉的子排列,从表 2 的结果可得,最优子排列可以减少算法在求解过程中生成的子排列个数,从而降低算法的求解时间.FE 算法最终得到的排列为 $\pi=v_1v_2v_4v_6v_5v_3$,所对应的路径为 $p|_\pi=(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$,$w(p|_\pi)=8$,因此,$p|_\pi$即为基于规则$\mathcal{R}$的最短路径.

Table 2 Solving process of FE algorithm

表 2 FE 算法的求解过程

$\|\pi_l\|$	π_l	V_{π_l}	v^*	最优前缀	$p\|_{\pi_l}$
1	v_1	$\{v_1\}$	v_1	v_1	v_1
2	v_1v_2	$\{v_1,v_2\}$	v_2	v_1v_2	(v_1,v_3,v_2)
2	v_1v_6	$\{v_1,v_6\}$	v_6	v_1v_6	(v_1,v_3,v_2,v_4,v_6)
3	$v_1v_2v_4$	$\{v_1,v_2,v_4\}$	v_4	$v_1v_2v_4$	(v_1,v_3,v_2,v_4)
3	$v_1v_2v_5$	$\{v_1,v_2,v_5\}$	v_5	$v_1v_2v_5$	(v_1,v_3,v_2,v_4,v_5)
3	$v_1v_2v_6$	$\{v_1,v_2,v_6\}$	v_6	$v_1v_2v_6$	(v_1,v_3,v_2,v_4,v_6)
3	$v_1v_6v_2$	$\{v_1,v_2,v_6\}$	v_2	$v_1v_6v_2$	$(v_1,v_3,v_2,v_4,v_6,v_5,v_3,v_2)$
4	$v_1v_2v_4v_5$	$\{v_1,v_2,v_4,v_5\}$	v_5	$v_1v_2v_4v_5$	(v_1,v_3,v_2,v_4,v_5)
4	$v_1v_2v_6v_5$ ~~$v_1v_6v_2v_5$~~	$\{v_1,v_2,v_5,v_6\}$	v_5	$v_1v_2v_6v_5$	$(v_1,v_3,v_2,v_4,v_6,v_5)$
4	$v_1v_2v_5v_6$	$\{v_1,v_2,v_5,v_6\}$	v_6	$v_1v_2v_5v_6$	$(v_1,v_3,v_2,v_4,v_5,v_3,v_2,v_4,v_6)$
4	$v_1v_2v_4v_6$	$\{v_1,v_2,v_4,v_6\}$	v_6	$v_1v_2v_4v_6$	(v_1,v_3,v_2,v_4,v_6)
4	$v_1v_2v_5v_4$	$\{v_1,v_2,v_4,v_5\}$	v_4	$v_1v_2v_5v_4$	$(v_1,v_3,v_2,v_4,v_6,v_5,v_3,v_2,v_4)$
4	~~$v_1v_6v_2v_4$~~ $v_1v_2v_6v_4$	$\{v_1,v_2,v_4,v_6\}$	v_4	$v_1v_2v_6v_4$	$(v_1,v_3,v_2,v_4,v_6,v_5,v_3,v_2,v_4)$
5	$v_1v_2v_4v_5v_6$ ~~$v_1v_2v_5v_4v_6$~~	$\{v_1,v_2,v_4,v_5,v_6\}$	v_6	$v_1v_2v_4v_5v_6$	$(v_1,v_3,v_2,v_4,v_5,v_3,v_2,v_4,v_6)$
5	~~$v_1v_2v_6v_4v_5$~~ $v_1v_2v_4v_6v_5$	$\{v_1,v_2,v_4,v_5,v_6\}$	v_5	$v_1v_2v_4v_6v_5$	$(v_1,v_3,v_2,v_4,v_6,v_5)$
5	$v_1v_2v_6v_5v_4$ ~~$v_1v_2v_5v_6v_4$~~	$\{v_1,v_2,v_4,v_5,v_6\}$	v_4	$v_1v_2v_6v_5v_4$	$(v_1,v_3,v_2,v_4,v_6,v_5,v_3,v_2,v_4)$
6	~~$v_1v_2v_4v_5v_6v_3$~~ $v_1v_2v_4v_6v_5v_3$ ~~$v_1v_2v_6v_5v_4v_3$~~	$\{v_1,v_2,v_3,v_4,v_5,v_6\}$	v_3	$v_1v_2v_4v_6v_5v_3$	$(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$

4.3 复杂度分析

4.3.1 时间复杂度.

FE 的时间复杂度主要集中在计算 I_R 的长度为 1 至 $r(r=|I_R|)$的子排列上,对于长度为 $l(1\leqslant l\leqslant r)$的子排列,由最优子排列可知,FE 至多生成 rC_r^l 个,其中,C_r^l 表示 I_R 中顶点的组合数.因此,FE 至多生成 $\sum_{l=1}^{r}rC_r^l$ 个子排列.本文采用 CH 技术求解两点之间的最短路径,它的时间复杂度为 $O(n\log n+m)(n=|V|,m=|E|)$,因此,FE 的时间复杂度为 $O((n\log n+m)\cdot(r\cdot 2^r))$.

4.3.2 空间复杂度.

FE 需要利用长度为 l–1 的子排列来求解长度为 l 的子排列,因为 FE 至多生成 rC_r^l 个长度为 $l(2\leqslant l\leqslant r)$的子排列,因此在求解长度为 l 的子排列时,内存的总消耗为 $O(n\cdot rC_r^{l-1}+n\cdot rC_r^l)$;一旦求得长度为 l 的子排列之后,即可将存储长度为 l–1 的子排列的内存释放.因此,FE 所消耗的内存最大为 $O\left(\max_{l=2}^{r}(n\cdot rC_r^{l-1}+n\cdot rC_r^l)\right)=O(n\cdot r\cdot 2^r)$,即,FE 的空间复杂度为 $O(n\cdot r\cdot 2^r)$.

5 FEBF 算法

本节提出 FE 算法的改进算法——基于最短优先策略的前向扩展算法(forward expanding based on best-first searching,简称 FEBF).接下来,本文首先分析如何对 FE 算法进行改进;然后给出基于最短优先策略的前向扩展算法,并对算法做出详细的介绍;接着,对 FEBF 算法提出了几个优化技术;最后,分析了 FEBF 算法的时间和空间复杂度.

5.1 基于最短优先策略的前向扩展算法

从第 4.2 节可知,FE 算法依次扩展规则点集 I_R 的长度为 1 至长度为 $r(r=|I_R|)$的子排列,最后从长度为 r 的子排列对应的路径中选择权重最小的路径作为问题的解.在求解长度为 l 的子排列时,FE 必须先求得长度为 l–1 的所有子排列.事实上,某些长度为 l–1 的子排列所对应路径的权重已大于基于规则$\mathcal{R}$的最短路径的权重,因此没有必要对这部分子排列进行扩展操作.在例 4.1 中长度为 3 的子排列 $v_1v_6v_2$ 对应路径为$(v_1,v_3,v_2,v_4,v_6,v_5,v_3,v_2)$,其权重为 9,而基于规则$\mathcal{R}$的最短路径权重为 8,因此,没有必要再对子排列 $v_1v_6v_2$ 进行扩展操作.基于此,本文提出基于最短优先策略来逐步扩展路径的算法,算法的主要思想是:在对子排列进行扩展操作时, FEBF 首先从已有的子排列中选择一个对应路径的权重最小的一个子排列,然后对它进行扩展操作,并得到对应的路径,所有规则点扩展结束之后,FEBF 删除这个子排列,然后重复上述操作,直至找到一个基于规则$\mathcal{R}$的排列,它所对应的路径即为基于规则$\mathcal{R}$的最短路径.

FEBF 算法本质上也是逐步扩展 I_R 的长度为 1 的子排列到长度为 r 的子排列,它采用最短优先的策略尽可能少地计算 I_R 的子排列,从而可以尽快找到问题的解.FEBF 算法的伪代码见算法 3.

算法 3. *FEBF*$(G,v_s,v_e,\mathcal{R})$.

输入:图 G,起点 v_s,终点 v_e,规则$\mathcal{R}=(I,R)$;

输出: $p^*_{s,e,\mathcal{R}}$.

1. 定义一个优先级队列 Q,Q 中的元素为元组$\langle\pi,w(p|_\pi)\rangle$且以 $w(p|_\pi)$的大小进行从小到大排序;
2. $T_{\mathcal{R}}\leftarrow$*GeneralizedRuleTree*$(G,v_s,v_e,\mathcal{R})$;
3. 利用 NN 算法得到路径 p_g;
4. $\theta\leftarrow w(p_g)$;
5. 将Ω中的每一个元素的值初始化为∞;
6. $\pi_1\leftarrow v_s$;
7. 将$\langle\pi_1,w(p|_{\pi_1})\rangle$ 插入到 Q 中;
8. 弹出 Q 的队首元素$\langle\pi',w(p|_{\pi'})\rangle$,令 v^*为π'末尾位置的点;
9. **while** $v^*\neq v_e$ **do**
10. **for each** $v\in I_R-\{v_s\}$ **do**
11. **if** π'不包含 v 且 v 的所有父结点均已被 $p|_{\pi'}$合法访问 **then**
12. $\pi\leftarrow\pi'\oplus v$;
13. **if** $\Omega_{v,V_\pi}>w(p|_\pi)$ **then**
14. $\Omega_{v,V_\pi}\leftarrow w(p|_\pi)$;
15. 将$\langle\pi,w(p|_\pi)\rangle$插入到 Q 中;
16. **end if**
17. **end if**
18. **end for**
19. 弹出 Q 的队首元素$\langle\pi',w(p|_{\pi'})\rangle$,令 v^*为π'末尾位置的点;
20. **end while**

21. $p^*_{s,e,\mathcal{R}} \leftarrow p|_{\pi'}$;

22. **return** $p^*_{s,e,\mathcal{R}}$;

给定图 $G(V,E,w)$,起点为 v_s,终点为 v_e,规则$\mathcal{R}=(I,R)$,FEBF 利用优先级队列实现从已有子排列中选出具有最小权值的路径及其相应的子排列.算法首先定义优先级队列 Q,Q 中存储的元素为二元组,记为$\langle \pi, w(p|_\pi)\rangle$,其中,$\pi$为 I_R 的子排列,并用 $w(p|_\pi)$的值进行优先级比较,以实现对 Q 中的元素从小到大进行排序(第 1 行).接下来调用算法 1 得到广义规则树 $T_{\mathcal{R}}$ (第 2 行),然后用 NN 算法求解得到基于规则$\mathcal{R}$的路径 p_g,令θ等于 p_g的权重(第 3 行、第 4 行).为了利用最优子排列过滤子排列,FEBF 逐步构建最优子排列表(记为Ω)来存储已得到的最优子排列.当扩展得到子排列π_l 时,FEBF 首先查找最优子排列表中对应位置的元素(记为 $\Omega_{v^*,V_{\pi_l}}$,其中,v^*为π_l 末尾位置的顶点,V_{π_l} 为子排列π_l 对应的点集),然后对 $\Omega_{v^*,V_{\pi_l}}$ 和 $w(p|_{\pi_l})$ 进行比较:若 $\Omega_{v^*,V_{\pi_l}} > w(p|_{\pi_l})$,则将 $\Omega_{v^*,V_{\pi_l}}$ 的值设为 $w(p|_{\pi_l})$ 且保留子排列π_l 并对其进行后续操作;否则,删除子排列π_l.FEBF 将最优子排列表中每一个元素的值都初始化为∞(第 5 行).接下来,FEBF 将逐步扩展子排列直至找到基于规则$\mathcal{R}$的排列(第 6 行~第 20 行).

在构建 I_R 的 1-子排列时,因为只有$\pi_1=v_s$这一个 1 子排列满足规则,因此,FEBF 将$\langle \pi_1, w(p|_{\pi_1})\rangle$ 插入到优先级队列 Q 中(第 6 行、第 7 行),然后弹出 Q 的队首元素$\langle \pi', w(p|_{\pi'})\rangle$,令 v^*为π'末尾位置的点(第 8 行).如果 $v^*=v_e$(第 9 行),则表示π'是一个基于规则$\mathcal{R}$的排列(由定理 2.1 可得),又因为 Q 每次弹出的位于队首的元素都具有最小的路径权重,因此,$p|_{\pi'}$即为基于规则$\mathcal{R}$的最短路径,FEBF 返回此路径作为问题的解(第 21 行、第 22 行);否则($v^*\neq v_e$),执行以下循环操作(第 10 行~第 19 行).首先,FEBF 依次遍历规则点集 I_R 中除 v_s 之外的其他所有点(第 10 行),然后对每一个 $v\in I_R-\{v_s\}$做如下操作:如果π'中不包含点 v 且 v 的所有父结点均已被 $p|_{\pi'}$合法访问,那么将 v 添加到π'的末尾构成新的子排列π,若 $\Omega_{v,V_\pi} > w(p|_\pi)$,则将 Ω_{v,V_π} 的值设为 $w(p|_\pi)$,并将$\langle \pi, w(p|_\pi)\rangle$插入到 Q 中(第 11 行~第 17 行).若 $I_R-\{v_s\}$中的每个点均已完成上述操作,则 FEBF 弹出 Q 的队首元素$\langle \pi', w(p|_{\pi'})\rangle$,令 v^*为π'末尾位置的点(第 19 行),然后开始下一轮循环.

例 5.1:接例 1.1,利用 FEBF 算法求解基于规则$\mathcal{R}$的最短路径.

FEBF 首先生成$\langle v_1,0\rangle$并把它插入到优先级队列 Q 中,接下来的操作是:

从 Q 中弹出$\langle v_1,0\rangle$,利用它扩展得到$\langle v_1v_2,2\rangle$,$\langle v_1v_6,5\rangle$并分别插入到 Q 中;

从 Q 中弹出$\langle v_1v_2,2\rangle$,利用它扩展得到$\langle v_1v_2v_4,3\rangle$,$\langle v_1v_2v_5,4\rangle$,$\langle v_1v_2v_6,5\rangle$并分别插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_4,3\rangle$,利用它扩展得到$\langle v_1v_2v_4v_5,4\rangle$,$\langle v_1v_2v_4v_6,5\rangle$并分别插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_5,4\rangle$,利用它扩展得到$\langle v_1v_2v_5v_4,7\rangle$,$\langle v_1v_2v_5v_6,9\rangle$并分别插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_4v_5,4\rangle$,利用它扩展得到$\langle v_1v_2v_4v_5v_6,9\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_6,5\rangle$,利用它扩展得到$\langle v_1v_6v_2,9\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_6,5\rangle$,利用它扩展得到$\langle v_1v_2v_6v_4,10\rangle$,$\langle v_1v_2v_6v_5,7\rangle$并分别插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_4v_6,5\rangle$,利用它扩展得到$\langle v_1v_2v_4v_6v_5,7\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_5v_4,7\rangle$,利用它扩展得到$\langle v_1v_2v_5v_4v_6,9\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_6v_5,7\rangle$,利用它扩展得到$\langle v_1v_2v_6v_5v_4,10\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_4v_6v_5,7\rangle$,利用它扩展得到$\langle v_1v_2v_4v_6v_5v_3,8\rangle$并插入到 Q 中;

从 Q 中弹出$\langle v_1v_2v_4v_6v_5v_3,8\rangle$且满足路径搜索终止条件,此时算法终止,FEBF 返回$(v_1,v_3,v_2,v_4,v_6,v_5,v_3)$作为问题的解.

在例 4.1 中,FE 算法总共生成 24 个子排列;而在例 5.1 中,FEBF 总共生成 18 个子排列.显然,FEBF 能有效减少算法生成的子排列个数.

5.2 优化技术

本节提出 3 个优化技术来提高 FEBF 算法的效率.

- 缓存机制

给定 I_R 的基于规则$\mathcal{R}$的两个不同排列π和π',这两个排列对应的路径 $p^*|_\pi$和 $p|_{\pi'}$可能包含一些相同的路径片段,因此这些相同的路径片段本质上只需要计算一次.本文可以采用缓存机制,将计算过的路径片段存储在内存中,这样就可以避免大量重复的计算,从而提高算法的效率.比如在例 4.1 中,长度为 6 的排列 $v_1v_2v_4v_6v_5v_3$ 和 $v_1v_2v_4v_6v_5v_3$所对应的路径包含相同的路径片段 $v_1\rightsquigarrow v_2$ 和 $v_2\rightsquigarrow v_4$,因此当采用缓存机制后,这两个路径片段仅仅只需要计算一次,当再次需要计算这两个路径片段时,可以直接从缓存中读取出来.实验结果验证了缓存机制可以有效避免大量的冗余计算.

- 排列过滤

当 FE 对长度为 l–1 的子排列π_{l-1}进行扩展操作时,它首先判断规则点 $v\in I_R$是否已在π_{l-1}中,然后再判断 v 的所有父结点是否已被路径 $p|_{\pi_{l-1}}$ 合法访问,若上述两个条件都满足,则将 v 添加到π_{l-1}的末尾构成长度为 l 的子排列π_l.上述两个条件本质上是从规则点 v 是否应该被合法访问这个角度判断 v 是否应该添加到π_{l-1}的末尾,本文也可以从路径权重的角度来进一步过滤生成的排列,即:已知 $v_i,v_j\in I_R$且 v_j 已被添加到π_{l-1}的末尾构成$\pi_{l-1}\oplus v_j$, v^*是π_{l-1}末尾位置的顶点,如果 v^*到 v_j 的最短路径是 v^*到 v_i 最短路径的子路径,则 v_i 不应被添加到π_{l-1}的末尾构成$\pi_{l-1}\oplus v_i$.下面的定理保证了排列过滤的正确性.

定理 5.1. 给定长度为 l–1 的子排列π_{l-1},v^*是π_{l-1}末尾位置的顶点,已知 $v_i,v_j\in I_R$,$\pi_{l-1}\oplus v_j$和$\pi_{l-1}\oplus v_i$分别为 FE 用 v_j和 v_i对π_{l-1}进行扩展操作得到的长度为 l 的子排列,如果 v^*到 v_j 的最短路径是 v^*到 v_i 最短路径的子路径,则 I_R 的所有基于规则$\mathcal{R}$且以$\pi_{l-1}\oplus v_i$ 为前缀的排列π都对应一个 I_R 基于规则$\mathcal{R}$且以$\pi_{l-1}\oplus v_j$ 为前缀的排列π',且 $w(p|_{\pi'})\leqslant w(p|_\pi)$.

证明:不失一般性,设$\pi_{l-1}=v_1v_2\ldots v^*$是 I_R的长度为 l–1 的子排列,$\pi_{l-1}\oplus v_j$和$\pi_{l-1}\oplus v_i$分别为 FE 用 v_j和 v_i对π_{l-1}进行扩展操作得到的长度为 l 的子排列,已知 v^*到 v_j 的最短路径是 v^*到 v_i 最短路径的子路径.

设$\pi=v_1v_2\ldots v^*v_i\ldots v_kv_jv_x\ldots v_r$ 为 I_R 的基于规则$\mathcal{R}$的一个排列,显然,$\pi'=v_1v_2\ldots v^*v_jv_i\ldots v_kv_x\ldots v_r$ 也为 I_R的基于规则$\mathcal{R}$的一个排列,$\pi_{l-1}\oplus v_i$ 和$\pi_{l-1}\oplus v_j$ 分别为π和π'的前缀,因为任意排列对应的路径包含的所有路径片段均为最短路径,因此 $p|_\pi$中的路径片段 $v^*\rightsquigarrow v_i$ 可以分解为 $p|_{\pi'}$中的两个路径片段 $v^*\rightsquigarrow v_j$ 和 $v_j\rightsquigarrow v_i$,而且分解前后路径的权重不变.此外, $p|_\pi$中的路径片段 $v_k\rightsquigarrow v_j$ 和 $v_j\rightsquigarrow v_x$ 的权重之和大于等于 $p|_{\pi'}$中的路径片段 $v_k\rightsquigarrow v_x$ 的权重.

综上可得 $w(p|_{\pi'})\leqslant w(p|_\pi)$.证毕. □

在例 4.1 中,对于长度为 2 的子排列 v_1v_2,可以分别用点 v_4,v_5,v_6 对 v_1v_2 进行扩展操作,从而得到 $v_1v_2v_4$,$v_1v_2v_5$, $v_1v_2v_6$.因为从 v_2 到 v_4 的最短路径(v_2,v_4)是从 v_2 到 v_5 最短路径(v_2,v_4,v_5)的子路径,也是从 v_2 到 v_6 最短路径(v_2,v_4,v_6)的子路径,因此可以过滤掉子排列 $v_1v_2v_5$ 和 $v_1v_2v_6$.

- 权重预估

当 FE 扩展长度为 l–1 的子排列得到长度为 l 的子排列π_l 时,v^*是π_l 末尾位置的顶点.对于 I_R 的所有基于规则$\mathcal{R}$且以π_l为前缀的排列π,可以估计 $w(p|_\pi)$的下限,从而使得 FE 可以尽量早的过滤掉一些排列,提高算法的效率.具体的做法是:当得到π_l后,首先计算 $w(p|_{\pi_l})$,然后计算从 v^*到终点 v_e 的最短路径的权重,然后得到这两个路径的权重之和 w^*,若 $w^*>\theta$,则舍弃子排列π_l;否则保留π_l.

5.3 复杂度分析

本文首先分析 FEBF 的时间复杂度,对于长度为 $l(1\leqslant l\leqslant|I_R|)$的子排列,FEBF 至多生成 rC_r^l 个,因为本文采用 CH 技术求解两点之间的最短路径,因此,FEBF 的时间复杂度为 $O((n\log n+m)\cdot(r\cdot 2^r))(n=|V|,m=|E|)$;接下来分析算法的空间复杂度,FEBF 的优先级队列 Q 中至多存储 $O(r\cdot 2^r)$个元素,每个元组至多消耗的内存为 $O(n)$,除此之外,最优子排列表Ω消耗的内存至多为 $O(r\cdot 2^r)$,因此,FEBF 的空间复杂度为 $O(n\cdot r\cdot 2^r+r\cdot 2^r)=O(n\cdot r\cdot 2^r)$.

6 实验分析

本节使用不同规模的真实数据集来验证算法的有效性,并与解决同类问题中已有的最好算法作对比.第 6.1

节介绍了实验的基本设置,第 6.2 节对实验的结果进行分析.

6.1 实验设置

本文的所有算法都使用 C++语言实现,并在 Linux 操作系统环境下进行测试,硬件的信息为:Intel(R) Core(TM) i7-4770K,32GB RAM.本文对每组实验重复 100 次,并把平均值作为此组实验的结果.如果一个算法在某个数据集上的运行时间超过 24h 或者超过 32GB RAM,则忽略此算法在这个数据集上的运行结果.

- 数据集

本文使用 8 个真实数据集来验证本文提出的算法,其中 4 个为道路交通网数据集(http://www.dis.uniroma1.it/challenge9/index.shtml),另外 4 个为非道路交通网数据集(http://snap.stanford.edu/data/),数据集的详细信息见表 3,d 表示顶点的平均度的大小.对于道路交通网数据集,本文将两点之间的距离作为边的权重;对于非道路交通网数据集,本文给每一条边随机生成一个 1~100 之间的值作为此边的权重值.

Table 3 Datasets

表 3 数据集

Name	Nodes	Edges	d	Description
NY	264 346	733 846	2.78	New York City Road network
BAY	321 270	800 172	2.49	San Francisco Bay Area Road network
COL	435 666	1 057 066	2.43	Colorado Road network
FLA	1 070 376	2 712 798	2.53	Florida Road network
ca-CondMat	23 133	93 497	4.04	Collaboration network of Arxiv condensed matter
email-EuAll	265 214	420 045	1.25	Email network from a EU research institution
com-Amazon	334 863	925 872	2.76	Amazon product network
p2p-Gnutella30	36 682	88 328	2.41	Gnutella peer to peer network from August 30 2002

- 查询集

对于每个数据集,本文生成 25 个查询集 Q1~Q25,它们分别对应不同规模的规则$\mathcal{R}$.在现实生活中,I 的规模一般不超过 10,本文从实际应用的角度设计了以下的查询集.对于 Q1~Q5,$|R|=5$,I 中顶点的个数依次为 6~10;对于 Q6~Q10,$|I|=8$,R 中偏序关系个数依次为 4,6,8,10,12;对于 Q11~Q15,$R=\varnothing$,I 中顶点的个数依次为 6~10;对于 Q16~Q20,I 中顶点的个数依次为 6~10,对应的 R 中偏序关系个数依次为 15,21,28,36,45;对于 Q21~Q25,$|R|=10$,I 中顶点的个数依次为 16~20.注意到:查询集 Q11~Q15 中偏序集个数为 0,此问题退化为广义旅行商问题;Q16~Q20 中的偏序集可以构成一个全序集,即 I 中顶点的合法访问顺序已确定,此问题退化为最优序列路径查询问题.Q21~Q25 中规则$\mathcal{R}$的规模较大,是用来验证算法能否应用于输入规模较大的情形.对于上述所有的查询集,I 中所有的顶点以及起点 v_s 和终点 v_e 都是采用随机数的方式随机生成,R 中偏序关系也是随机从对应的 I 中选择两个点构成且保证 R 中不存在两个相同的偏序关系.

- 对比算法

对每一组实验,本文将算法 FE、FEBF 与算法 SBS 和 SFS 作对比,SBS[13]和 SFS[13]是同类问题的已有算法中性能最好的精确算法.本文不与其他的算法对比的原因是:(1) BBS[13]和 BFS[13]分别为 SBS 和 SFS 的优化算法,主要是解决当 I 中包含的不是点而是类别信息时,随着每个类中包含顶点的个数增多,导致搜索空间增大的问题,当每个类中只包含一个顶点时,优化前后的算法性能相同;(2) NN[14]是一个启发式算法,它最终返回的路径是一条接近最优的路径,而本文的算法返回的是一条最优的路径.

- 其他

由于 SBS 和 SFS 都为应用于空间数据集上的算法,在空间数据集上可以直接利用经纬度坐标求得两点之间的欧氏距离,而本文所设计的算法均应用于普通的图数据,即,由点和边构成的数据,考虑到对比的公平性,本文对所有的算法均利用 CH 技术来求解两点之间的最短距离.

6.2 结果分析

- 查询效率

在查询集 Q1~Q5 上算法的运行结果如图 4 所示.可以看出:对于每一个数据集,SFS 的查询时间都大于其他算法的查询时间,FE 算法的查询时间明显小于 SBS 和 SFS 的查询时间,FEBF 算法的查询时间小于其他所有算法,即使在具有百万个顶点的数据集(FLA)上,FEBF 也能在 1s 左右得到查询的结果.因为查询集 Q1~Q5 具有相同个数的偏序关系,随着规则点集的规模逐渐增大,规则点集的基于规则$\mathcal{R}$的排列也逐渐增多,因此算法的查询时间也逐渐增多,图中的结果验证了这一点,即:从 Q1~Q5,所有算法的查询时间都逐渐增多.显然,改进后的 FEBF 算法比原始的 FE 算法性能有较大的提高.

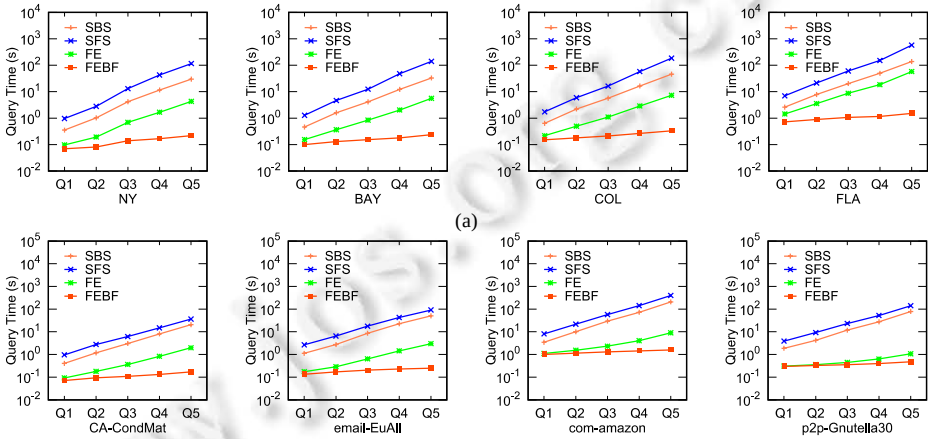


Fig.4 Query efficiency on Q1~Q5

图 4 在 Q1~Q5 上的查询效率

图 5 展示了算法在查询集 Q6~Q10 上的查询结果.对于所有的数据集,SFS 具有最大的查询时间,FEBF 算法的查询时间小于其他所有算法.因为查询集 Q6~Q10 具有相同个数的规则点,随着 R 中偏序关系的逐渐增多,规则点集的基于规则$\mathcal{R}$的排列将逐渐减少.可以看出对于所有算法,Q6~Q10 的查询时间都逐渐减小.

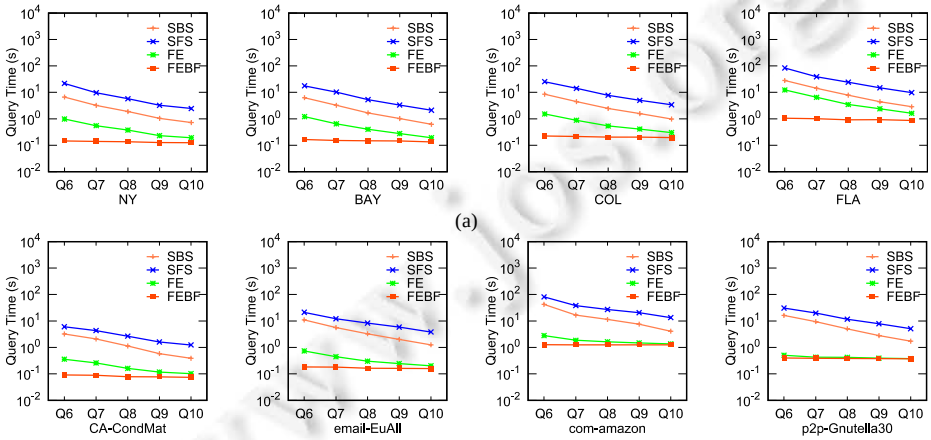


Fig.5 Query efficiency on Q6~Q10

图 5 在 Q6~Q10 上的查询效率

查询集 Q11~Q15 中包含的偏序关系个数为 0,此问题退化为广义旅行商问题,算法的查询结果如图 6 所示.因为 Q11~Q15 的偏序关系个数为 0,则规则点集的任意排列都为基于规则$\mathcal{R}$的排列,算法的查询时间显然大于在 Q1~Q5 上的查询时间.对于每一个数据集,FEBF 的查询时间都小于其他算法的查询时间.

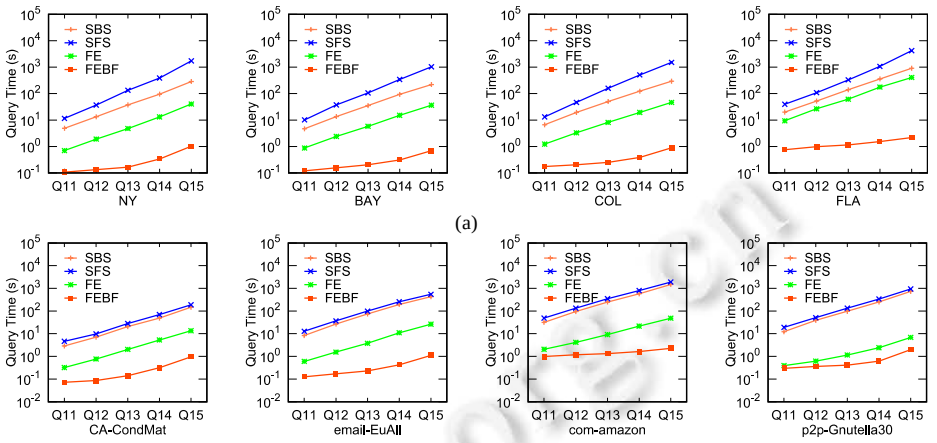


Fig.6 Query efficiency on Q11~Q15

图 6 在 Q11~Q15 上的查询效率

查询集 Q16~Q20 的偏序集可以构成一个全序集,即 I 中顶点的合法访问顺序已确定,此问题退化为最优序列路径查询问题,算法的查询结果如图 7 所示.由于每个类中仅包含一个顶点,因此,Q16~Q20 本质上为计算$|I|+1$个最短路径.从图中可知:对于每一个数据集,SBS 算法的查询时间都是最短的.这是因为 SBS 没有利用其他的优化策略来过滤子排列,而对于 Q16~Q20 来说,基于规则$\mathcal{R}$的子排列只有一个,因此,SBS 可以快速地求解得到问题的解.FE 和 FEBF 算法由于利用了一些过滤子排列的优化技术(Q16~Q20 并不需要对子排列进行过滤),从而增加了算法的运行时间.然而在现实生活中,这种查询几乎不存在(退化为普通的最短路径问题).

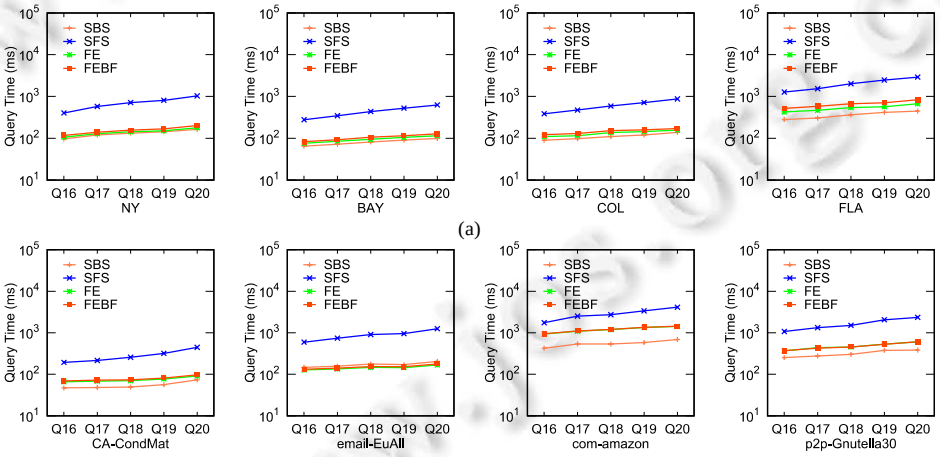


Fig.7 Query efficiency on Q16~Q20

图 7 在 Q16~Q20 上的查询效率

查询集 Q21~Q25 对应的规则$\mathcal{R}$规模较大,因此算法的运行时间较长.在此,本文仅给出算法 FEBF 的查询时间.从图 8 的结果来看:即使在规则$\mathcal{R}$的规模较大时,本文的 FEBF 算法也可以在一定时间内给出问题的解.

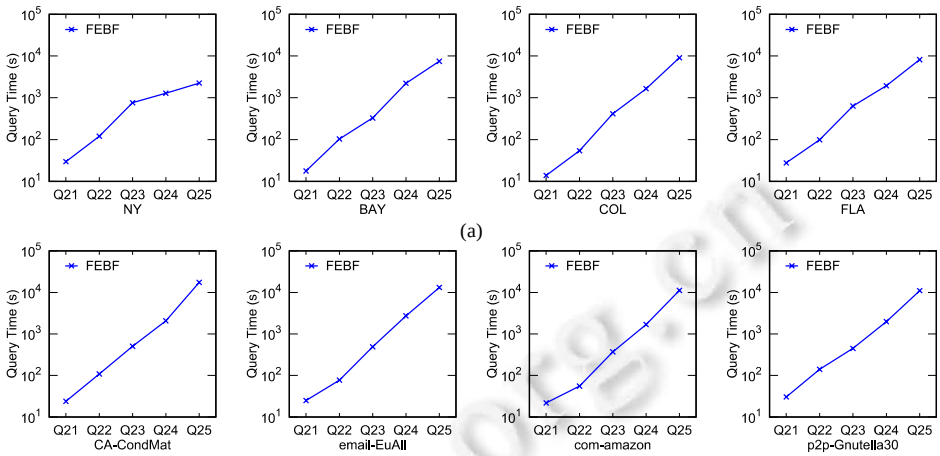


Fig.8 Query efficiency on Q21~Q25

图 8 在 Q21~Q25 上的查询效率

- 改进后的算法有效性

FEBF 算法是 FE 算法的改进算法,图 9 展示了 FE 和 FEBF 算法在查询集 Q11~Q15 上的加速比(加速比是指 FE 算法的查询时间与 FEBF 算法的查询时间的比值),结果表明:在每个数据集上,随着规则点个数的增多,加速比越来越大.显然,改进后的 FEBF 算法的运行效率极大地优于 FE 算法的运行效率.

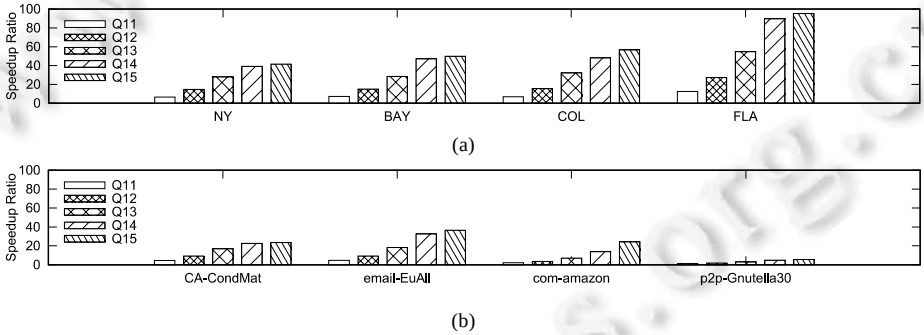


Fig.9 Speedup ratio of improved algorithm

图 9 改进后算法的加速比

FE 和 FEBF 算法的查询时间主要与生成的子排列个数有关:生成的子排列越多,查询时间则越长;反之,查询时间则越短.因此,本文用子排列的减少比 $\frac{N_{FE}-N_{FEBF}}{N_{FE}}$ (N_{FE} 表示 FE 算法生成的子排列个数,N_{FEBF} 表示 FEBF 算法生成的子排列个数)来反映 FE 和 FEBF 算法的性能优劣,结果如图 10 所示.对于道路交通网数据集,FEBF 算法生成的子排列个数比 FE 算法生成的子排列个数降低了至少 20%;对于非道路交通网数据集,FEBF 算法生成的子排列个数相比于 FE 算法生成的子排列个数也有所降低.

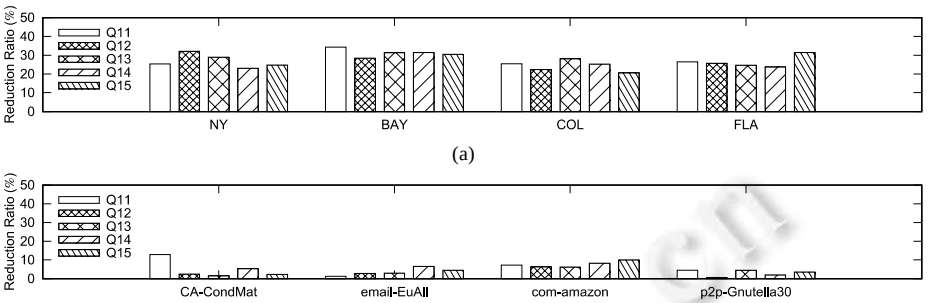


Fig.10 Reduction ratio of sub-permutation

图 10 子排列的减少比

- 优化技术的有效性

对于 FEBF 算法,本文提出了 3 个优化技术.图 11 展示了在查询集 Q11~Q15 上,FEBF 算法在未使用优化技术与使用优化技术两种情况下的加速比(即未使用优化技术的 FEBF 算法的查询时间与使用优化技术的 FEBF 算法的查询时间的比值),结果表明:在每个数据集上,相对于未使用优化技术的 FEBF 算法,使用优化技术的 FEBF 算法能够有效降低算法的查询时间.

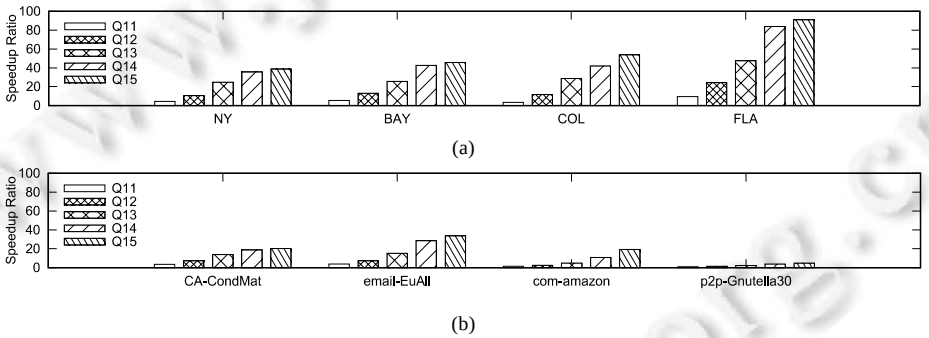


Fig.11 Effectiveness of optimizing techniques

图 11 优化技术的有效性

7 相关工作

本节介绍相关的工作,并把最短路径问题分为传统的最短路径问题和 ORQ 问题.ORQ 问题又细分为如下几类:已指定必须经过类别的访问顺序、未指定必须经过类别的访问顺序、部分指定必须经过类别的访问顺序.下面分类依次介绍相关的工作.

- 传统的最短路径问题

传统的最短路径问题具有不同的应用场景.稠密距离图(dense distance graph,简称 DDG)是平面图的分解图,即:将一个平面图分解为几个区域,每个区域所包含的顶点个数最多为 r,其中,$r<n$,n 为平面图中包含的顶点个数.Shay 等人[15]提出了一种求解稠密距离图上的最短路径的算法,并从理论上证明了所提的算法优于已有的最优算法.对于一个有向加权图,如果它的权重存在负值,称此图为实有向加权图,Fahim 等人[16]提出了一种实有向加权图上的单源最短路径算法,此算法可以快速检测路径中是否包含权值为负的环路,且性能优于已有的算法.对于规模较大的加权图,Qiu 等人[17]提出了一种并行最短路径距离查询框架——ParaPLL,ParaPLL 能够快速给出大规模加权图上的最短路径的距离,ParaPLL 通过使用共享内存和消息传递来实现最短路径距离的并行计算.给定起点 v_s 和终点 v_e,重路由序列是一系列从 v_s 到 v_e 的最短路径序列,使得在序列中相邻的两个最短路径只

有一个顶点不相同.Paul[18]研究了最短路径重路由问题(即:给定图 G 中从 v_s 到 v_e 的两条最短路径 P 和 Q,是否存在从 P 到 Q 的重路由序列),并给出了求解最短路径重路由问题的动态规划方法.进化图序列是指一系列静态图序列,它本质上是随时间变化的时序图,在每一时刻的图快照都对应于进化图序列中的一张静态图.Ren 等人[19]研究了进化图序列上的最短路径查询问题,并提出了解决方案框架——FVF.Jihye 等人[20]提出了基于磁盘的大规模动态图上的最短路径查询算法,此算法可以有效求解边的插入或删除以及边权重的增加或减少情况下的最短路径查询问题.Hasan[21]提出了针对稀疏图的 top-k 最短路径查询算法,该算法较已有的 top-k 最短路径算法有明显的性能提升.对于有向无环图,Matus 等人[22]提出了计算两点之间的近似最短路径算法,即:给定起点 v_s 和终点 v_e 以及权重阈值 L,该算法返回一条从 v_s 到 v_e 权重不超过 L 的路径.

- 已指定必须经过类别的访问顺序

已知必须经过的类别集合,若已规定所有类别的访问顺序,则 ORQ 问题称为类别全序的最优路径查询问题.R-LORD[23]是一个精确算法,用来求解空间数据集上类别全序的最优路径查询问题,假设 $p^*=(v_s,v_1,v_2,\ldots,v_r,v_e)$ 是上述问题的一个最优路径,则 p^*的任意一个后缀 p 在 p 中都是最短的,P 是满足下面两个条件的路径集合:(1) 以 p 的起点作为起点;(2) 与 p 相同的顺序访问 p 中的所有类.基于此,R-LORD 采用动态规划的方式逐步构建最优后缀表.在构建最优后缀表之前,R-LORD 利用贪心算法得到一条满足类别访问顺序的路径,并用此路径的权重作为阈值来过滤不可能成为最优路径的后缀.在构建最优后缀表时,R-LORD 利用椭圆剪枝技术来排除无法构成最优路径的子路径.Michael 等人提出了基于动态规划的算法求解类别全序的最优路径查询问题[24],此算法主要是逐步构建一个实值矩阵 X,X 是一个 r 行 g 列的矩阵,其中,r 表示必须经过的类别个数,g 表示规模最大的类中包含的顶点个数.矩阵的每个元素 $X[i,j]$是一个最优子路径的长度,此路径表示从起点到访问顺序为 i 的类别中的第 j 个顶点的最优子路径.此外,Michael 还设计了两个优化技术,分别为图结构优化和问题结构优化,并通过剪枝策略来解决由于类别规模过大导致算法运行时间过长的问题.由于本文所研究的问题对类别的访问顺序只做了部分的限制,即,并未完全指定每个类别的访问顺序,因此,上述两个算法都无法用来求解本文所研究的问题.

- 未指定必须经过类别的访问顺序

已知必须经过的类别集合,若没有规定任何类别的访问顺序,则 ORQ 问题称为类别无序的最优路径查询问题.对于空间数据集,Li 等人提出了几个近似算法来求解类别无序的最优路径查询问题[25].A_{NN}[25]是一个近似算法,其主要思想是:从单个起点构成的路径 p'开始,反复地从 G 中选择一个离 p'末尾顶点最近的点(记为 v^c),且 v^c 对应的类别 c 尚未被 p'访问,然后将 v^c 添加到 p'的末尾构成新的 p'.重复上述操作,直至所有的类别都已被 p'访问,最后将终点添加到 p'的末尾,从而构成一条包含指定所有类别的路径.A_{MD}[25]也是一个近似算法,它的主要思想是:对于每一个必须访问的类 c,从类 c 中选择一个顶点 v^c,使得 v^c 离起点和终点的距离之和小于其他任意点离起点和终点的距离之和,从每个类中选择得到的点构成集合 V_C,然后调用 A_{NN} 算法(此处在调用 A_{NN} 算法时,将 G 中的点替换为 V_C 中的点),从而得到一条包含指定所有类别的路径.LESS[26]是一个求解此类问题的精确算法,它的本质是将指定的所有类别进行全排列,每个类别在排列中的位置表示此类别在路径中被访问时所处的位置.对于每一个排列,按顺序依次从排列的类别中选择一个顶点,最终构成一条包含指定所有类别的路径.LESS 枚举出所有排列对应的所有路径,然后选择一条长度最小的路径作为问题的最优解.由于本文所研究的问题对类别的访问顺序做了部分的限制,而上述算法只能解决访问顺序没有限制的情况,因此,上述两个算法都无法用来求解本文所研究的问题.

- 部分指定必须经过类别的访问顺序

已知必须经过的类别集合,若已规定部分类别的访问顺序,则 ORQ 问题称为类别偏序的最优路径查询问题.NN[14]是求解类别偏序的最优路径查询问题的启发式算法,它最终返回一条接近最优的路线,算法的具体流程为:从单个起点构成的路径 p'开始,反复地从 G 中选择一个离 p'末尾顶点最近的点(记为 v^c),且 v^c 对应的类别 c 在图 G_Q(G_Q 是查询图且是一个有向无环图,G_Q 中的每个点都对应于 G 中的一个类别信息,G_Q 中每条有向边 (c,c')表示 G 中类别 c 的点先于类别 c'的点访问)中且入度为 0,然后,从 G_Q 中删除类别 c 以及与它相连的边,接

着将 v^c 添加到 p'的末尾构成新的 p'.重复上述操作,直至 G_Q 中不包含任何点,最终找到一条满足约束条件的路线.Li 等人提出了 4 个精确算法来求解在空间数据集上的此类问题[13],它们分别为 SBS,BBS,SFS 和 BFS. SBS 利用 R-LORD[23]中提出的后缀最优定理,逐步求解 1 后缀到 r 后缀(r 为必须经过的类别的个数),然后,从 r 后缀中选择一个路径长度最小的路径作为此问题的最优解.BBS 是 SBS 的优化算法,对于每一个必须经过的类别,BBS 首先将具有相同类别的点进行聚类,即,把彼此距离较近的一些点归为同一组,然后,BBS 以每一个组的点为单位,执行与 SBS 相同的操作,当每个类中包含的点的个数较多时,BBS 可以很大程度地提高算法的运行效率.SFS 本质上是采用分层回溯的方式求解问题的解,算法运行时所处的层数表示此时已得到的子路径所包含的类别个数.SFS 采用 NN[14]添加顶点的方式,每次选择离当前路径末尾顶点最近的一个点,每往路径中添加一个新的顶点,则意味着 SFS 进入到当前层的下一层,然后再执行相同的添加顶点的操作.当算法到达第 r 层后,则表明此算法已找到一条满足约束的路径,然后算法回溯到 r–1 层,将离当前路径末尾顶点次近的点添加到路径的末尾.重复上述回溯操作,直至得到所有满足约束的路径,最后返回一条最优的路径.BFS 是 SFS 的优化算法,与 BBS 类似,BFS 首先将具有相同类别的点进行聚类,即,把彼此距离较近的一些点归为同一组,然后,BFS 以每一个组的点为单位,执行与 SFS 相同的操作,最终得到问题的最优解.由于本文所研究的问题是基于普通的图数据而非空间数据,因此上述 4 个算法无法直接应用于本文所研究的问题,而且这些算法较少过滤掉不可能成为最优路径的子路径,使得算法的运行时间较长.

广义旅行商问题是传统旅行商问题的变种,带优先级约束的旅行商问题属于广义旅行商问题的一种,它可以描述为:已知 n 个城市之间的相互距离,现有一旅行商计划全部访问这 n 个城市,并且每个城市只能访问一次,在访问这些城市时必须满足一定的约束条件(例如,必须先访问城市 1 和城市 2,然后才能访问城市 3),最后,旅行商返回到出发的城市,目的是寻找一条旅行商行走的路线,使得此路线满足上述约束条件且具有最小的距离.Ascheuer 等人[27]提出了基于分支切割的算法来解决不对称的有约束条件的旅行商问题.Moon 等人[28]和 Wang 等人[29]分别采用遗传算法和整数规划的方式解决带有约束条件的旅行商问题;有优先约束的哈密顿路径问题(Hamiltonian path problems with precedence constraints)又称为顺序排序问题(sequential ordering problem),可描述为寻找由指定的起点前往指定的终点的最短路径查询问题,途中经过其他所有顶点有且只有一次,而且顶点之间存在先后顺序约束,Karan 等人[30]提出了一个基于分支界限法的算法来解决顺序排序问题.已有的解决广义旅行商问题的精确算法本质上为枚举出每一种可能的路径,因此这些算法不适用于大规模图上问题的求解,而本文所提的算法可以很好地应用于大规模图上的查询.

8 总 结

本文设计了一个基于最优子路径的前向扩展算法,该算法可以快速求解基于规则的最短路径查询问题,并设计了前向扩展算法的改进算法——基于最短优先策略的前向扩展算法.从实验的结果来看,基于最短优先策略的前向扩展算法对原始算法的查询效率有很大程度的提高.本文在真实的数据集上设计了大量的实验,并与已有的性能最好的算法作比较,实验结果表明,本文的算法大幅度优于已有的算法.

References:

[1] Pham TAN, Li X, Cong G, *et al*. A general graph-based model for recommendation in event-based social networks. In: Proc. of the 2015 IEEE 31st Int'l Conf. on Data Engineering (ICDE). IEEE, 2015. 567–578.

[2] Chen X, Lui JCS. Mining graphlet counts in online social networks. In: Proc. of the IEEE Int'l Conf. on Data Mining. IEEE, 2017. 71–80.

[3] Salamanis A, Kehagias DD, Filelis-Papadopoulos CK, *et al*. Managing spatial graph dependencies in large volumes of traffic data for travel-time prediction. IEEE Trans. on Intelligent Transportation Systems, 2016,17(6):1678–1687.

[4] Hartsperger ML, Blöchl F, Stümpflen V, *et al*. Structuring heterogeneous biological information using fuzzy clustering of *k*-partite graphs. Bmc Bioinformatics, 2010,11(1):1–15.

[5] Wang T, Li WS. A fast low-cost shortest path tree algorithm. Ruan Jian Xue Bao/Journal of Software, 2004,15(5):660−665 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/660.htm

[6] Tang JT, Wang T, Wang J. Shortest path approximate algorithm for complex network analysis. Ruan Jian Xue Bao/Journal of Software, 2011,22(10):2279−2290 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3924.htm [doi: 10.3724/SP.J.1001.2011.03924]

[7] Maier M, Rattigan M, Jensen D. Indexing network structure with shortest-path trees. ACM Trans. on Knowledge Discovery from Data (TKDD), 2011,5(3):15.

[8] Zhang X, Chan FTS, Yang H, *et al*. An adaptive amoeba algorithm for shortest path tree computation in dynamic graphs. Information Sciences, 2017,405:123−140.

[9] Zhu CJ, Lam KY, Cheng RCK, *et al*. On using broadcast index for efficient execution of shortest path continuous queries. Information Systems, 2015,49:142−162.

[10] Wang S, Xiao X, Yang Y, *et al*. Effective indexing for approximate constrained shortest path queries on large road networks. Proc. of the VLDB Endowment, 2016,10(2):61−72.

[11] Liu H, Jin C, Yang B, *et al*. Finding top-*k* shortest paths with diversity. IEEE Trans. on Knowledge and Data Engineering, 2018, 30(3):488−502.

[12] Geisberger R, Sanders P, Schultes D, *et al*. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: Proc. of the Int'l Workshop on Experimental and Efficient Algorithms. Berlin, Heidelberg: Springer-Verlag, 2008. 319−333.

[13] Li J, Yang YD, Mamoulis N. Optimal route queries with arbitrary order constraints. IEEE Trans. on Knowledge & Data Engineering, 2013,25(5):1097−1110.

[14] Chen H, Ku WS, Sun MT, *et al*. The multi-rule partial sequenced route query. In: Proc. of the 16th ACM SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems. ACM Press, 2008. 10.

[15] Mozes S, Nussbaum Y, Weimann O. Faster shortest paths in dense distance graphs, with applications. Theoretical Computer Science, 2018,711:11−35.

[16] Ahmed F, Anzum F, Islam MN, *et al*. A new algorithm to compute single source shortest path in a real edge weighted graph to optimize time complexity. In: Proc. of the 2018 IEEE/ACIS 17th Int'l Conf. on Computer and Information Science (ICIS). IEEE, 2018. 185−191.

[17] Qiu K, Zhu Y, Yuan J, *et al*. ParaPLL: Fast parallel shortest-path distance query on large-scale weighted graphs. In: Proc. of the 47th Int'l Conf. on Parallel Processing. ACM Press, 2018. 2.

[18] Bonsma P. Rerouting shortest paths in planar graphs. Discrete Applied Mathematics, 2017,231:95−112.

[19] Ren C, Lo E, Kao B, *et al*. Efficient processing of shortest path queries in evolving graph sequences. Information Systems, 2017,70: 18−31.

[20] Hong J, Park K, Han Y, *et al*. Disk-based shortest path discovery using distance index over large dynamic graphs. Information Sciences, 2017,382:201−215.

[21] Jamil HM. Efficient top-*k* shortest path query processing in sparse graph databases. In: Proc. of the 7th Int'l Conf. on Web Intelligence, Mining and Semantics. ACM Press, 2017.

[22] Mihalák M, Šrámek R, Widmayer P. Approximately counting approximately-shortest paths in directed acyclic graphs. Theory of Computing Systems, 2016,58(1):45−59.

[23] Sharifzadeh M, Kolahdouzan M, Shahabi C. The optimal sequenced route query. VLDB Journal, 2008,17(4):765−787.

[24] Rice MN, Tsotras VJ. Engineering generalized shortest path queries. In: Proc. of the IEEE Int'l Conf. on Data Engineering. IEEE Computer Society, 2013. 949−960.

[25] Li F, Cheng D, Hadjieleftheriou M, *et al*. On trip planning queries in spatial databases. In: Proc. of the Int'l Symp. on Spatial and Temporal Databases. Berlin, Heidelberg: Springer-Verlag, 2005. 273−290.

[26] Rice MN, Tsotras VJ. Exact graph search algorithms for generalized traveling salesman path problems. In: Proc. of the Int'l Symp. on Experimental Algorithms. Berlin, Heidelberg: Springer-Verlag, 2012. 344−355.

[27] Ascheuer N, Jünger M, Reinelt G. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. Computational Optimization & Applications, 2000,17(1):61−84.

[28] Moon C, Kim J, Choi G, *et al*. An efficient genetic algorithm for the traveling salesman problem with precedence constraints. European Journal of Operational Research, 2002,140(3):606–617.

[29] Wang X, Regan AC. The traveling salesman problem with separation requirements. European Journal of Operational Research, 2002,140(5).

[30] Karan M, Skorin-Kapov N. A branch and bound algorithm for the sequential ordering problem. In: Proc. of the 34th Int'l Convention. IEEE, 2011. 452–457.

附中文参考文献:

[5] 王涛,李伟生.低代价最短路径树的快速算法.软件学报,2004,15(5):660−665. http://www.jos.org.cn/1000-9825/15/660.htm

[6] 唐晋韬,王挺,王戟.适合复杂网络分析的最短路径近似算法.软件学报,2011,22(10):2279−2290. http://www.jos.org.cn/1000-9825/3924.htm [doi: 10.3724/SP.J.1001.2011.03924]

李忠飞(1992−),男,硕士,CCF 学生会员,主要研究领域为图数据管理,图挖掘.

王鑫(1981−),男,博士,副教授,CCF 高级会员,主要研究领域为知识图谱数据管理,图数据库,大规模知识处理.

杨雅君(1983−),男,博士,讲师,CCF 专业会员,主要研究领域为图数据管理,图挖掘.