软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):3115−3136 [doi: 10.13328/j.cnki.jos.006285]

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

动态网络中多规则的最短路径查询算法*

李艳红[1], 王 猛[1], 李国徽[2], 罗昌银[3,4,5], 杜小坤[1]

[1](中南民族大学 计算机科学学院, 湖北 武汉 430074)

[2](华中科技大学 软件学院, 湖北 武汉 430074)

[3](人工智能与智慧学习湖北省重点实验室(华中师范大学), 湖北 武汉 430079)

[4](国家语言资源监测与研究网络媒体中心, 湖北 武汉 430079)

[5](华中师范大学 计算机学院, 湖北 武汉 430079)

通信作者: 罗昌银, E-mail: changyinluo@mail.ccnu.edu.cn

摘 要: 最佳排序路径查询, 是智能交通中的热点问题. 在实际的应用中, 由于最佳排序路径查询有许多限制条件, 现有的算法不能有效地解决动态网络中受限制的路径查询问题. 为了解决动态网络中最佳排序路径查询问题, 用规则表示每个限制条件, 提出了一种新的最佳排序路径查询形式, 即多规则的最短路径查询. 提供了统一的框架, 该框架包含了路径集合查询和最短路径查询. 在路径集合查询部分, 为了高效地查询出满足多规则的路径集合, 在广义规则树的基础上, 提出一种新的树的遍历方式, 即树的继承全遍历; 并基于树的继承全遍历思想,提出一种剪枝技术, 对路径集合进行删减, 最后求得候选路径集合. 在最短路径查询部分, 提出一种基于动态阈值的最短路径搜索方法. 通过两个真实的动态道路网络的实验验证, 所提出的算法能够高效地解决多规则的最短路径查询问题.

关键词: 动态网络; 最短时间路径查询; 动态阈值; 预处理; 树的遍历

中图法分类号: TP393

中文引用格式: 李艳红, 王猛, 李国徽, 罗昌银, 杜小坤. 动态网络中多规则的最短路径查询算法. 软件学报, 2022, 33(8): 3115−3136. http://www.jos.org.cn/1000-9825/6285.htm

英文引用格式: Li YH, Wang M, Li GH, Luo CY, Du XK. Multi-rule Shortest Path Query Algorithm in Time-dependent Networks. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 3115−3136 (in Chinese). http://www.jos.org.cn/1000-9825/6285.htm

Multi-rule Shortest Path Query Algorithm in Time-dependent Networks

LI Yan-Hong[1], WANG Meng[1], LI Guo-Hui[2], LUO Chang-Yin[3,4,5], DU Xiao-Kun[1]

[1](College of Computer Science, South-Central Minzu University, Wuhan 430074, China)

[2](School of Software Engineering, Huazhong University of Science and Technology, Wuhan 430074, China)

[3](Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning (Central China Normal University), Wuhan 430079, China)

[4](National Language Resources Monitor & Research Center for Network Media, Wuhan 430079, China)

[5](School of Computer, Central China Normal University, Wuhan 430079, China)

Abstract: The optimal sequenced path query is a hot topic in the intelligent transportation. In practical applications, the existing approaches cannot effectively solve these constrained path query problems in the time-dependent network because of the constraints of the optimal sequenced path query. This study employs the rules to stand for the constraints. In order to solve the shortest travel time problem of the constrained path in the time-dependent network, a new query form of the optimal sequenced path, namely the multi-rule based

* 基金项目: 国家自然科学基金(61572215, 61772562); 教育部人文社科基金(20YJCZH111); 湖北省自然科学基金(2017CFB135); 中央高校基本科研业务费项目(CCNU20ZT013)

收稿时间: 2020-03-25; 修改时间: 2020-06-27, 2020-10-25; 采用时间: 2020-12-11

shortest path query, is studied. This study provides a unified framework, which includes the path set query and the shortest path query. In order to efficiently retrieve the path set satisfying multiple rules, a new tree traversal method, inheritance and traversal of trees, is proposed on the basis of generalized rule tree. Based on the idea of tree inheritance and traversal, a pruning method is proposed to prune the path set, and finally, the candidate path set is obtained. In the shortest path query part, a dynamic threshold method is proposed. Extensive experiments with two real data offer evidence that the proposed solutions can solve the problem of multi-rule based shortest path query.

Key words: time-dependent networks; shortest time path query; dynamic threshold; preprocessing; traversal of trees

最短路径查询问题一直是研究者关注的热点问题, 它在导航、物流运输、机器人路径规划和路由算法等领域中得到了应用. 近年来, 随着智能交通的广泛实施, 人们对动态道路网络中最短时间路径问题的关注越来越高, 对查询的限制也越来越多, 比如对经过地点数量的限制、对地点之间的访问顺序限制[1,2]、用户的偏好[3−5]等, 由此产生了最佳排序路径问题(optimal sequenced route, OSR).

OSR 的目标是搜索一条距离或时间最短的路径, 该路径从给定的源位置开始, 以特定的顺序经过多个地点, 且经过顺序取决于这些地点所属的类型[6]. 例如, 某用户发起 OSR 查询:从家出发, 先到银行, 然后到商场和餐馆. 现实中可供用户选择的银行、商场、餐馆不止一个, 它们代表的是不同的类别, 而不是具体的位置. 其次, 用户需要到银行取钱后才能去其他地点. 因此, 规划的最短路径除了满足时间要求外, 还需满足如下要求: 第 1 个访问地点是银行, 第 2 个和第 3 个访问地点要在餐馆和商场两类中选择. 显然, 满足用户需求的路径有两种组合: (1) 家→银行→餐馆→商场; (2) 家→银行→商场→餐馆. 因此, 搜索多类别的、存在顺序关系的最佳排序路径可分为两步: (1) 找到满足用户要求的路径集合; (2) 从这些集合中找出时间最短的路径.

本文研究的 OSR 具有 3 个限制条件: (1) 多类别; (2) 各类别中存在多个可供选择的地点, 每个类别中只能且必须选择一个地点访问; (3) 部分类别间存在先后顺序. 本文采用规则表示每个限制条件, 将具有多个规则限制的 OSR 称为满足多规则的最短路径查询问题. 如图 1 所示, 用户需要在 7 个位置 P_1~P_7 进行最短路径查询. 其中, P_1 为路径的起点和终点, P_2~P_7 为路径需要选择经过的点. P_2~P_7 每个位置均对应一个类别, 例如 P_2 和 P_4 是银行, P_3 和 P_5 是商场, P_6 和 P_7 是餐馆. 各类别的访问顺序是: 从 P_1 出发, 先去银行(P_2 或 P_4), 然后无先后顺序地去商场(P_3 或 P_5)和餐馆(P_6 或 P_7), 最后回到 P_1. 本文研究的问题遵循一个假设, 即起点和终点不对应类别, 且起点和终点均唯一(它们可以为同一个地点, 也可以是不同的两个地点), 起点和终点的访问顺序固定为第 1 个访问和最后一个访问. 满足多规则的最短路径满足以下要求.

(1) 能够访问所有类别且满足访问顺序;

(2) 每个类别中选择一个地点;

(3) 查询的路径在满足条件(1)和条件(2)的所有路径中时间代价最小.

图 1 多规则最短路径查询实例

目前, 各地图供应商提供的导航服务大多是两个地点间的路径导航, 很少提供包含多个地点的路径查询. 此外, 现有对多规则的最短路径查询问题的研究大部分集中在静态网络, 很少考虑动态网络. 然而静态网络

中的相关研究不适用于动态网络, 主要原因如下.

(1) 静态网络中边的权值固定不变, 而动态网络中边的权值随时间发生变化;

(2) 动态网络中, 每个结点存在时间消耗, 且随时间而变化. 例如, 用户到同一家餐馆就餐, 若到达时间是就餐高峰期, 则用户需要等待的时间会相对较长. 另外, 不同类别地点上的消耗时间一般是不同的, 在同种类别的不同地点的时间消耗可能也不相同. 例如: 一般到银行取钱比逛商场耗费的时间要少, 在等待客人较多的餐馆要比等待客人较少的餐馆耗费的时间要长.

动态网络更接近现实环境, 因此, 对动态网络中多规则的最短路径查询的研究具有实际意义, 对其研究面临的挑战如下.

在多规则的路径规划问题中, 目前主要有两种方式: 先验证后查询法和先查询后验证法. 它们的区别是: 先验证后查询法首先按照类别顺序关系进行排序, 得到符合多规则的候选路径集合, 然后对候选集合中的路径进行最短时间查询. 先查询后验证法是先查询最短子路径, 然后判断当前子路径是否满足多规则的约束: 若满足, 则最短子路径继续扩展直至到达终点; 否则, 对当前子路径进行符合多规则约束的操作. 目前还没有一种有效的先验证后查询法算法能够快速地求得满足多规则的路径集合, 人们常采用穷举验证法, 当有 n ($2\leqslant n$)个类别的地点时, 首先穷举所有类别组成的序列, 全排列后得到 $n!$个组合序列, 然后对这 $n!$个组合序列根据地点的顺序关系进行筛选, 最终得出符合多规则的路径集合. 边查询边验证法是以结点为基本元素进行路径扩展, 若每个类别包含一个元素时, 需要验证的子路径最好情况下为 1 个, 最坏为 $n!$个. 若每个类别中包含 k ($2\leqslant k$)个元素时, 穷举验证法需要在 $n!\cdot n\cdot k$ 个序列中查询符合多规则的路径集合, 边查询边验证法需要验证的序列最好情况下为 1 个, 最坏为$(n\cdot k)!/(n\cdot k-n)!$个. 随着 k 增大, $(n\cdot k)!/(n\cdot k-n)!$要远大于 $n!\cdot n\cdot k$. 由此可见, 穷举验证法和先查询后验证法对多规则的路径规划时, 筛选出符合多规则的路径十分困难;

在静态网络中, 由于路径的权值固定, 解决多规则的最短路径查询问题常见方法主要有向后扩展、向前扩展和从中间向两端扩展等方式. 而在动态网络中, 边的权值随时间变化而无法确定, 必须从前一个结点向后一个结点逐个扩展, 这就造成对路径中的结点扩展方法比较单一. 此外, 在路径规划完成后, 若出现道路拥塞等原因导致实际到达某个结点时超出了规划的时间, 需要对路径进行重新实时规划, 在动态网络中的重新实时规划代价比静态路网的更大.

为了解决多规则的最短路径查询问题, 本文在广义规则树[7]的基础上提出了一种树的继承全遍历方式, 并设计了继承全遍历算法(inheritance and traversal of trees, Tree_IT). Tree_IT 算法能够高效地计算出满足多规则的路径集合 PM. 提供了一种解决多规则最短路径查询的统一框架: 时间依赖网络中过滤查询最短路径(filter query shortest path in time dependent network)框架, 简称 TDF_SP 查询框架. 该框架由两部分组成:路径集合查询和最短路径查询. 在路径集合查询部分设计了剪枝技术, 该技术包含最短路径的权值上限查询算法(shortest path based on Tree_IT, Tree_IT_SP)和剪枝算法(pruning based on Tree_IT, Tree_IT_P). 剪枝技术的基本思想为: 运用 Tree_IT_SP 算法得到动态网络中多规则的最短路径的权值上限 $W_{os}\_G_{\max}$, 运用 Tree_IT_P 算法, 以 $W_{os}\_G_{\max}$ 为最大阈值, 对路径集合 PM 进行剪枝得到候选路径集 $PM_{\min}$. 在最短路径查询部分, 提出了在动态网络中进行最短路径查询的算法(the shortest path in time dependent road network, TDRN_SP), 该算法引入动态阈值思想, 在查询过程中, 以 $W_{os}\_G_{\max}$ 为初始阈值, 利用每次对候选路径的查询结果调整全局查询最优阈值, 进而提高查询效率. TDF_SP 查询框架的主要思想是: 在路径集合查询部分, 运用剪枝技术过滤出候选路径集 $PM_{\min}$; 在最短查询部分, 运用动态阈值思想在 $PM_{\min}$ 中过滤出最短路径. 通过两个动态道路网络的实验证明, 本文提出的 TDF_SP 查询框架具有高效性和可行性.

本文的主要贡献如下: 提出了树的继承全遍历方式, 并设计了对应的 Tree_IT 算法, 利用该算法, 可以高效地求得满足多规则的路径集合. 设计了动态网络中多规则的最短路径查询框架TDF_SP, 该框架包含路径集合查询和最短路径查询,并设计了用于剪枝的Tree_IT_SP算法和Tree_IT_P算法以及用于动态网络的最短路径查询的 TDRN_SP 算法. 精心设计和实施了对比实验, 实验结果证明了本文提出算法的正确性和高效性.

本文第 1 节给出相关工作的研究现状和存在的问题. 第 2 节介绍动态网络中多规则的最短路径查询问题

的相关定义. 第 3 节介绍树的继承全遍历, 并设计继承全遍历算法. 第 4 节提出 TDF_SP 查询框架, 分为路径集合查询和最短路径查询两部分: 在路径集合查询部分, 运用树的继承全遍历思想设计剪枝技术的两个算法——Tree_IT_SP 算法和 Tree_IT_P 算法; 在最短路径查询部分, 提出用于动态网络的最短路径查询的 TDRN_SP 算法. 第 5 节在真实道路网络中对本文提出的算法进行验证. 最后一节对全文总结.

1 相关工作

OSR 问题按照访问的类别是否存在顺序关系可以分为类别无序和类别有序两类, 按照类别中的地点数量可以分为类别中只有一个地点和类别中含有超过一个地点的问题.

1.1 类别无序

访问的类别无顺序要求的问题称为无序 OSR 问题. 若每个类别只包含一个地点时, 类似于中国邮路问题(Chinese postman problem, CPP)[8,9], CPP 不考虑对地点访问的先后顺序关系, 只需访问给定的全部的地点. Ma 等人[10]在对每个类别包含多个地点的研究中, 提出了一种基于 R 树的多类型最近邻(multi-type nearest neighbor, MTNN)查询解决方案, MTNN 查询利用页面级上限对 R 树的结点进行修剪, 最终规划出一条最短路径.

1.2 类别有序

访问类别有序问题即为有序 OSR 问题. 有序 OSR 问题又分为访问顺序全部有序和部分有序两类.

1) 全部有序

Rice 等人[11]探讨了访问指定全部有序状态的多个地点的路径规划问题, 提出了一种构建实值矩阵的方法, 通过最优子路径逐步扩展得到全局最优路径. Costa 等人[12]在研究动态网络中全部有序的 OSR 问题中, 将动态网络的路径最大权值构成一个静态网络, 利用在静态网络中得到启发值, 在动态网络中, 采用 A*搜索算法查找最短路径. Ahmadi 等人[13]研究了动态网络中最佳出发时间的 OSR 问题, 提出一种连续时间序列路由方法, 该方法以子路径的最小下限进行扩展, 并指定时间间隔, 查询出路径的最佳出发时间.

2) 部分有序

李忠飞等人[7]在对类别部分有序且类别中只有一个地点问题的研究中, 提出了构建广义规则树的方法, 并设计了一种向前搜索算法来查询最短路径, 在路径查询中, 通过广义规则树验证扩展到的结点是否满足顺序关系. Sharifzadeh 等人[6]提出了一种 LORD (light optimal route discoverer)方法, 利用阈值过滤掉不可能在最佳路线上的点, 然后从后向前查询最佳路线. 为了减少对不符合顺序关系的路径的计算, Li 等人[14]提出了向后搜索和向前搜索的两种方法, 并设计了 SBS (simple backward search), BBS (batch backward search), SFS (simple forward search)和 BFS (batch forward search)这 4 种算法来求解此类问题. Chen 等人[15]提出一种拓扑排序与邻接表相结合的 NNPSR (the nearest neighbor-based partial se-quenced route)方法, 首先根据多规则生成拓扑排序和邻接表, 在生成满足多规则的路径集合时, 先访问前置计数为零结点, 将此结点添加到列表 Lzero 中, 同时减少邻接列表中所有此结点的后续结点的前置计数, 直至将邻接列表中所有结点添加到列表 Lzero 中, 最后提出 3 种最大似然比查询算法用于计算最优路径. 虽然利用拓扑排序与邻接表的方法能够得到满足多规则的路径集合, 但每次访问结点时, 需要对邻接表中所有结点进行遍历来判断前置计数是否为 0, 而且访问后需要更新邻接表和后续结点前置计数, 因此在生成所有拓扑排序的所有路径时效率较低. Li 等人[16]提出一种改进的最小边界矩形算法, 用于更新历史轨迹数据与道路网络之间的关联, 并构建了一个两层的偏好网络, 用于挖掘偏好和旅行时间, 提出了3种近似算法来查询最佳路径. 对最短路径旅行时间具有实时性的研究中, 大多集中在基于交通信息的预测模型上, 通过分析历史交通数据和当前交通信息的预测, 获得更接近于现实的旅行时间[17−20].

在以上对类别有序的 OSR 问题的研究中, 文献[6,7,11,14,15]是在静态网络中进行的相关研究, 文献[12,13,16]讨论了动态路网中的OSR问题. 文献[11−13]研究了全部有序的OSR问题. 在部分有序的研究中, 文

献[6,7]采用先查询后验证的方式, 文献[14,15]采用先验证后查询的方式. 文献[6,7,14]所采用的向前扩展方法不适用于本文研究的动态网络. 文献[15,16]均采用近似算法, 都不能精确地查询出最短路径. 文献[17–20]均是基于预测模型的研究, 更多的考虑交通信息预测模型的有效性和高效性, 而不能对规划的最短路径能否满足顺序关系进行高效的判断.

综上可知, 目前在多规则的最短路径的研究中, 至少有以下一个局限性.

(1) 未考虑不同类别的地点之间的顺序限制;

(2) 未考虑同一个类别中包含多个地点供选择;

(3) 只在静态网络中研究, 未能考虑动态网络中的情况;

(4) 理想化了到达时间, 没有考虑在规划的时间内到未到达, 需要重新实时规划的场景;

(5) 在先验证后查询的方法中, 计算满足多规则的路径集合时, 计算复杂度为阶乘级, 难以应用到复杂的网络中;

(6) 在先查询后验证的方法中, 路径扩展过程中, 查询下一个地点时盲目扩展, 每扩展一个结点均要验证是否符合多规则, 计算量大.

针对上面的局限性, 本文设计了一种用于解决动态网络中的多规则的最短路径查询框架, 该框架不仅能高效的解决动态网络中多规则的最短路径问题, 而且在重新实时规划最短路径时效率更高.

2 问题描述及相关定义

本文研究的多规则的最短路径查询问题是基于动态 FIFO (first in first out)网络的研究, 本节首先介绍动态网络和 FIFO 网络的有关定义, 然后给出多规则的相关定义, 最后建立动态网络中多规则最短路径查询的模型.

定义 2.1(动态网络). 又称为时间依赖网络、时变网络. 将动态网络定义为图 $G(V,E,T)$, 其中, $V=\{v_1,v_2,...,v_n\}$为网络的结点集合, E 为边的集合, $T=f_{ij}(t)$是任意一条边 $e(v_i,v_j)\in E$ 在 t 时刻的权值, 即在 t 时刻, 从结点 v_i 到结点 v_j,所用的时间. $f_{ij}(t)$为非负实数, 是在时间 $t\in[t_0,t_u]\subset R$ 区间上一个矩阵函数(可为连续函数或离散函数).

定义 2.2(FIFO 网络). 将动态网络模型分成两类: FIFO 网络模型和非 FIFO 网络模型. 在图 $G(V,E,T)$中, 边 $e(v_i,v_j)$的权值函数 $f_{ij}(t)$为连续可微时, 若对于$\forall t$, $f_{ij}(t)$满足 $f_{ij}(t)\leqslant \Delta t+f_{ij}(t+\Delta t)$, 则称边 e 为 FIFO 边; $f_{ij}(t)$为离散函数时, 若对于$\forall t$, $\forall s\in S$(S 表示时间离散化后的时间间隔)且 $s<t$, 满足 $s+f_{ij}(s)<t+f_{ij}(t)$, 则边 e 为 FIFO 边. 不满足上述关系的边 e 为非 FIFO 的边. 如果网络中$\forall e\in E$ 均具有 FIFO 的性质, 则该网络称为 FIFO 网络; 如果网络中$\exists e\in E$ 不具备 FIFO 的性质, 该网络称为非 FIFO 网络. 具体性质和证明参考文献[21].

此外, 本文的研究中, $f_{ij}(t)=f_{ji}(t)$, 表示从边 e 任意一个端点出发, 通过 e 到达另一个端点与反方向通过 e 耗费的时间相等.

定义 2.3(多规则). 用 M 表示, M 定义为三元组(I,Q,Z), 其中,

(1) I 为类别集合, $I=\{I_1,I_2,...,I_n\}$. 其中, 每个类别 $I_i(i=1,2,...,n)$由 k 个不同的结点构成, $I_i=(v_1,v_2,...,v_k)$. 任意两个不同类别中的结点可以全部相同或部分相同;

(2) Q 是与类别集合 I 一一映射的结点的权值集合, $Q=\{Q_1,Q_2,...,Q_n\}$, Q_i 表示由 I_i 中的所有结点的权值构成的集合, $Q_i=(q_1(t),q_2(t),...,q_k(t))$. $q_i(t)$表示 t 时刻到达结点 v_i 后, 在结点 v_i 停留的时间. 在动态网络中, $q_i(t)$的性质和定义 2.1 和定义 2.2 中边 e 的性质相同, 随时间而改变且具有 FIFO 的性质;

(3) Z 是类别顺序关系构成的集合, $Z=(Z_1,Z_2,...,Z_m)$, $Z_i=(I_x,I_y)$, 其中, I_x, $I_y\in I$ 且 $I_x\neq I_y$, 表示先访问类别 I_x 后, 才能访问类别 I_y.

需要注意的是: Z 中不存在环, 即不存在一个顺序关系子集$\{(I_1,I_2),(I_2,I_3),...,(I_k,I_1)\}$, 其中, $I_1,I_2,I_3,...,I_k\in I$.

定义 2.4(基于多规则 M 的路径 P_t). 给定图 $G(V,E,T)$, 起点为 v_s, 终点为 v_e, 出发时刻为 t, $M=(I,Q,Z)$, 如果 P_t 满足下面两个条件, 则称为由 v_s 到 v_e 基于 M 的路径, 由所有基于 M 的路径 P_t 组成的路径集合记为 PM.

(1) P_t 除去起点 v_s 和终点 v_e 的子路径 $N=(v_1,v_2,...,v_n)$, 恰好经过类别集合 I 中每个类别中的一个结点, 即

$v_1 \in I_1, v_2 \in I_2, \ldots, v_n \in I_n$. 记 $P_t=v_s \rightarrow N \rightarrow v_e$;

(2) 路径 P_t 中的两个结点 v_i 和 v_j, 若 $v_i \in I_x$, $v_j \in I_y$, 在 Z 中, I_x 和 I_y 存在顺序关系(I_x,I_y), 则 P_t 中存在一条子路径 $v_i \rightarrow v_j$.

需要注意的是: v_s 和 v_e 不对应类别, 为给定图 G 中固定的两个地点, v_s 和 v_e 可以是同一个地点, 也可以是不同的两个地点, 且 v_s 和 v_e 的访问顺序均固定为第 1 个访问和最后一个访问.

定义 2.5(基于多规则 M 的最短路径 P_{mt}). 在 t 时刻, 在满足定义 2.4 的路径集合 PM 中, 权值最小的路径即为最短路径 P_{mt}, P_{mt} 权值为 $W(G,P_{mt},t)$. $W(G,P_{mt},t)$表示 t 时刻从 v_s 出发经过子路径 N, 到达终点 v_e 所耗费的时间, 包含了路径中各边耗费时间和中途各结点耗费的时间.

例 1: 给定动态无向图 G(图 2)和多规则 M, 起点为 v_s, 终点为 v_e, 出发时刻为 0, 各参数如表 1 所示, 求取 v_s 到 v_e 并满足 M 的最小时间的路径 P_{mt}. 即求取从起点 v_s, 经过类别集合 I_1 中的 v_1、类别集合 I_2 中的 v_2、类别集合 I_3 中的 v_3 或 v_4 任意点、类别集合 I_4 中的 v_5, 到达终点 v_e 的最小时间路径 P_{mt}.

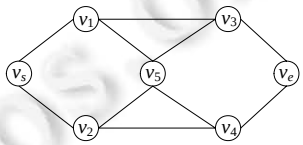


图 2 动态网络图 G

表 1 参数

	参数名称	值(t_f=t%11)
M	$I=\{I_1,I_2,I_3,I_4\}$	$I_1=(v_1)$, $I_2=(v_2)$, $I_3=(v_3,v_4)$, $I_4=(v_5)$
	$Q=\{Q_1,Q_2,Q_3,Q_4\}$	$Q_1=(10)$, $Q_2=(t_f+10)$, $Q_3=(t_f+5,10)$, $Q_4=(5)$
	$Z=\{Z_1,Z_2\}$	$Z_1=(I_1,I_3)$, $Z_2=(I_1,I_4)$
$f_{ij}(t)$	$f_{s1}(t)$	t_f+5
	$f_{s2}(t)$	45
	$f_{13}(t)$	50
	$f_{15}(t)$	t_f+5
	$f_{25}(t)$	t_f+5
	$f_{24}(t)$	t_f+5
	$f_{35}(t)$	t_f+30
	$f_{3e}(t)$	40
	$f_{45}(t)$	t_f+20
	$f_{4e}(t)$	5

3 树的继承全遍历

3.1 广义规则树

在类别部分有序的 OSR 问题中, 文献[7]采用先查询后判断的方法, 提出一种通过广义规则树判断查询到的子路径是否满足顺序关系的方法.

定义 3.1(广义规则树)[7]. 给定 $M(I,Q,Z)$、起点 v_s 和终点 v_e, 由 v_s, v_e 和类别集合 I 构成的、并满足下面两个条件的树称为广义规则树, 记为 *M-Tree*.

(1) v_s 为树的根结点, v_e 为唯一叶结点, 树中的每个结点均不同;

(2) *M-Tree* 树中, 除根结点 v_s 外, 若 I_x 为 I_y 的父结点, 则 Z 中有顺序关系(I_x,I_y).

广义规则树由文献[7]提出, 与普通树的主要区别是: 广义规则树允许结点可以有多个父结点, 且只有一个叶结点. 运用广义规则树判断查询到的路径 P 是否满足多规则的原理是: 路径 P 中任意结点 v_i, 若 v_i 为起点 v_s, 则 v_i 为 *M-Tree* 的根结点; 若 v_i 不为起点, 则 v_i 在 *M-Tree* 中的父结点必包含在路径 P 的子路径 v_s 到 v_i 中. 广义规则树的性质和构建算法具体细节见文献[7].

3.2 树的继承全遍历

多规则的最短路径查询中, 虽然运用广义规则树能够判断查询到的路径是否满足多规则, 但是需要对查询过程中产生的所有子路径进行判断, 这种先查询后判断的方式具有计算量大和子路径扩展盲目性的不足. 本文在广义规则树基础上提出一种先判断后查询的方法来解决此类问题: 通过广义规则树进行继承全遍历操作, 生成满足多规则的路径集合, 然后在此路径集合中查询出最短路径.

为使树的继承全遍历不仅能够遍历广义规则树, 也能遍历普通树(二叉树和多叉树), 本文采用的广义规则树是将定义 3.1 的广义规则树的唯一叶结点 v_e 去掉后的树. 此广义规则树解决多规则的最短路径查询时,继承全遍历得到路径集合 *PL*, 在 *PL* 中的所有路径后追加原广义规则树的唯一叶结点 v_e 组成的满足多规则 *M* 路径集合 *PM*.

下面首先介绍树的继承全遍历的定义, 然后给出树的继承全遍历算法.

定义 3.2(树的继承定义). 满足下面直接继承和断绝继承的操作称为继承, 将下述情形(1)和情形(2)中 *C* 继承 *A* 的操作记为 $C \oplus A$.

(1) 直接继承: 树的根结点 *v* 的孩子结点不止 v_i 一个时, 将 *v* 除 v_i 以外的其他的孩子结点均转化为 v_i 的孩子结点, 并删除根结点 *v*, 形成以 v_i 为根结点的新树, 此过程称为直接继承. 如图 3 所示, 则 *Tree*-1 的孩子结点 *C* 直接继承根结点 *A* 的操作, 并得到 *C* 为根结点的新树 *Tree*-2;

(2) 断绝继承: 孩子结点 v_i 在对根结点 *v* 直接继承操作时, 若 v_i 和其兄弟结点存在共同的孩子结点 v_x 时, 需要结点 v_i 与结点 v_x 先断绝父子关系, 再进行直接继承, 此过程称为断绝继承. 如图 3 所示, 结点 *C* 继承 *A* 时, 由于 *C* 和 *D* 有共同的孩子结点 *E*, 则 *C* 要与 *E* 断父子关系后才能继承 *A*, 得到以 *C* 为根结点的新树 *Tree*-4.

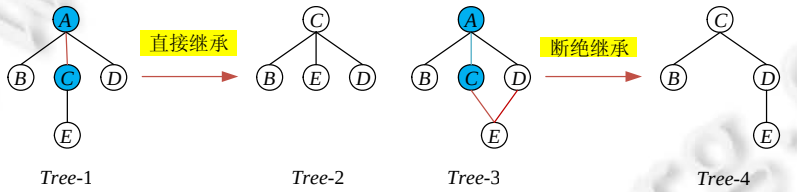


图 3 树的继承

定义 3.3(树的继承全遍历). 给定树 *tree*, 若树为空, 则空操作; 否则,

(1) 访问根结点;

(2) 从左到右依次对每个孩子结点进行继承操作. 其中的一个孩子结点继承操作时, 若产生不为空的新树, 对产生的新树重复步骤(1)和步骤(2), 直至新树为空. 当产生一棵新树为空时, 称之为完成树 *tree* 的一次继承子遍历, 将此过程中访问的根结点按照访问的先后顺序构成一个遍历子序列;

(3) 若步骤(2)中所有新树均为空, 即完成树 *tree* 的继承全遍历, 所有遍历子序列构成树 *tree* 的继承全遍历序列集合.

性质 3.1. 给定根结点为 v_s 的树 *tree*, *tree* 的继承全遍历为不少于一组、起点为 v_s 的不同遍历子序列构成的继承全遍历序列集合.

证明: 若 *tree* 中存在一个有多于 1 个孩子结点的结点 v_i, 假设 v_i 的两个孩子结点为 v_x 和 v_y. 根据定义 3.2 可知: 存在两种继承操作 $v_x \oplus v_i$ 和 $v_y \oplus v_i$, 分别形成以 v_x 为根结点、v_y 为孩子结点和以 v_y 为根结点、v_x 为孩子结点的两棵新树. 那么根据定义 3.3, 存在一组遍历子序列为 $v_s,\ldots,v_i,v_x,\ldots$和一组遍历子序列为 $v_s,\ldots,v_i,v_y,\ldots$; 若 *tree* 中所有结点的孩子结点均不多于 1 个, 则每次继承只产生一棵新树, 全遍历序列只有一组. 证毕. □

性质 3.2. 给定根结点为 *vs* 的树 *tree*, *tree* 的结点个数为 *N_tree*, 那么 *tree* 的继承全遍历得到的遍历集合中, 每组遍历子序列包含了 *tree* 的所有结点, 且每组序列的长度均等于 *N_tree*.

证明: 根据定义 3.3 可知: 在任意一次继承子遍历中, 每次仅访问树的根结点并对此根结点进行继承操

作, 完成一次继承子遍历产生了 N_tree-1 棵不为空且根结点互不相同的新树. 那么一组遍历子序列由 v_s 和其余 N_tree-1 个根结点构成, 可得一组遍历子序列的长度为 1+N_tree-1, 即为 N_tree. 证毕. □

性质 3.3. 给定根结点为 v_s 的树 $tree$, 若树中有一个结点 v_i 和它的一个孩子结点 v_x, 继承全遍历得到的序列集合中的任意一组遍历子序列中均包含序列 $v_i,...,v_x$, 均不包含序列 $v_x,...,v_i$.

证明: 由定义 3.3 可知: 在树的任意一次继承遍历过程中, 必存在一棵根结点为 v_i、且 v_x 为 v_i 孩子结点的新树, 那么由定义 3.3 首先访问根结点和性质 3.2 的每组遍历子序列包含了 $tree$ 的所有结点可知, 任意一组遍历子序列中均包含序列 $v_i,...,v_x$; 由定义 3.2 可知, 继承操作 $v_x \oplus v_i$ 后, 需要删除 v_i 构成以 v_x 为根结点的新树, 那么不会存在一棵根结点为 v_i、且 v_x 为 v_i 孩子结点的新树. 由此可知, 任意一组遍历子序列中均不包含序列 $v_i,...,v_x$. 证毕. □

树的继承全遍历算法如算法 1. 首先定义全局变量 P_{List} 为二元组(PL,$Tree$)构成的集合, 其中: PL 表示对树继承全遍历得到的一个遍历子序列, 初始值 v_s; $Tree$ 表示孩子结点继承父结点得到的新树, 初始值是以起点 v_s 为根结点的树 $M\text{-}Tree$(第 1 行、第 2 行). 定义变量 N_tree 表示给定树 $M\text{-}Tree$ 的所有结点总数量(第 3 行), 然后对 P_{List} 中每一个元素进行操作(第 4 行). 由性质 3.2 可知: 若某一元素 P_{Listi} 中的序列 PL 长度不等于 N_tree(第 5 行), 则表示还未得到某个继承全遍历序列, 则需要对当前的树进行继承操作, 进而更新相应的全遍历序列 PL 和新继承树 $Tree$, 并将它们并入 P_{List}(第 6 行–第 10 行). 对 P_{List} 中的 P_{Listi} 进行继承遍历操作后, 将此 P_{Listi} 在 P_{List} 中删除, 进而更新 P_{List}(第 11 行). 最后得到的 P_{List} 中的 PL 即是继承全遍历序列.

算法 1. $Tree\_IT(M\text{-}tree,v_s)$.

输入: 广义规则树 $M\text{-}tree$, 根结点 v_s;

输出: 树的继承全遍历序列的集合 $P_{List}.PL$.

1. P_{List} 为二元组集合{(PL,$Tree$)};
2. $P_{List} \leftarrow \{(v_s, M\text{-}Tree)\}$;
3. $N\_tree \leftarrow count(M\text{-}Tree)$;
4. **for each** $P_{Listi} \in P_{List}$ **do**
5. **if** $long(P_{Listi}.PL) \neq N\_tree$ **then**
6. $root \leftarrow P_{Listi}.Tree \leftarrow parent$;
7. **for each** $node \in root \rightarrow child$ **do**
8. $P_{Listi}.PL \leftarrow P_{Listi}.PL \cup node$;
9. $Tree \leftarrow node \oplus root$;
10. $P_{List} \cup (P_{Listi}.PL, Tree)$;
11. **end for**
12. $PL \leftarrow P_{List} - P_{Listi}$;
13. **end if**
14. **end for**
15. **return** $P_{List}.PL$

定义 3.4(子结点的继承全排列和子结点的继承全排列序列). 在树的继承全遍历算法中, 根结点 v_s 的一个孩子结点 v_i 进行继承时, 将 v_s 的其他孩子结点均继承为 v_i 的孩子, 并对 v_i 的所有孩子结点进行一次全排列的过程称为 v_i 结点的继承全排列, 记为 Θv_i, 得到的继承全排列序列称为 v_i 的继承全排列序列, 记为 $L(\Theta v_i)$.

例 2: 对例 1 构建的广义规则树(图 4)进行继承全遍历.

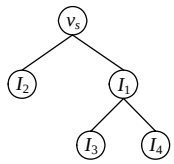


图 4 广义规则树

运用 Tree_IT 算法对图 4 的树进行继承全遍历过程如图 5 所示，得到 8 个继承全排序序列为 $\{(v_s,I_2,I_1,I_3,I_4),(v_s,I_2,I_1,I_4,I_3),(v_s,I_1,I_3,I_4,I_2),(v_s,I_1,I_3,I_2,I_4),(v_s,I_1,I_4,I_3,I_2),(v_s,I_1,I_4,I_2,I_3),(v_s,I_1,I_2,I_3,I_4),(v_s,I_1,I_2,I_4,I_3)\}$.

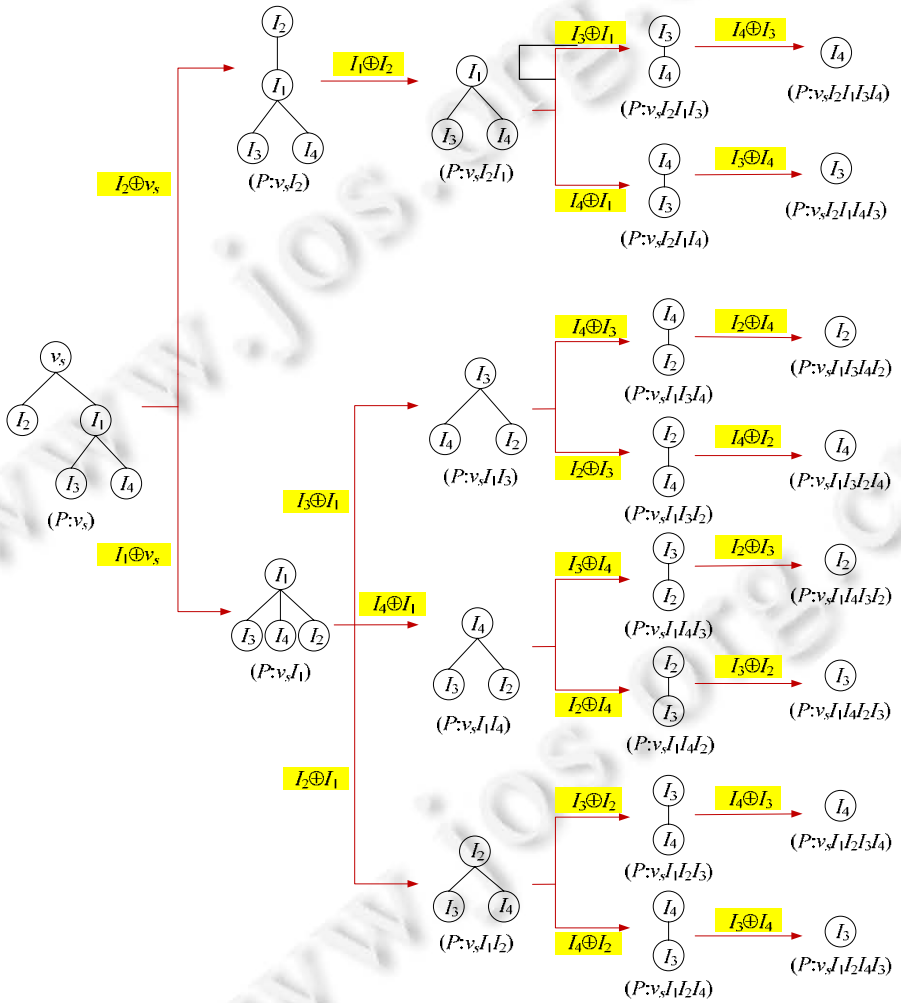


图 5 树的继承全遍历

例 3: 某树如图 4 构成的广义规则树，则对子结点 I_1 进行继承全排列操作，得到的继承全排列序列为 $\{(I_2,I_3,I_4),(I_2,I_4,I_3),(I_3,I_2,I_4),(I_3,I_4,I_2),(I_4,I_2,I_3),(I_4,I_3,I_2)\}$.

4 TDF_SP 查询框架

本节提出的 TDF_SP 查询框架由路径集合查询和最短路径查询两部分构成: 在路径集合查询部分，运用最短路径权值上限对满足多规则的路径集合进行剪枝，得到候选路径集合 $PM_{\min}$; 在最短路径查询部分，提出

了最短路径查询的算法, 该算法在候选路径集合 $PM_{\min}$ 中查询出最短路径.

4.1 理论基础

首先介绍 TDF_SP 查询框架的相关定理和推论; 然后基于该框架, 提出了多规则的最短路径查询步骤.

从定义 2.1 和定义 2.2 可知: 在动态网络中, 每条边的权值随时间变化而改变. 为了在动态网络中有效地解决最短路径问题, 提出了相关定理和推论, 分析了剪枝技术的原理. 本节的定理和推论遵循下面的假设: 在多规则的问题 $M(I,Q,Z)$中, 每个地点耗费时间 Q 和边的变化性质一样, 因此可以将 Q 的值附加到与其相接的边上. 所以在描述相关理论时, 为了便于阐述, Q 默认为 0.

定理 4.1. 给定图 $G(V,E,T)$, 将图 G 转换为两个静态网络, 取 E 中每条边的最小值构成图 $G_{\min}(V,E,L_{\min})$, 取 E 中每条边的最大值构成图 $G_{\max}(V,E,L_{\max})$, $L_{\min}$ 和 $L_{\max}$ 表示静态网络中边的权值, 则 t 时刻, 任意一条边 e 在图 G, $G_{\min}$ 和 $G_{\max}$ 的权值大小关系为 $W(G_{\min},e)\leqslant W(G,e,t)\leqslant W(G_{\max},e)$. 若需要在图 G 中查询一条起点为 v_s、终点为 v_e 的最短路径 P, 在 t 时刻, 路径 P 的权值为 $W(G,P,t)$, 则在图 $G_{\min}$ 和 $G_{\max}$ 中, 对应的路径 P 的权值分别为 $W(G_{\min},P)$, $W(G_{\max},P)$. 则有如下关系: $W(G_{\min},P)\leqslant W(G,P,t)\leqslant W(G_{\max},P)$.

证明: 设 t 时刻, 查询起点 v_s 到终点 v_e 的最短路径中途需要经过结点集合 $O=(v_1,v_2,...,v_n)$. 在图 G, $G_{\min}$ 和 $G_{\max}$ 中查询到的最短路径分别为 P, $PG_{\min}$ 和 $PG_{\max}$. 路径 P 在图 G, $G_{\min}$ 和 $G_{\max}$ 对应的权值分别为 $W(G,P,t)$, $W(G_{\min},P)$和 $W(G_{\max},P)$. 若 $v_i\in O\cup v_e$, 则路径 P 中必存在子路径 $P\_v_{s\to i}=v_s\to v_i$, 因为 t 时刻, 任意一条边 e 在图 G, $G_{\min}$ 和 $G_{\max}$ 的权值大小关系为 $w(G_{\min},e)\leqslant W(G,e,t)\leqslant W(G_{\max},e)$, 那么子路径 $P\_v_{s\to i}$ 在图 G, $G_{\min}$和图 $G_{\max}$ 的权值满足 $W(G_{\min},P\_v_{s\to i})\leqslant W(G,P\_v_{s\to i},t)\leqslant W(G_{\max},P\_v_{s\to i})$, 则 $W(G_{\min},P)\leqslant W(G,P,t)\leqslant W(G_{\max},P)$. 证毕. □

根据定理 4.1, 可以得到下面 4 个推论.

推论 4.1. 3 条路径 P, $P_{\min}$ 和 $P_{\max}$ 中, 各结点并不一定一一对应, 即: 3 条路径中的结点顺序不一定相同.

证明: 假设 3 条路径 P, $P_{\min}$ 和 $P_{\max}$ 中, 各结点一一对应. 若 $v_i,v_j\in O$, v_i 和 v_j 无顺序要求. 若路径 P, $P_{\min}$ 和 $P_{\max}$ 为路径 $v_s\to v_i\to v_j\to v_e$, 在图 $G_{\min}$ 和图 $G_{\max}$ 如图 6 所示, 则在 $G_{\max}$ 中查询的最短路径为 $v_s\to v_j\to v_i\to v_e$, 则假设中的 $P_{\max}=v_s\to v_i\to v_j\to v_e$ 与定理中 $G_{\max}$ 中的最短路径矛盾, 假设不成立. 因此, 3 条路径 P, $P_{\min}$ 和 $P_{\max}$ 中各结点并不一定一一对应. 证毕. □

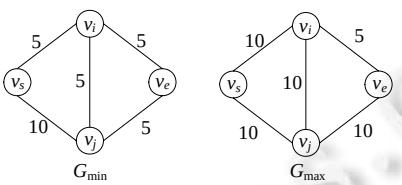


图 6 $G_{\min}$ 和 $G_{\max}$

推论 4.2. 3 条路径 P, $P_{G\min}$ 和 $P_{G\max}$ 的权值关系为: $W(G_{\min},PG_{\min})\leqslant W(G,P,t)\leqslant W(G_{\max},PG_{\max})$.

证明: 根据定理 4.1 的证明可知: 若 $v_i\in O\cup v_e$, 则路径 P, $PG_{\min}$ 和 $PG_{\max}$ 中均必存在从 v_s 开始的子路径 $P\_v_{s\to i}=v_s\to v_i$, $P\_G_{\min}\_v_{s\to i}=v_s\to v_i$, $P\_G_{\max}\_v_{s\to i}=v_s\to v_i$. 由推论 4.1 可知, $P\_v_{s\to i}$, $P\_G_{\min}\_v_{s\to i}$ 和 $P\_G_{\max}\_v_{s\to i}$ 各结点顺序不一定相同. 由定理 4.1 可得, 3 条子路径 $v_s\to v_i$ 的权值满足:

$$W(G_{\min},v_s\to v_i)\leqslant W(G,v_s\to v_i,t)\leqslant W(G_{max},v_s\to v_i),$$

则 $W(G_{\min},PG_{\min})\leqslant W(G,P,t)\leqslant W(G_{\max},PG_{\max})$. 证毕. □

推论 4.3. 路径 $PG_{\min}$ 和 $PG_{\max}$ 分别映射到图 G 中, 在 t 时刻, 路径 $P_{\min}$ 和 $P_{\max}$ 在 G 中的最小权值 $W(G,PG_{\min},t)$和 $W(G,PG_{\max},t)$与 $W(G,P,t)$的关系为: $W(G,P,t)\leqslant W(G,PG_{\min},t)$,$W(G,P,t)\leqslant W(G,PG_{\max},t)$.

证明: 由推论 4.1 可知: $PG_{\min}$ 和 $PG_{\max}$ 分别映射到图 G 中的路径不一定与 P 路径相同, 即不一定为图 G 中的最短路径, 则二者映射在图 G 中最小权值 $W(G,PG_{\min},t)$和 $W(G,PG_{\max},t)$与 $W(G,P,t)$的关系为: $W(G,P,t)\leqslant W(G,PG_{\min},t)$, $W(G,P,t)\leqslant W(G,PG_{\max},t)$. 证毕. □

推论 4.4. 假设图 G 中, t 时刻, 从起点 v_s 经过 O 中所有结点到终点 v_e 有子路径 $P\_v_{s\to i}=v_s\to v_i$, 路径 $P\_v_{s\to i}$ 中, 映射到图 $G_{\max}$ 中的最小权值为 $W(G_{\max},P\_v_{s\to i})$, 映射到图 $G_{\min}$ 的最小权值为 $W(G_{\min},P\_v_{s\to i})$.

若 $W(G_{\min},P\_v_{s\to i})>W(G_{\max},P\_v_{s\to i})$, 那么路径 $P_v_{s\to i}$一定不是 P 的子路径; 反之, $P_v_{s\to i}$可能是 P 的子路径.

证明: 由推论 4.3, 因为 P 为图 G 中的最短路径, 则满足 $W(G_{\min},P)\leqslant W(G,P,t)\leqslant W(G_{\max},P)$, 由此可知: 若 $W(G_{\min},P_v_{s\to i})>W(G_{\max},P_v_{s\to i})$时, $P_v_{s\to i}$一定不为 P 的子路径; 反之, $P_v_{s\to i}$可能是 P 子路径. 证毕. □

在动态网络 $G(V,E,T)$中, 起点为 v_s, 终点为 v_e. 对于基于多规则 M 的路径集合 PM 而言, 设 PM 在 $G_{\max}$ 中的所有路径的权值的最小值为 $W_{os}\_G_{\max}$, 在 $G_{\min}$ 中的所有路径的权值的最小值为 $W_{os}\_G_{\min}$. 在图 G 中, 基于多规则 M 最短路径 P_{mt}和任意一条路径 P_{mti}, 由推论 4.2 可得: 在 t 时刻, 路径 P_{mt}和 P_{mti}的最小权值为 $W(G,P_{mt},t)$和 $W(G,P_{mti},t)$, 则有:

$$W_{os}\_G_{\min}\leqslant W(G,P_{mt},t)\leqslant W(G,P_{mti},t)\leqslant W_{os}\_G_{\max} \tag{1}$$

通过公式(1)可知: P_{mt}的权值上限为 $W_{os}\_G_{\max}$, 若路径 P_{mti}的权值 $W(G,P_{mti},t)$大于 $W_{os}\_G_{\max}$, 则 P_{mti}肯定不是最短路径.

在图 $G_{\min}$中, 集合 PM 中路径权值小于 $W_{os}_G_{\max}$的路径构成的集合为 $PM_{\min}$, 由推论 4.4 可得路径 P_{mt}、集合 PM, $PM_{\min}$的关系为

$$P_{mt}\in PM_{\min},\ PM_{\min}\subseteq PM \tag{2}$$

TDF_SP 分为路径集合查询和最短路径查询两部分: 在路径集合查询部分, 首先在 $G_{\max}$ 中得到最短路径的权值上限 $W_{os}\_G_{\max}$, 根据公式(1)和公式(2)对路径集合 PM 进行剪枝, 得到候选路径集合 $PM_{\min}$; 在最短路径查询部分, 通过公式(1)可知, 最短路径的权值一定小于 $W_{os}\_G_{\max}$, 因此设置阈值 U,用 $W_{os}\_G_{\max}$ 对 U 进行初始化, 即 $U=W_{os}\_G_{\max}$. 在候选路径集合 $PM_{\min}$ 中, 以 U 为权值上限遍历路径, 在遍历出的路径 P_{mti}的最小权值 $W(G,P_{mti},t)$小于 U 时, 更新 U 为 $W(G,P_{mti},t)$, 继续以 U 为权值上限对 $PM_{\min}$中的路径进行查询, 直至将 $PM_{\min}$中的路径完全查询, 那么权值为 U 的路径即为最短路径 P_{mt}.

通过以上的分析, 动态网络中多规则的最短路径查询框架如下.

(1) 路径集合查询: ① 将动态网络 G 根据权值上下界划分为静态网络 $G_{\max}$和 $G_{\min}$, 并构建广义规则树; ② 在静态网络 $G_{\max}$中计算最短路径的权值上限 $W_{os}\_G_{\max}$; ③ 以 $W_{os}_G_{\max}$为阈值, 通过剪枝算法对路径集合 PM 进行剪枝, 得到候选路径集合 $PM_{\min}$;

(2) 最短路径查询: 以 $W_{os}_G_{\max}$为初始阈值, 对候选路径集合 $PM_{\min}$在动态网络 G 中进行逐一遍历, 进而求得最短路径 P_{mt}.

4.2 路径集合查询

4.2.1 权值上限查询

本节提出 Tree_IT_SP 算法, 用于查询静态网络下的最短路径和其权值. 通过公式(1)可知, 该权值即为动态网络中最短路径权值上限 $W_{os}\_G_{\max}$. Tree_IT_SP 算法如算法 2.

算法 2. *Tree_IT_SP*($G_{\max}$,*M-Tree*,v_s,v_e).

输入: 图 $G_{\max}$, 广义规则树 *M-Tree*, 起点 v_s, 终点 v_e;

输出: 最短路径 $OpenL_{\min}.P$ 和其权值 $OpenL_{\min}.W$.

```
1.  OpenL 定义为三元组集合{(PL,PL_Tree,PL_W)}, OpenL_min 定义为三元组(P,Tree,W);
2.  OpenL←{(v_s,M-Tree,0)}, OpenL_min←(v_s,M-Tree,0);
3.  while OpenL_min.Tree≠0 do
4.    if length(OpenL_min.P−v_s)=length(M.I) then
5.      Tree←0;
6.      P←(OpenL_min.P+v_e);
7.      W←(OpenL_min.W+w(G_max,P));
8.      OpenL←OpenL+(P,Tree,W);
9.    else
10.     root←OpenL_min.Tree←parent;
```

```
11.      for each I∈root←child do
12.        Tree←I⊕root;
13.        for each v∈I do
14.          P←(OpenL_min.P+v);
15.          W←OpenL_min.W+W(G_max,P);
16.          OpenL←OpenL∪(P,Tree,W);
17.        end for
18.      end for
19.      OpenL←(OpenL−OpenL_min);
20.    end if
21.    OpenL_min←OpenL.Min(OpenL.W);
22.  end while
23.  return OpenL_min.P, OpenL_min.W;
```

首先定义参数 *OpenL* 为三元组(*PL*,*PL_Tree*,*PL_W*)构成的集合和 $OpenL_{min}$ 定义为三元组(*P*,*Tree*,*W*). *OpenL* 中: *PL* 表示对树继承全遍历得到的序列, 初始序列为 v_s; *PL_Tree* 表示孩子结点继承父结点得到的新树, 初始值是以起点 v_s 为根结点的树 *M-Tree*; *PL_W* 初始值为 0, 表示序列 *PL* 的权值. $OpenL_{min}$ 中, *P* 表示序列 *PL* 中权值 *W* 最小的序列, *Tree* 表示序列 *P* 对应的需要继承全遍历的树, *W* 表示序列 *P* 的权值. $OpenL_{min}$ 中, 初始化 *OpenL* 中的值(第 1 行、第 2 行). 当 $OpenL_{min}$ 中的 *Tree* 不为空时, 此时 *P* 未到达终点 v_e, 需要对 *P* 继续扩展(第 3 行). 若序列 *P* 减去起点 v_s 剩余结点的个数等于类别 *I* 的个数, 表示此时 *P* 已经扩展到终点 v_e 的前一个结点, 此时 *Tree* 设置为空, 将序列 *P* 添加终点 v_e 并更新 *P* 的权值, 然后将得到的 *P*, *Tree*, *W* 添加到 *OpenL* 中(第 4 行–第 8 行). 若序列 *P* 减去起点 v_s 剩余结点的个数不等于类别 *I* 的个数, 需要对 *Tree* 进行继承全遍历操作(第 10 行–第 19 行). 由于本文研究同类别的 *I* 中不止一个元素, 因此需要将 *I* 中所有元素进行遍历, 遍历到类别中的结点 *v* 时, 将序列 *P* 扩展到结点 *v* 并更新权值 *W*, 最后将(*P*,*Tree*,*W*)添加到 *OpenL* 中(第 13 行–第 16 行). 完成一次树的继承全遍历后, 更新 *OpenL*(第 19 行). 将 $OpenL_{min}$ 更新为 *OpenL* 中权值最小的序列(第 21 行), 当 $OpenL_{min}$ 的路径 *P* 到达 v_e 时, 则查询完毕. 输出结果中, $OpenL_{min}$ 中的路径 *P* 为图 G_{max} 中最短路径, 权值为 $OpenL_{min}$ 中的 *W*.

例 4: 将例 1 中的动态网络构建为最大权值网络 G_{max}(如图 7 所示), 利用 Tree_IT_SP 算法求解 G_{max} 中最短路径 *p*. 过程见表 2, 表中 *OpenL* 加粗的元素即是每次查询得到的最短子路径. 初始化 $OpenL$=$\{(v_s,Tree,0)\}$; $OpenL_{min}$=$(v_s,Tree,0)$, 得到最短路径为$(v_s,v_1,v_5,v_2,v_4,v_e)$, 该路径的权值为 110, 即在动态网络, 最短路径的权值上限为 110.

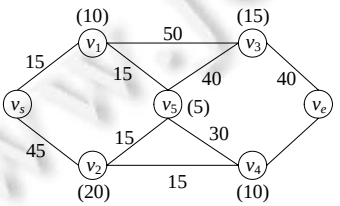


图 7 最大权值网络 G_{max}

若直接采用穷举验证法, 得到的 8 个序列中(见例 2), 通过对同类别的地点集合 I_i 中的多个元素进行遍历并加上终点 v_e 后, 得到 16 条路径构成的集合 *PM*:$\{(v_s,v_2,v_1,v_3,v_5,v_e),(v_s,v_2,v_1,v_5,v_3,v_e),(v_s,v_1,v_3,v_5,v_2,v_e),(v_s,v_1,v_3,v_2,v_5,v_e),(v_s,v_1,v_5,v_2,v_3,v_e),(v_s,v_1,v_5,v_3,v_2,v_e),(v_s,v_1,v_2,v_5,v_3,v_e),(v_s,v_1,v_2,v_3,v_5,v_e),(v_s,v_2,v_1,v_4,v_5,v_e),(v_s,v_2,v_1,v_5,v_4,v_e),(v_s,v_1,v_4,v_5,v_2,v_e),(v_s,v_1,v_4,v_2,v_5,v_e),(v_s,v_1,v_5,v_2,v_4,v_e),(v_s,v_1,v_5,v_4,v_2,v_e),(v_s,v_1,v_2,v_5,v_4,v_e),(v_s,v_1,v_2,v_4,v_5,v_e)\}$. 若运用穷举验证法, 需要遍历这 16 条路径中的 90 个子路径, 计算量大, 而且对无用的子路径扩展较多. 通过表 2 可知: 运用

Tree_IT_SP 算法, 直观上仅需对 14 个路径的 49 个子路径进行查询计算即可, 远低于穷举验证法需要验证的 16 个序列中的 90 个子路径; 此外, 在 Tree_IT_SP 算法实际计算中, 路径权值是在子路径权值的基础上进行叠加, 计算过的子路径是无需重复计算, 其中需要计算的子路径只有 13 个.

表 2 Tree_IT_SP 算法求解最短路径过程

Number	*Open_L*
1	(({v_S,v_1},25),({v_S,v_2},65))
2	(({v_S,v_1,v_5},45),({v_S,v_2},65),({v_S,v_1,v_2},75),({v_S,v_1,v_4},80),({v_S,v_1,v_3},90))
3	(({v_S,v_2},65),({v_S,v_1,v_2},75),({v_S,v_1,v_4},80),({v_S,v_1,v_5,v_2},80),({v_S,v_1,v_5,v_4},85),({v_S,v_1,v_3},90),({v_S,v_1,v_5,v_3},100))
4	(({v_S,v_1,v_2},75),({v_S,v_1,v_4},80),({v_S,v_1,v_5,v_2},80),({v_S,v_1,v_5,v_4},85),({v_S,v_1,v_3},90),({v_S,v_1,v_5,v_3},100),({v_S,v_2,v_1},105))
5	(({v_S,v_1,v_4},80),({v_S,v_1,v_5,v_2},80),({v_S,v_1,v_5,v_4},85),({v_S,v_1,v_3},90),({v_S,v_1,v_2,v_5},95), ({v_S,v_1,v_2,v_4},100),({v_S,v_1,v_5,v_3},100),({v_S,v_2,v_1},105),({v_S,v_1,v_2,v_3},145))
6	(({v_S,v_1,v_5,v_2},80),({v_S,v_1,v_5,v_4},85),({v_S,v_1,v_3},90),({v_S,v_1,v_2,v_5},95),({v_S,v_1,v_2,v_4},100), ({v_S,v_1,v_5,v_3},100),({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_2,v_3},145))
7	(({v_S,v_1,v_5,v_4},85),({v_S,v_1,v_3},90),({v_S,v_1,v_2,v_5},95),({v_S,v_1,v_2,v_4},100),({v_S,v_1,v_5,v_3},100),({v_S,v_1,v_5,v_2,v_4},105), ({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150))
8	(({v_S,v_1,v_3},90),({v_S,v_1,v_2,v_5},95),({v_S,v_1,v_2,v_4},100),({v_S,v_1,v_5,v_3},100),({v_S,v_1,v_5,v_2,v_4},105),({v_S,v_2,v_1},105), ({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150))
9	(({v_S,v_1,v_2,v_5},95),({v_S,v_1,v_2,v_4},100),({v_S,v_1,v_5,v_3},100),({v_S,v_1,v_5,v_2,v_4},105),({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115), ({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150),({v_S,v_1,v_3,v_2},165))
10	(({v_S,v_1,v_2,v_4},100),({v_S,v_1,v_5,v_3},100),({v_S,v_1,v_5,v_2,v_4},105),({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115), ({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_1,v_2,v_5,v_e},130),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150),({v_S,v_1,v_3,v_2},165))
11	(({v_S,v_1,v_5,v_3},100),({v_S,v_1,v_5,v_2,v_4},105),({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120), ({v_S,v_1,v_2,v_5,v_e},130),({v_S,v_1,v_2,v_4,v_5},135),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150),({v_S,v_1,v_3,v_2},165))
12	(({v_S,v_1,v_5,v_2,v_4},105),({v_S,v_2,v_1},105),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_1,v_2,v_5,v_e},130), ({v_S,v_1,v_2,v_4,v_5},135),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150),({v_S,v_1,v_3,v_2},165),({v_S,v_1,v_5,v_3,v_2},175))
13	(({v_S,v_2,v_1},105),({v_S,v_1,v_5,v_2,v_4,v_e},110),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_1,v_2,v_5,v_e},130), ({v_S,v_1,v_2,v_4,v_5},135),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150),({v_S,v_1,v_3,v_2},165),({v_S,v_1,v_5,v_3,v_2},175))
14	(({v_S,v_1,v_5,v_2,v_4,v_e},110),({v_S,v_1,v_4,v_2},115),({v_S,v_1,v_4,v_5},115),({v_S,v_1,v_5,v_4,v_2},120),({v_S,v_2,v_1,v_5},125), ({v_S,v_1,v_2,v_5,v_e},130),({v_S,v_1,v_2,v_4,v_5},135),({v_S,v_1,v_3,v_5},135),({v_S,v_1,v_2,v_3},145),({v_S,v_1,v_5,v_2,v_3},150), ({v_S,v_2,v_1,v_4},160),({v_S,v_1,v_3,v_2},165),({v_S,v_2,v_1,v_3},170),({v_S,v_1,v_5,v_3,v_2},175))

4.2.2 剪枝算法

在动态网络中, 边的权值会随着时间而变化, 因此在计算子路径的权值时, 不能像静态网络中对计算过的子路径进行重复使用. 而且到达某个结点的时间无法确定, 只能从路径的起点向后扩展[22,23]. 运用 Tree_IT_SP 算法可以在动态网络中进行最短路径求解, 由于需要扩展的查询路径较多, 而且动态网络中的子路径权值不能复用, 导致效率低下. 因此提出了 Tree_IT_P 算法对路径集合 *PM* 进行剪枝, 使得候选路径集合 $PM_{\min}$ 更小.

Tree_IT_P 算法如算法 3 所示, 其中, 阈值 $W_{os}\_G_{\max}$ 为 Tree_IT_SP 算法求得满足 *M* 的所有路径集合 *PM* 的权值上限. Tree_IT_P 算法类似于 Tree_IT_SP 算法, 所不同的是, 需要判断查询到的子序列的权值 *W* 是否大于 $W_{os}\_G_{\max}$: 若 *W* 大于 $W_{os}\_G_{\max}$, 此序列不再向后扩展; 若 *W* 小于 $W_{os}\_G_{\max}$, 则将此时的 *P*, *Tree* 和 *W* 添加到 *OpenL* 中(第 12 行、第 13 行). 通过公式(1)可知, 最小路径 P_{mt} 在静态网络 $G_{\min}$ 中的权值应小于 $W_{os}\_G_{\max}$, 因此可在静态网络 $G_{\min}$ 中对路径集合 *PM* 进行剪枝, 即对路径的权值 *W* 与 $W_{os}\_G_{\max}$ 进行大小比较, 将 *W* 小于 $W_{os}\_G_{\max}$ 的路径 *P*、空树 *Tree* 和 *W* 添加到 *OpenL* 中, 进而剪枝掉权值大于 $W_{os}\_G_{\max}$ 的路径(第 21 行、第 22 行); 最后输出 *OpenL* 中所有 *P* 序列即候选序列集合 $PM_{\min}$.

算法 3. *Tree_IT_P*($G_{\min}$,*M-Tree*,v_s,v_e,$W_{os}\_G_{\max}$).

输入: 图 $G_{\min}$, 广义规则树 *M-Tree*, 起点 v_s, 终点 v_e, 阈值 $W_{os}\_G_{\max}$;

输出: 剪枝后的路径集 $OpenL.PM_{\min}$.

1. $OpenL=\{PM_{\min},Tree,W\}$;
2. $OpenL\leftarrow\{(v_s,M\text{-}Tree,0)\}$, $U\leftarrow W_{os}\_G_{\max}$;
3. **for each** $Op\in OpenL$ **do**
4. **if** $Op.Tree\neq 0$ **then**
5. **if** $long(OpenL.PM_{\min}-v_s)\neq long(M.I)$ **then**

```
root←Op.Tree←parent;
for each I∈root←child do
  Tree←I⊕root;
  for each v∈I do
    P←(Op.PM_min+v);
    W←Op.W+W(G_min,P);
      if W<U then
        OpenL←OpenL+(P,Tree,W);
      end if
    end for
  end for
else
  Tree←0;
  P←(OpenL.PM_min+v_e);
  W←(OpenL.W+W(G_min,P));
  if W<U then
    OpenL←OpenL+(P,Tree,W);
  end if
end if
OpenL←(OpenL−Op);
end if
end for
return OpenL.PM_min;
```

例 5: 在例 4 中, 利用 Tree_IT_SP 算法求解 $G_{\max}$ 得到的权值上限 $W_{os}\_G_{\max}$ 为 110, 将 $W_{os}\_G_{\max}$=110 作为阈值, 对 $G_{\min}$ 运用 Tree_IT_P 算法处理后, 将原路径集合 PM 中的 16 个序列剪枝, 得到包含 8 条路径的候选路径集合: $PM_{\min}$={(v_s,v_1,v_4,v_5,v_2,v_e),(v_s,v_1,v_4,v_2,v_5,v_e),(v_s,v_1,v_2,v_4,v_5,v_e),(v_s,v_1,v_2,v_5,v_4,v_e),(v_s,v_1,v_5,v_4,v_2,v_e),(v_s,v_1,v_5,v_2,v_4,v_e),(v_s,v_2,v_1,v_4,v_5,v_e),(v_s,v_2,v_1,v_5,v_4,v_e)}.

通过例 5 的结果可以得出: 运用 Tree_IT_P 算法进行剪枝后, 得到的候选路径集合是原来的二分之一, 体现出 Tree_IT_P 算法的高效性.

4.3 最短路径查询

本节主要用 TDRN_SP 算法(算法 4)对候选路径集合 $PM_{\min}$ 进行逐个查询, 以得到最短路径 P_{mt}. 算法思想: 首先为 $W(P)$建立一个动态阈值 U, 作为最短路径的权值上限, 所有子路径均在阈值 U 下进行扩展. 阈值 U 初始值为 $W_{os}\_G_{\max}$. 在 t 时刻, 对候选路径集合 $PM_{\min}$ 中的某条路进行验证查询时, 得到一条子路径 p'={v_s,v_1,...,v_i}的权值 $W(G,p',t)$不小于 U 且 v_i 不是终点 v_e 时, 则停止此结点向后扩展查询. 若 $W(G,p',t)$小于 U 且 v_i=v_e(即此时已扩展一条完整路径), 则更新 U 为 $W(G,p',t)$, 以此收紧子路径的查询. 最后得到的 U 即为所有候选路径中的最小权值.

算法 4. $TDRN\_SP(G,t,OpenL_{\min}.P,OpenL_{\min}.W,OpenL.PM_{\min})$.

输入: 动态图 G, 出发时间 t, 算法 Tree_IT_SP 在图 $G_{\max}$ 中得到的最短路径 $OpenL_{\min}.P$ 及其权值 $OpenL_{\min}.W$, 算法 Tree_IT_P 在图 $G_{\min}$ 中得到的候选路径集合 $OpenL.PM_{\min}$;

输出: 最短路径 P_{mt}, 路径 P_{mt} 的权值 $W(G,P_{mt},t)$.

```
1. P_mt←OpenL_min.P, U←OpenL_min.W, PM_min←OpenL.PM_min;
2. for each p∈PM_min do
```

3. $p'\leftarrow\{v_s\}$, $W\leftarrow 0$;
4. **for each** $v_i\in(p.v-p.v_s)$ **do**
5. $p'\leftarrow(p'+v_i)$;
6. $W\leftarrow W(G,p',t)$
7. **if** $W>U$ **then**
8. **break**;
9. **else**
10. **if** $v_i=v_e$ **then**
11. $P_{mt}\leftarrow p$;
12. $U\leftarrow W$;
13. **end if**
14. **end if**
15. **end for**
16. **end for**
17. **return** P_{mt}, U;

在算法输入部分, 通过 Tree_IT_SP 算法输出的 $OpenL_{\min}.W$ 作为最短路径权值上限 W_{os}_$G_{\max}$, Tree_IT_SP 算法输出的路径 $OpenL_{\min}.P$ 作为 TDRN_SP 算法的初始最短路径, 算法 Tree_IT_P 输出的集合 $OpenL.PM_{\min}$ 即为候选路径集合 $PM_{\min}$; 首先, 将 U, P_{mt}, $PM_{\min}$ 初始化(第 1 行). 候选路径集合 $PM_{\min}$ 中, 每条路径在动态网络中进行遍历查询即可查询出最短路径. 在遍历时, 定义了两个参数 p'和 w(第 3 行): 参数 p'表示当前扩展到的子路径, 初始化为(v_s); 参数 W 表示 p'的权值, 默认为 0. 若扩展到 v_i 时, 将子路径 p'的权值赋值给 W(第 6 行). 若当前路径的权值 W 已经大于 U, 则停止后续路径扩展(第 7 行、第 8 行); 若 W 小于 U 时, 判断 v_i 是否是最后的目的结点 v_e, 若是 v_e, 则更新 P_m 和 U(第 11 行、第 12 行). 更新最短路径和阈值后, 继续对候选路径集合 $PM_{\min}$ 进行遍历, 最终输出 P_{mt} 和 U, 即为最短路径 P_{mt} 及其权值 $W(G,P_{mt},t)$.

例 6: 运用 TDRN_SP 算法对例 5 求得的候选路径集合 $PM_{\min}$ 进行验证查询. 通过例 4 中利用 Tree_IT_SP 算法求解 $G_{\max}$ 得出 P_{mt} 初始值为$(v_s,v_1,v_5,v_2,v_4,v_e)$, 最短路径权值上限, 即阈值 U 为 110. 出发时刻 t=0, 运用 TDRN_SP 算法得到最短路径 $P_{mt}=(v_s,v_1,v_5,v_2,v_4,v_e)$, 权值 $W(G,P_{mt},t)$=83. 在遍历过程中, 其中的 5 个序列$(v_s,v_1,v_4,v_5,v_2,v_e)$, $(v_s,v_1,v_4,v_2,v_5,v_e)$, $(v_s,v_1,v_2,v_4,v_5,v_e)$, $(v_s,v_2,v_1,v_4,v_5,v_e)$, $(v_s,v_2,v_1,v_5,v_4,v_e)$的权值均大于阈值, 因为不必对这 5 条路径的全部结点进行扩展, 从而减少了计算时间.

TDF_SP 框架在查询一条基于多规则 M 的最短路径 P_{mt} 的过程中, 需要在 3 个网络中进行 3 次查询, 但前两次扩展均是在静态网络中查询, 而且实验表明, 前两次的查询过程相比穷举验证法和先查询后验证法在效率上有很大的提高. 在实际的应用场景中, 例如在道路网络中查询基于多规则的最短路径时, 由于存在各种突发情况, 用户几乎无法在查询到的最短路径中按时到达中途的地点, 因此需要对最短路径进行重新查询. TDF_SP 框架在对路径进行重新规划时, 不需要再次在两个静态网络 $G_{\max}$ 和 $G_{\min}$ 中查询, 只需在动态网络 G 中进行查询, 所以不必调用 Tree_IT_SP 和 Tree_IT_P 算法, 只需做减法计算即可调用 TDRN_SP 算法对路径进行重新查询. 假定在 t 时刻, 需要在 v_i 点重新查询最短路线.

(1) 将首次最短路径查询得到的路径 P_{mt} 中 v_i 之前的结点全部删除, 得到以 v_i 为起点、v_e 为终点的新路径 Pk, 路径 Pk 在 $G_{\max}$ 中的最小权值通过减法计算可以得到: $W(G_{\max},Pk)=W(G_{\max},P_m)-W(G_{\max},v_s\rightarrow v_i)$;

(2) 将候选路径集合 $PM_{\min}$ 中的所有路径保留子路径 $v_i\rightarrow v_e$, 构成了新的候选路径集合 $PK_{\min}$;

(3) 将动态网络 G、当前时间 t'、Pk、$W(G_{\max},Pk)$、$PK_{\min}$ 输入 TDRN_SP 算法, 即可重新查询最短路径.

4.4 时间复杂度分析

在 Tree_IT_SP 和 Tree_IT_P 算法中, 时间复杂度主要集中在对树的继承全遍历操作中, 树中的每个结点需要访问 n 次, 因此树的继承全遍历操作时间复杂度为 $O(n^2)$, 每个类别 I 中又包含了 k 个地点, 因此 Tree_IT_

SP 和 Tree_IT_P 算法的时间复杂度均为 $O(n^2 \cdot k)$. 在 TDRN_SP 算法中只需要对所有路径进行一次遍历即可, 因此其时间复杂度为 $O(n)$.

5 实 验

5.1 实验设置

本文使用 2 个真实的道路网络进行实验, 美国旧金山湾区(San Francisco Bay Area, SFBA)和德国的奥尔登堡(Oldenburg, OLD)真实道路网络数据(见表 3): SFBA 道路网络包含 175 345 个结点和 23 606 条边, OLD 道路网络包含 6 105 个结点和 7 305 条边. 根据路网每天变化规律对路网进行赋值: 将全天分为 6 个时间段, [7:00,9:00], [9:00,14:00], [14:00,17:00], [17:00,21:00], [21:00,7:00]; 每隔 5 分钟对每条边和每个地点随机生成一个 10–60 之间的值, 以此表示通过某条边的时间和在某个地点的耗费时间, 根据每条边和每个地点的值拟合为分段函数. 实验语言采用 Java 语言, 实验平台为 Windows 10, 64 位操作系统, 内存 8 GB, 处理器 Intel Corei5-3230 CPU@2.60 GHz.

表 3 数据集

路网名称	简称	结点数量	边的数量
San Francisco Bay Area	SFBA	175 343	23 606
Oldenburg	OLD	6 105	7 035

本文通过生成 15 个数据集 $D1$–$D15$ 对提出的算法在不同规模的数据集上进行对比实验, 出发时间 t 随机在 24 小时内产生, 实验结果取每个数据集重复实验 10 次的平均值. 在现实情况中, 不同的类别组成的集合 I 一般不超过 10 个元素, 每个类别包含的地点一般较少, 因此设置的每个类别 I_i 中的地点个数不超过 5 个. $D1$–$D5$ 设置不同规模的集合 I, I 中类别个数依次是 6–10 个, 每个类别 I_i 中的地点个数均取 5 个, Z 中的顺序关系个数均设置为 5. $D6$–$D10$ 设置不同规模的顺序关系 Z, 顺序关系个数依为 2, 4, 6, 8, 10, 类别集合 I 均设置为 5, 每个类别 I_i 中的个数均取 5 个. $D11$–$D15$ 对每个类别 I_i 中的地点个数设置了不同规模, 个数依次是 2, 4, 6, 8, 10, 集合 I 均设置为包含 5 个 I_i, 顺序关系 Z 均设置为 5. 以上各组数据中的起点 v_s 和 v_e 均在地图中随机产生, 类别集合 I 和 I_i 中的地点及顺序关系 Z 均在地图中随机产生满足定义 2.3 的数据.

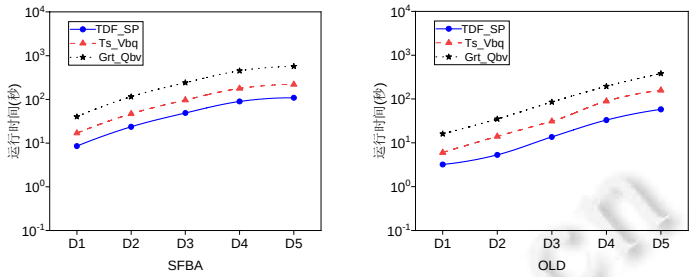
实时规划最短路径的对比实验中, 在 3 种算法均查询出最短路径后开始计时. 为了避免误差, 每次实验的 3 种算法实时规划的起点和出发的时间均相同, 起点是在已查询出的最短路径中随机选取的一个不为起点和终点的结点, 在 24 小时内随机产生出发时间 t'.

由于目前解决多规则的问题中可分类两类算法: 先验证后查询和先查询后验证, 对比算法采用目前处理同类问题中效率较好的广义规则树验证方法[7]和拓扑排序法[15]进行对比, 拓扑排序法和 TDF_SP 均属于的先验证后查询的处理方式, 广义规则树则属于先查询后验证的方式. 虽然广义规则树验证方法和拓扑排序法均能解决多规则的最短路径查询问题, 但文献[7,15]中提出的算法均适用于静态网络环境, 因此将广义规则树验证方法和拓扑排序法改进为 Grt_Qbv(generalized rule tree_query before verify)算法和 Ts_Vbq (topological sorte_verify before query)算法, 以适用于动态网络环境. Grt_Qbv 算法的验证部分采用了文献[7]中利用广义规则树判断路径是否满足多规则的方法, Ts_Vbq 算法的验证部分采用了文献[15]中通过拓扑排序图生成路径集合的方法. 文献[7,15]这些验证方法均不区分动态路网或静态路网, 只是用来对多规则验证, 不涉及动态路网或静态网络中路径的查询.

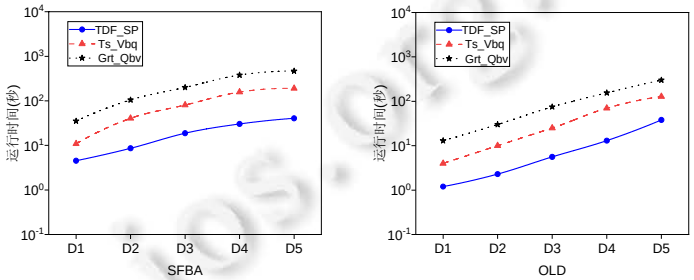
由于 TDF_SP 查询框架在静态网络和动态网络均需要最短路径查询, 因此为了体现算法对比的公平性, 在 TDF_SP, Grt_Qbv 和 Ts_Vbq 对两点间的最短路径查询部分, 采用 SHARC (shortcuts+arc-flags)算法, SHARC 在静态网络[24]和动态网络[25]的最短路径查询均具有高效性.

5.2 结果分析

在道路网络 SFBA 和 OLD 中, 对 3 种算法使用数据集 $D1$–$D5$ 的实验结果如图 8 和图 9 所示.

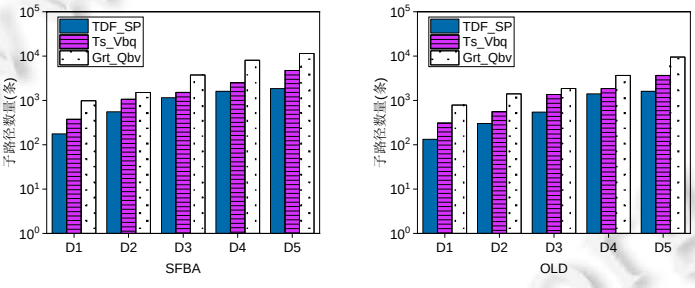


(a)

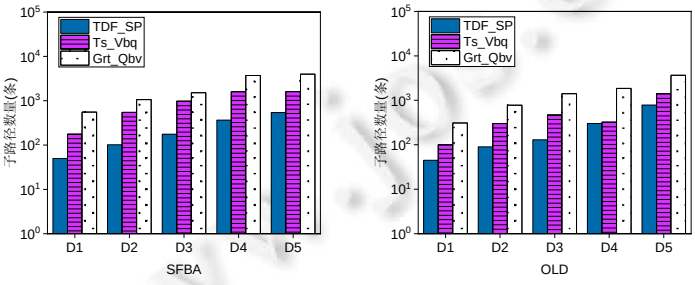


(b)

图 8 *D*1–*D*5 的查询时间



(a)



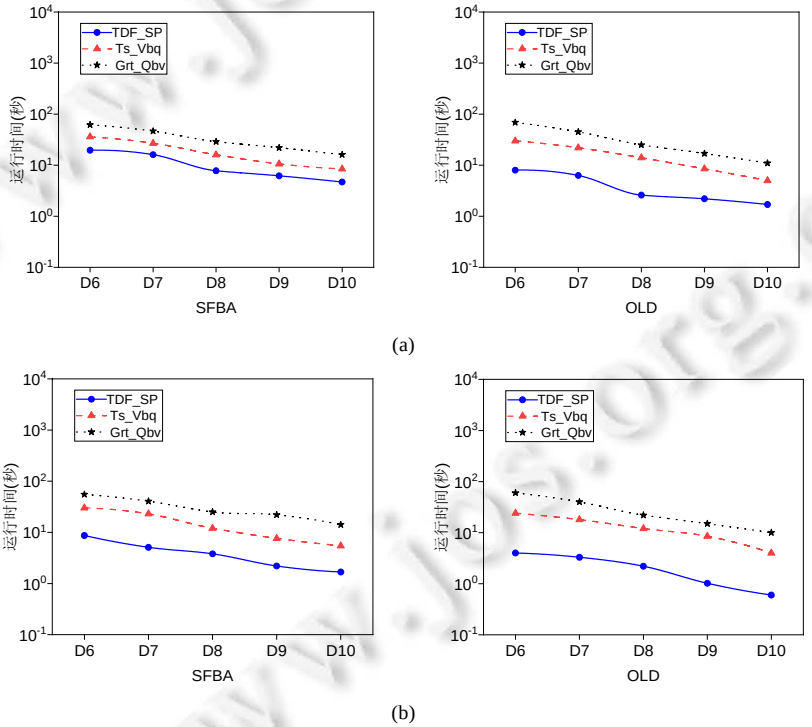
(b)

图 9 *D*1–*D*5 的子路径数量

图 8 为 3 种算法的查询效率, 其中, 图 8(a)表示首次查询耗费的时间, 图 8(b)表示重新规划耗费的时间. 可以看出: 在对 *D*1–*D*5 的 5 个数据集的查询中, 随着类别集合 *I* 的增大, 所需要的查询时间逐渐增加. TDF_SP 在每个数据集的首次查询时间均明显小于 Grt_Qbv 和 Ts_Vbq 算法. 在图 8(b)重新规划的实验结果中, TDF_SP 计算时间大幅度减少, 而 Ts_Vbq 算法和 Grt_Qbv 算法变化较小. 此外, TDF_SP 与 Ts_Vbq 算法在实时规划时比首次查询时差距变大. 这是因为 TDF_SP 在首次查询得到的阈值和候选路径集合经过加减计算即可再次使用, 而 Ts_Vbq 算法和 Grt_Qbv 算法在重新查询时不能复用首次查询得到的任何结果, 因此效率较低.

图 9 为查询的子路径的数量. 从图 9(a)中可以看出, TDF_SP 和 Ts_Vbq 算法查询的子路径数量相差不大. 这是由于二者均是根据类别进行的查询, 比 Grt_Qbv 算法通过结点为单位进行扩展子路径更少. TDF_SP 比 Ts_Vbq 算法扩展子路径更少, 是因为在查询下一个类别时, 可以通过树的继承全遍历得到符合顺序关系的类别, 而 Ts_Vbq 算法需要对未访问的所有类别进行遍历验证前置计数是否为零; 再者, Ts_Vbq 算法需要对符合多规则 M 路径集合在动态网络中全部验证查询, 导致扩展子路径较多. 图 9(b)为重新实时规划时, 各算法需要查询的子路径数量. 可以看出, TDF_SP 中子路径数量减少了约 2/3. 这是因为在重新实时规划时, 只需要查询剪枝得到的候选集合中的路径即可, 不需要对剪枝部分的子路径进行扩展查询.

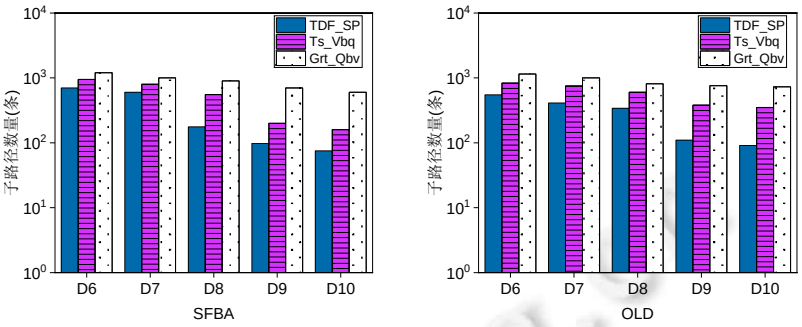
数据集 $D6$–$D10$, 是保持类别 I 和 I 中地点数量不变的情况下, 逐渐增大顺序关系 Z. 图 10 和图 11 分别是 3 种算法在此数据集上的查询时间和子路径数量的实验结果. 随着 $D6$–$D10$ 中顺序关系的增多, 需要查询的子路径减少, 查询时间也会随之减少, 在图 10 和图 11 中均有明显的较少趋势. 在重新实时最短路径查询时(如图 10(b)所示), TDF_SP 查询时间显著减少, 均能在 10 s 以下得出结果; 而 Ts_Vbq 算法和 Grt_Qbv 算法的运行时间并没有显著降低. 在图 11(a)中, Grt_Qbv 算查询的子路径并没有随着顺序关系增多而明显降低. 主要是因为每次扩展一个新的结点构成一条新的子路径, 均要通过广义规则树判断是否符合顺序关系: 若符合, 则继续查询;反之, 不再查询此结点所在的类别中的其他结点. TDF_SP 和 Ts_Vbq 算法根据顺序关系进行子路径扩展, 在顺序关系增大时, 查询的子路径数量会显著减少.



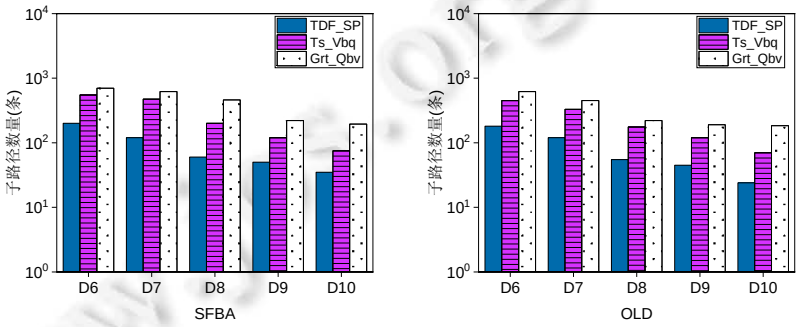
(a)

(b)

图 10 $D6$–$D10$ 的查询时间



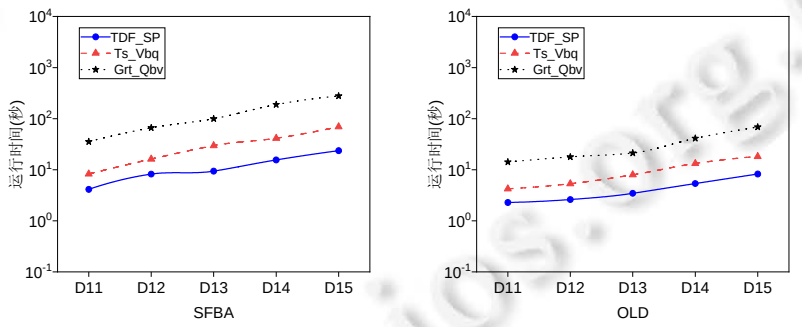
(a)



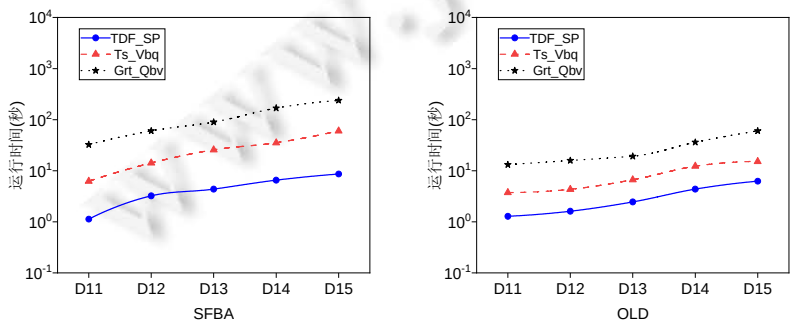
(b)

图 11 *D*6–*D*10 的子路径数量

图 12 和图 13 是在对每个类别中地点个数发生变化的数据集 *D*11–*D*15 的实验结果.



(a)



(b)

图 12 *D*11–*D*15 的查询时间

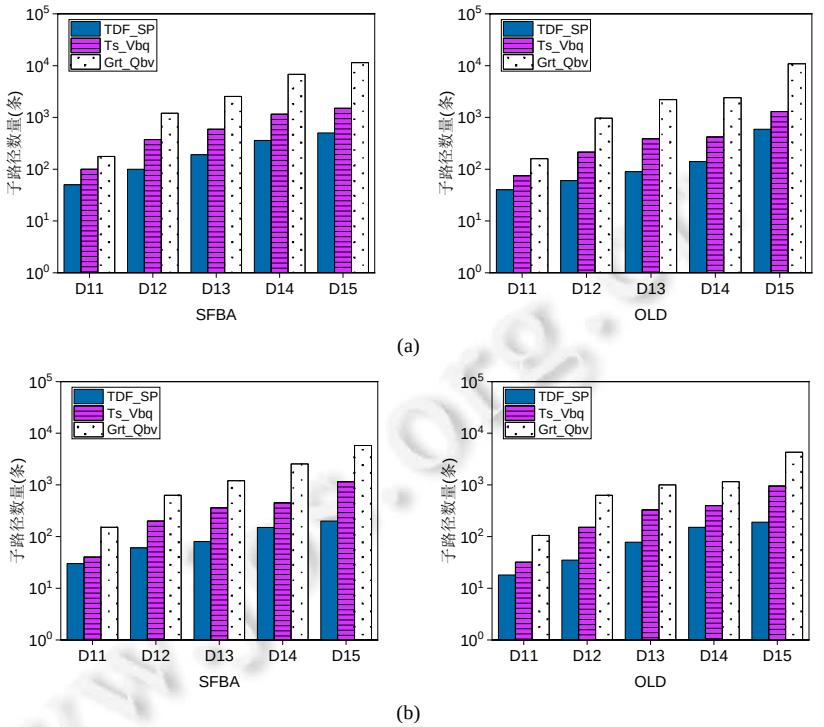


图 13 *D*11–*D*15 的子路径数量

在首次最短路径实验中(如图 12(a)所示), Grt_Qbv 算法查询时间与 Ts_Vbq 算法查询时间的差值远大于 Ts_Vbq 算法查询时间与 TDF_SP 查询时间的差值. 可以看出, Grt_Qbv 算法由于验证多规则耗费较多时间. 在重新实时路径查询实验中(如图 12(b)所示), Ts_Vbq 算法查询时间与 TDF_SP 查询时间的差值增大, 表明 TDF_SP 框架在重新实时路径查询中的高效性. 另外, 通过图 13 的实验表示: 随着每个类别中的地点数量增多, 需要查询的子路径变多, 因此查询时间增加.

从对数据集 *D*11–*D*15 的查询结果来看: TDF_SP 查询效率更高, 算法中的 Tree_IT 算法思想极大地提高了解决此类问题的效率; Tree_IT_P 算法高效的剪枝技术避免了不必要的扩展, 进一步提高了算法的效率. 在重新规划的实验中, 由于 TDF_SP 查询框架能够对首次查询结果进行复用, *D*1–*D*15 数据集中查询的子路径数量平均比首次查询的子路径数量减少了大约 70%, 另外两个对比算法比首次查询的子路径数量减少大约 40%.

6 总 结

本文设计了一种多规则最短路径查询框架: TDF_SP 查询框架, TDF_SP 由路径集合查询和最短路径查询两部分构成: 在路径集合查询部分, 提出了一种新颖的树遍历方法, 即树的继承全遍历, 并基于此方法提出了 Tree_IT_SP 和 Tree_IT_P 这两个算法, 用于求取最短路径的上限值和候选路径集合; 在最短路径查询部分, 提出了 TDRN_SP 对候选路径集合中的路径进行最短路径搜索. 实验表明: 动态网络中, 本文提出的多规则的最短路径算法优于目前已有的算法. 本文提出的框架虽然能够解决动态网络中多规则最短路径查询问题, 但若每个类别中地点数量较多时, 查询需要耗费更多的时间. 在未来的工作中, 我们将对类别中地点数量进行剪枝, 从而缩小查询的地点集合, 进而使查询框架更加高效. 此外, 由于动态路网中边的权值最大值集中在交通繁忙时段, 在交通非繁忙时段进行最短路径规划时, 会导致剪枝效果不明显. 在未来的工作中, 我们将对动态路网的权值最大值根据查询时间分段处理, 以提高剪枝效率, 缩小候选路径集合.

References:

[1] Ahmadi E, Nascimento MA. A mixed breadth-depth first search strategy for sequenced group trip planning queries. In: Proc. of the 16th IEEE Int'l Conf. on Mobile Data Management. IEEE, 2015. 24−33. [doi: 10.1109/MDM.2015.49]

[2] Liu H, Jin C, Yang B, *et al*. Finding top-*k* optimal sequenced routes. In: Proc. of the 34th Int'l Conf. on Data Engineering (ICDE). IEEE, 2018. 569−580.

[3] Liu H, Jin C, Zhou A. Popular route planning with travel cost estimation. In: Proc. of the Int'l Conf. on Database Systems for Advanced Applications. Springer Int'l Publishing, 2016. 403−418. [doi: 10.1007/978-3-319-32049-6_25]

[4] Zeng Y, Chen X, Cao X, *et al*. Optimal route search with the coverage of users' preferences. In: Proc. of the Int'l Conf. on Artificial Intelligence. AAAI, 2015. 2118−2124.

[5] Cai Y, Zhao Z, Yao L, *et al*. An algorithm for LBS-based schedule recommendation with time constraint. In: Proc. of the 12th Web Information System and Application Conf. (WISA). IEEE, 2015. 191−196. [doi: 10.1109/WISA.2015.17]

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

[7] 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 with English abstract). http://www.jos.org.cn/1000-9825/5692.htm [doi: 10.13328/j.cnki.jos.005692]

[8] Guan MG. Graphic programming using odd or even points. Chinese Mathematics, 1960, 10(3): 263−266 (in Chinese with English abstract).

[9] Guan MG. A historical review on the research and development. Operations Research Transactions, 2015, 19(3): 1−7. [doi: 10.15960/j.cnki.issn.1007-6093.2015.03.001]

[10] Ma X, Shekhar S, Xiong H, *et al*. Exploiting a page-level upper bound for multi-type nearest neighbor queries. In: Proc. of the 14th Annual ACM Int'l Symp. on Advances in Geographic Information Systems. 2006. 179−186.

[11] Rice MN, Tsotras VJ. Engineering generalized shortest path queries. In: Proc. of the 29th IEEE Int'l Conf. on Data Engineering (ICDE). IEEE, 2013. 949−960.

[12] Costa CF, Nascimento MA, Macêdo JAF, *et al*. Optimal time-dependent sequenced route queries in road networks. In: Proc. of the 23rd SIGSPATIAL Int'l Conf. on Advances in Geographic Information Systems. 2015. 1−4.

[13] Ahmadi MH, Haghighatdoost V. General time-dependent sequenced route queries in road networks. IEEE Computer Society, 2017. 949−956. [doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.158]

[14] Li J, Yang YD, Mamoulis N. Optimal route queries with arbitrary order constraints. IEEE Trans. on Knowledge and Data Engineering, 2013, 25(5): 1097−1110. [doi: 10.1109/TKDE.2012.36]

[15] 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. 2008. 1−10.

[16] Li M, Li X, Yin J. TORD problem and its solution based on big trajectories data. IEEE Trans. on Intelligent Transportation Systems, 2015, 17(6): 1−12. [doi: 10.1109/TITS.2015.2491269]

[17] Sun N, Shi H, Han G, *et al*. Dynamic path planning algorithms with load balancing based on data prediction for smart transportation systems. IEEE Access, 2020. 15907−15922. [doi: 10.1109/ACCESS.2020.2966995]

[18] Kwon OS, Cho HJ. Design and implementation of real-time shortest path search system in directed and dynamic roads. Journal of Korea Multimedia Society, 2017, 20(4): 649−659. [doi: 10.9717/kmms.2017.20.4.649]

[19] Sun Y, Yu X, Bie R, *et al*. Discovering time-dependent shortest path on traffic graph for drivers towards green driving. Journal of Network and Computer Applications, 2017, 83: 204−212. [doi: 10.1016/j.jnca.2015.10.018]

[20] Zhang D, Yang D, Wang Y, *et al*. Distributed shortest path query processing on dynamic road networks. The VLDB Journal, 2017, 26(3): 399−419. [doi: 10.1007/s00778-017-0457-6]

[21] Tan GZ, Gao W. Shortest path algorithm in time-dependent networks. Chinese Journal of Computers, 2002, 25(2): 165−172 (in Chinese with English abstract).

[22] Hartley J, Alhoula W. Fast replanning incremental shortest path algorithm for dynamic transportation networks. In: Proc. of the Computing Conf. on Intelligent Computing. Cham: Springer, 2019. 22−43. [doi: 10.1007/978-3-030-22871-2_3]

[23] Chen BY, Chen XW, Chen HP, *et al*. Efficient algorithm for finding *k* shortest paths based on re-optimization technique. Transportation Research Part E: Logistics and Transportation Review, 2019, 101819. [doi: 10.1016/j.tre.2019.11.013]

[24] Bauer R, Delling D. SHARC: Fast and robust unidirectional routing. Journal of Experimental Algorithmics (JEA), 2010, 14: 2.4−2.29.

[25] Delling D. Time-Dependent SHARC-routing. In: Proc. of the European Symp. on algorithms. Berlin, Heidelberg: Springer, 2008. 332−343. [doi: 10.1007/s00453-009-9341-0]

附中文参考文献:

[7] 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 [doi: 10.13328/j.cnki.jos.005692]

[8] 管梅谷. 奇偶点图上作业法. 数学学报, 1960, 10(3): 263−266.

[9] 管梅谷. 关于中国邮递员问题研究和发展的历史回顾. 运筹学学报, 2015, 19(3): 1−7. [doi: 10.15960/j.cnki.issn.1007-6093.2015.03.001]

[21] 谭国真, 高文. 时间依赖的网络中最小时间路径算法. 计算机学报, 2002, 25(2): 165−172.

李艳红(1973－), 女, 博士, 教授, CCF 专业会员, 主要研究领域为空间和时空数据库, 数据可用性, 车联网技术.

罗昌银(1983－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为空间数据库, 位置查询, 大数据.

王猛(1989－), 男, 硕士, 主要研究领域为时空数据库.

杜小坤(1980－), 男, 博士, 讲师, CCF 会员, 主要研究领域为数据融合, 教育数据挖掘.

李国徽(1973－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为大数据, 人工智能, 实时计算.