软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(11):3469−3485 [doi: 10.13328/j.cnki.jos.005623]

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

公交网络下的一种费用限制最小时态路径查询索引*

马 慧[1], 汤 庸[2], 梁瑞仕[1]

[1](电子科技大学 中山学院 计算机学院,广东 中山 528400)

[2](华南师范大学 计算机学院,广东 广州 510631)

通讯作者: 汤庸, Email: ytang@m.scnu.edu.cn

摘 要: 私人交通网络下的最短路径查询主要考虑路径长度、行驶时间等因素,而公共交通网络下的路径查询需要考虑路径上相邻的边的时间顺序约束以及路径的费用.研究了公共交通网络下 3 种查询:给定起点、终点、时间区间和费用上限,查找在时间区间内不超过费用上限的最早到达路径、最晚出发路径和最短耗时路径.首先给出一种 Dijkstra 变种算法 Dijk-CCMTP,在此基础上给出 3 类查询的查询算法.然后提出一种高效的索引结构 ACCTL(approximate cost constrained time labelling).ACCTL 采用 Dijk-CCMTP 对图中的每个顶点预先计算部分从该顶点出发的和到达该顶点的基本路径.对于任意从起点 s 到终点 d 的查询,可以采用类似数据库表连接的方式从 ACCTL 中连接从 s 出发的和到达 d 的路径生成近似解,避免遍历原图搜索路径.ACCTL 建立索引的时间复杂度是 $O(|V|\cdot\Delta_{max}\cdot|E|\cdot(\log|E|+\Delta_{max}))$,其中,$|V|$表示顶点数,$|E|$表示边数,$\Delta_{max}$ 表示顶点的最大度数.实验验证 ACCTL 索引支持的查询速度比 Dijkstra 的变种算法的查询速度快 2~3 个数量级,并分析了影响建立索引时间和空间大小的因素.

关键词: 时间信息图;最小时态路径;费用限制;图索引;hub-labelling

中图法分类号: TP393

中文引用格式: 马慧,汤庸,梁瑞仕.公交网络下的一种费用限制最小时态路径查询索引.软件学报,2019,30(11):3469−3485. http://www.jos.org.cn/1000-9825/5623.htm

英文引用格式: Ma H, Tang Y, Liang RS. Indexing method for approximate cost constrained minimal temporal paths queries in public transportation networks. Ruan Jian Xue Bao/Journal of Software, 2019,30(11):3469−3485 (in Chinese). http://www.jos.org.cn/1000-9825/5623.htm

Indexing Method for Approximate Cost Constrained Minimal Temporal Paths Queries in Public Transportation Networks

MA Hui[1], TANG Yong[2], LIANG Rui-Shi[1]

[1](School of Computer Science, University of Electronic Science and Technology of China, Zhongshan Institute, Zhongshan 528402, China)

[2](School of Computer Science, South Normal University, Guangzhou 510631, China)

Abstract: In contrast to the paths queries under private transportation, which mainly concern the length or the driving time of a path, the paths queries under public transportation have to consider the time sequences between subsequent edges, as well as the total cost over a path. This study looks into three types of queries: given a source node, a destination node, a time interval, and a cost constraint, to find out a path within the given time interval which is restricted by the cost constraint, and has (i) the earliest arrival time, or (ii) the latest departure time, or (iii) the shortest duration. Firstly, a modified Dijkstra's algorithm called Dijk-CCMTP is presented based on derived algorithms respectively for the three types of queries above. After that, an effective indexing structure ACCTL (approximate cost

* 基金项目: 国家自然科学基金(U1811263, 61772211); 广东省高等学校优秀青年教师项目(YQ2015241, YQ2015242); 中山市科技计划(2015B2307)

Foundation item: National Natural Science Foundation of China (U1811263, 61772211); Foundation for Excellent Young Teachers in Higher Education of Guangdong Province (YQ2015241, YQ2015242); Science and Technology Plan of Zhongshan City (2015B2307)

收稿时间: 2017-11-21; 修改时间: 2018-04-16; 采用时间: 2018-07-01

constrained time labelling) is proposed. ACCTL invokes Dijk-CCMTP to precompute some canonical paths from (and, to) each node in the graph. For any query from source node s to destination node d, the approximate answer path can be obtain by matching those canonical paths that are from s (and, to d) in a database table join manner, so as to avoid traversing on the entire graph. The preprocessing time to build ACCTL is $O(|V|\cdot\Delta_{\max}\cdot|E|\cdot(\log|E|+\Delta_{\max}))$, where $|V|$ is the number of nodes, $|E|$ the number of edges, and $\Delta_{\max}$ the maximal degree among the nodes in the graph. Finally, the efficiency and effectiveness of ACCTL are confirmed. Experimental results show that the query algorithm under ACCTL is 2~3 orders of magnitude faster than the Dijkstra variant methods. The index preprocessing time and index size are also analyzed.

Key words: time information graph; minimal temporal path; cost constrained; graph index; hub-labelling

公共交通网络下的路径查询是地图服务的一个重要应用.私人交通网络下的路径查询通常用路径长度、行驶时间、费用等某方面的因素来衡量一条路径的长度;而公交网络查询需要考虑交通工具换乘的问题,因此需要考虑路径上的相邻边之间的时间顺序.此外,费用也是公交网络路径查询所要考虑的重要因素.例如,从广州直飞到洛杉矶虽然耗时短但价格昂贵,用户更想找到一条便宜的、可中途换乘的、旅途时间稍长的路线.本文研究这样的问题:一个旅行者计划乘坐公共交通工具出行,他的预算有限,他想知道在不超过预算的前提下:(1) 他计划在 t 时刻或之后出发,最早什么时候能到达?(2) 他计划在 t'时刻或之前到达,最晚需要什么时候出发?(3) 他可以在t时刻或之后出发,希望在t'时刻或之前到达,想找一条旅途时间最短的路径.上述 3 种查询分别称为带费用限制的最早到达路径查询、最晚出发路径查询和最短耗时路经查询.3 种查询的共同点是在费用限制的前提下查找最快的路径,因此下文统称为带费用限制的最小时态路径查询.

与本文工作最相关的是文献[1].Biswas 等人提出了在时间信息图下求解带费用限制的最短路径问题.他们假设图中的每条边附有出发时刻、到达时刻、长度和费用值,求解不超过费用上限的、路径上相邻的边满足时间顺序的、长度最短的路径.文献[1]提出的两种查询算法的时间复杂度分别是 $O(|E|\cdot\log|E|+|E|\cdot\Delta_{\max}\cdot L_{opt})$和$O(|E|\cdot P\cdot(t_\beta-t_\alpha))$,其中,$|E|$表示图的边数,$\Delta_{\max}$ 表示顶点的最大度数,L_{opt} 表示最短路径的长度,P 表示费用上限,t_α和t_β分别表示查询时间区间的下界和上界.第 1 种算法的时间复杂度依赖于图的规模,不适合大规模图的实时查询;第 2 种算法的时间复杂度依赖于路径长度、费用上限、查询时间区间等数值,当数值较大时,显然查询速度变慢.此外,文献[1]的实验中采用通过一条边的时间作为边的长度,查找不超过费用限制的最短路径返回旅途中乘坐交通工具所花的总时间,而等待交通工具的时间并未计入,这在路径规划中是不合理的.Yang 等人用分段函数表示通过一条边的费用依赖于出发时刻变化[2],在这种连续时间模型上,提出了一种给定查询时间区间的最小费用路径查询方法 Tow-Step;在文献[3]中,Yang 等人还进一步假设通过每条边的时间也依赖于出发时刻变化,提出了 Tow-Step 的改进算法.Tow-Step 算法允许路径在某个顶点上等待至恰当的时刻才继续前行,以达到路径费用最小.这种模型更适用在私人交通网络,因为公共交通运行有固定的时间表,不能在某个地方等待任意的时间.此外,Two-Step 算法也是基于原图上做查询,时间复杂度是 $O(k\cdot|V|\cdot\log|V|+|E|\cdot k^2\cdot\log k)$,其中,$|V|$表示顶点数,$|E|$表示边数,$k$ 表示边上时间分区的段数.同样地,该算法复杂度过高,不适用于大规模图的实时查询.Li 等人认为公交网络中容易受堵车等不确定因素影响,所以用随机网络刻画道路网络:每条边的通行时间是基于一组随机时间点的随机变量[4],并针对此模型提出一种拉格朗日松弛算法求解.何胜学等人提出了一种考虑公交线路票价变化、并以总行程时间最短与换乘次数最少相结合的计算模型与寻路算法[5];魏金丽等人就公交车“限时免费换乘”的模型,寻找限定时间最短时间与超时条件最低费用的路径[6].上述方法都是在原图上做查询,不适用于大图的实时查询.

另外,有更多工作研究非时间信息图上的带费用限制的最短路径查询.这些研究假设图中的每条边带有长度和费用两个值,求解不超过费用上限的长度最短的路径.由于这种查询需要枚举从起点到终点的所有路径,因此是一个 NP 难的问题[7].CSP-CH[8]借鉴路网中求最短路径的高效算法 Contraction Hierarchy[9]的思路建立索引.尽管查询得到加速,但是在预处理阶段添加的边太多,导致索引太大.Lozano 等人提出了一种在大规模图上求解的方法[10].Ma 提出一种快速的基于 A^*标签的算法求解受资源限制路径的问题[11].此外,为了快速求解,学者提出了求近似解的方法[12−14].Tsaggouris 等人提出的方法保证近似解的路径长度在最优解的路径长度的α倍范

围内[12],其中,α是一个大于 1 的预定义参数.Wang 等人在文献[13]的基础上提出了更高效的α-dijk 算法求解近似解,并借鉴了路网下查找最短路径的高效索引 Hub Labelling[15]的思想设计了索引 COLA[14],极大地提高了查询速度,可应用在真实的大规模路网的实时查询.

此外,关于时间信息图上的最小路径查询已经有了很多的研究工作.Cooke 等人最早提出最早到达路径、最晚出发路径和最短耗时路径这 3 种查询[16].随着大规模图的出现,有学者提出事先在图上建索引的方法加速查询.Geisberger 提出的 CHT[17]算法与上文提到的 CSP-CH 类似,也是采用 Contraction Hierarchy[9]的思路建立索引.文献[18−20]对原图的边预先排序,可以在线性时间复杂度内得到查询结果.Wang 等人提出的 TTL[21]和 Daniel 等人提出的 Public Transit Labelling[22]采用 Hub Labelling 的思想,预先计算出图中部分最短路径信息,然后通过对部分路径集合的线性扫描得到结果.文献[23,24]对时间信息图下求解最短路径的前沿工作做了深入的研究.上述方法只关注路径的时间信息,没有考虑上费用的限制.

综上所述,带费用限制的最短路径查询和时间信息图下最小时态路径查询这两个问题已经有索引方法提供高效的查询,但是目前,关于带费用限制的最小时态路径查询的研究均是在原图上做搜索,在大规模图上查询效率低下,因此需要仿照图查询的传统做法,预先对图建立索引,再设计相应的查询算法利用索引查询结果.

本文的贡献如下:提出了一种带费用限制最小时态路径近似查询的高效索引(approximate cost constrained time labelling,简称 ACCTL).ACCTL 对图中每个顶点预先计算一些从该顶点出发的路径和到达该顶点的路径,使得任意查询可以通过检索 ACCTL 中的路径生成解,避免在原图上做查询,从而提高查询效率.实验结果表明,基于 ACCTL 的查询比基于 Dijkstra 的变种查询算法快 2 个~3 个数量级.虽然 ACCTL 与 COLA[14]和 TTL[21]一样,都是采用 Hub Labelling 的结构建立索引,但 ACCTL 并不是简单地将 COLA 和 TTL 合并:COLA 中的路径长度是全序关系,而 ACCTL 中的时间区间是偏序关系,不能像长度那样可以两两比较出大小;TTL 中任意时刻出发的最快路径只有一条,而 ACCTL 中需要考虑上费用,任意时刻出发的最快路径有多条.在下文中,将会说明 ACCTL 包含了许多精心设计和优化.

本文第 1 节给出问题的描述和相关符号的定义.第 2 节给出一种基于 Dijkstra 算法的方法,这种方法是构建索引的核心算法.第 3 节介绍 ACCTL 的设计思路.第 4 节介绍 ACCTL 的查询算法.第 5 节介绍 ACCTL 的预处理构建索引算法.第 6 节采用交通数据集测试 ACCTL 的查询效率、空间大小和建立索引的时间.最后总结全文.

1 问题描述

本文沿用文献[1,17,19–22]中的时间信息图来刻画公共交通网络,并给每条边添加一个费用值.设 $G=(V,E)$是一个有向多重图,V是顶点的集合,每个顶点对应公共交通网络中的站点;E是边的集合,边 $e=\langle u,v,t_d,t_a,c\rangle$表示在 t_d时刻从顶点 u 出发,在 t_a时刻到达顶点 v,所花费用是 c.两点之间可以存在多条边.本文假设 t_d,t_a和 c 都是非负整数,这个假设是合理的,因为现实应用中不存在费用为负的开销;此外,总可以找到一个最小单位粒度来度量时间和费用.对于时刻 t_1和 t_2,用符号 $t_1\leqslant t_2$表示 t_1早于或等于 t_2,$t_1\geqslant t_2$表示 t_1晚于或等于 t_2.定义 $P=\langle e_1,e_2,...,e_k\rangle$是一条路径,若对于 $1\leqslant i<k$,e_{i+1}的出发顶点等于 e_i的到达顶点,且 e_{i+1}的出发时刻$\geqslant e_i$的到达时刻.定义 P 的出发时刻等于 e_1的出发时刻,P 的到达时刻等于 e_k的到达时刻,P 的耗时等于到达时刻与出发时刻之差.此外,P 的费用记为 $c(P)$,等于路径上的边的费用的总和.

本文研究的带费用限制的最早到达路径、最晚出发路径和最短耗时路径定义如下.

定义 1(带费用限制的最早到达路径). 给定图 G,起点 s,终点 d,时刻 t,费用上限θ,带费用限制的最早到达路径指所有在 t 时刻或之后从 s 出发,到达 d,费用不超过θ的路径中,具有最早到达时刻的路径.

定义 2(带费用限制的最晚出发路径). 给定图 G,起点 s,终点 d,时刻 t',费用上限θ,带费用限制的最晚出发路径指所有从 s 出发,在 t'时刻或之前到达 d,费用不超过θ的路径中,具有最晚出发时刻的路径.

定义 3(带费用限制的最短耗时路径). 给定图 G,起点 s,终点 d,时刻 t,t',费用上限θ,带费用限制的最短耗时路径指所有在 t 时刻或之后从 s 出发,在 t'时刻或之前到达 d,费用不超过θ的路径中,具有最短耗时的路径.

约定:若查询 Q 存在多条路径具有最早的到达时刻(或最晚的出发时刻、或最短的耗时),则返回费用最小的

路径作为解.图 1 是一个带费用值的时间信息图,不同的线形表示不同的交通工具及班次.边上的数字表示[出发时刻,达时刻]和费用.假设查询从 v_4 到 v_2 的费用上限是 30 的最早到达路径,要求出发时刻≥3,则查询应该返回路径$\langle e_2,e_3,e_4\rangle$.随着图的规模增大,两个顶点之间的路径数目暴涨.本文沿用 CSP 问题的做法引入近似因子α来控制近似解与精确解的误差程度.

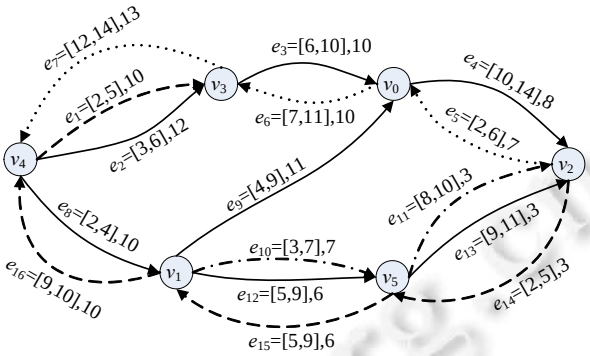


Fig.1 A time information graph with costs

图 1 一个带费用值的时间信息图

定义 4(带费用限制的最早到达路径查询的近似解). 给定近似因子α,带费用限制的最早到达路径查询 Q 返回的近似解 P 满足:(1) $c(P)\leqslant\alpha\cdot c(P_{opt})$;(2) P 的到达时刻≤P_{opt} 的到达时刻.其中,P_{opt} 是 Q 的精确解.

定义 5(带费用限制的最晚出发路径查询的近似解). 给定近似因子α,带费用限制的最晚出发路径查询 Q 返回的近似解 P 满足:(1) $c(P)\leqslant\alpha\cdot c(P_{opt})$;(2) P 的出发时刻≥P_{opt} 的出发时刻.其中,P_{opt} 是 Q 的精确解.

定义 6(带费用限制的最短耗时路径查询的近似解). 给定近似因子α,带费用限制的最短耗时路径查询 Q 返回的近似解 P 满足:(1) $c(P)\leqslant\alpha\cdot c(P_{opt})$;(2) P 的耗时≤P_{opt} 的耗时.其中,P_{opt} 是 Q 的精确解.

α是一个大于等于 1 的实数.当α=1 时,查询得出的解即为精确解.假设在图 1 中查找从 v_4 到 v_0 的带费用限制的最晚出发路径,t'=10,θ=21 精确解是$\langle e_1,e_3\rangle$.若α取 1.1,近似解是$\langle e_2,e_3\rangle$.

需要说明的是,虽然允许近似解 P 比它对应的查询 Q 的精确解 P_{opt} 的费用稍大,但并不等同于任意在 ACCTL 上的查询返回的 P 的费用总超过θ.实际上,ACCTL 的设计保证了:当$\theta\geqslant\alpha\cdot c(P_{opt})$时,ACCTL 查询返回的解 P 即为精确解;当$\theta<\alpha\cdot c(P_{opt})$,即最优路径的费用几乎达到费用上限时,可能返回精确解,也可能返回近似解.若返回的 P 是近似解,α限定了 P 的超支范围,同时,作为超支的补偿,P 比 P_{opt} 有更早的到达时刻或更晚的出发时刻或更短的耗时.换句话说,仅在最优解的费用几乎达到费用上限时,ACCTL 有可能返回费用稍微超支,但时间更优的近似解;其他情况下,ACCTL 返回精确解.下文将在第 5.2 节的定理 3 中给出证明.

表 1 给出了全文常用的符号及含义.

Table 1 Table of notations

表 1 符号表

符号	含义
$G=(V,E)$	图 G,V 是顶点的集合,E 是边集
$c(P)$	路径 P 的费用
α	近似因子控制误差范围,≥1 的实数
o	一个顶点的全序表示顶点的等级,$o(v)<o(u)$表示 v 比 u 高级,见第 3 节
$l=\langle t_d,t_a,c,l_{ptr}\rangle$	标签 l 表示路径在 t_d 时刻出发,t_a 时刻到达,费用 c,l_{ptr} 指向表示前缀路径的标签,见第 3.2 节的定义 10
L_u^v	标签集合,集合内的标签表示从顶点 u 到顶点 v 的路径,见第 3.2 节的定义 10
$\mathcal{L}_+(v)$(或$\mathcal{L}_-(v)$)	表示从 v 出发(或到达 v)的路径的标签组的集合,见第 3.2 节的定义 10

2 一种基于 Dijkstra 算法的方法

本节给出一种基于 Dijkstra 算法的 Dijk-CCMTP(Dijkstra-cost constrained minimal temporal path)算法,出于两点目的:首先,Dijk-CCMTP 是构建 ACCTL 索引的核心算法;其次,3 种查询可以通过调用 Dijk-CCMTP 的变种算法求得解,将作为实验中的对比算法.Dijk-CCMTP 与文献[1]中的算法相似,但由于本文所求解的问题和文献[1]不一样,所以两者的算法在细节上有差别.Dijk-CCMTP 按照费用值从小到大的顺序枚举不超过费用上限的路径,从中选择一条时间上最优的路径作为解;而文献[1]的算法选择长度最短的路径作为解.下文中首先给出 Dijk-CCMTP 算法的细节,再讨论如何用 Dijk-CCMTP 进行精确查询.

2.1 核心算法Dijk-CCMTP

求解带费用限制的最小时态路径需要用到的一个关键概念叫做路径支配.

定义 7(路径支配). 设 P_1 和 P_2 是两条连接相同起点和终点的路径,称 P_1 支配 P_2,或 P_2 被 P_1 支配,如果满足以下 3 个条件:(1) $c(P_1)\leqslant c(P_2)$;(2) P_1 的出发时刻$\geqslant P_2$ 的出发时刻;(3) P_1 的到达时刻$\leqslant P_2$ 的到达时刻.

若 P_1 支配 P_2,则表明 P_1 在费用和时间方面均不比 P_2 的差.若路径 P 不被其他路径支配,则称 P 是非支配路径(若两条路径的起点、终点、出发时刻、到达时刻和费用均相等,则任选一条作为非支配路径).

引理 1. 设 P 是查询 Q 的解,P_{sub} 是 P 上的一段子路.如果存在路径 P'_{sub} 支配 P_{sub},则用 P'_{sub} 代替 P_{sub} 所得到的路径也是 Q 的解.

算法 1 描述了 Dijk-CCMTP 算法的伪代码,求解在 t 时刻从 s 出发,费用不超过θ,到达 d 的最早到达时刻.算法从 s 开始,按路径的费用值升序,费用值相等的按到达时刻升序的顺序扩展路径.这种顺序可以在搜索的早期找到费用值小的、到达时刻早的路径,有助于利用引理 1 剪枝掉被支配的路径.算法维护一个最小堆 H 记录当前所找到的路径.每条路径用元组$\langle u,t_a^*,c^*\rangle$表示,指路径在 t 时刻从 s 出发,在t_a^*时刻到达 u,费用是 c^*.Dijk-CCMTP 循环地从 H 中取出路径扩展出新路径,直到 H 为空.

算法 1. Dijk-CCMTP 算法.

输入:出发时刻 t,起点 s,终点 d,费用上限θ.

输出:在 t 时刻从 s 出发的、费用不超过θ的、到达 d 的最早到达时刻.

1. $H=\varnothing$;
2. 对所有的边 e,令$\mathcal{C}(e)=\infty$;
3. 令$\mathcal{T}(s)=t$,对于除 s 以外的其他顶点 u,令$\mathcal{T}(u)=\infty$;
4. 对于从 s 出发的、出发时刻为 t 的边 e,生成一条仅包含 e 的路径,加入 H.记$\mathcal{C}(e)$等于 e 的费用.
5. **while** $H\neq\varnothing$ **do**
6. 设 $\rho=\langle u,t_a^*,c^*\rangle$ 为堆顶元素,摘除ρ.
7. **if** $u=d$, **then continue**.
8. **if** $t_a^*\geqslant\mathcal{T}(u)$, **then continue**.
9. 令 $\mathcal{T}(u)=t_a^*$;
10. **foreach** u 的出边 $e=\langle u,w,t_d,t_a,c\rangle$ **do**
11. **if** $t_d\geqslant t_a^*$ **then**
12. $\rho_{new}=\langle w,t_a,c^*+c\rangle$;
13. **if** $c^*+c\geqslant\mathcal{C}(e)$或 $c^*+c+c_\perp(w,d)>\theta$ **then continue**;
14. 令$\mathcal{C}(e)=c^*+c$;将ρ_{new} 加进 H;
15. 返回$\mathcal{T}(d)$;

算法维护两个映射,帮助剪枝掉一些不可能生成解的路径.映射$\mathcal{T}(u)$记录了当前发现的到达 u 的最早到达时刻.在第 8 行,如果 $t_a^*\geqslant\mathcal{T}(u)$,则$\rho$可以被剪枝掉,这是因为:(1) $t_a^*\geqslant\mathcal{T}(u)$ 说明在ρ之前存在路径ρ'出堆,ρ'的到

达时刻更新了$\mathcal{T}(u)$(第9行),而由于路径按费用的升序出堆,ρ'的费用小于等于ρ的费用;(2) ρ和ρ'有相同的起点s和出发时刻t.因此,ρ'支配ρ,ρ可以被剪枝掉.另一个映射是$\mathcal{C}(e)$,记录最后一条边是e的路径的最小费用值.如果ρ_{new}的费用大于等于$\mathcal{C}(e)$,表明已存在路径ρ',ρ'和ρ_{new}有相同的起点和终点,且出发时刻与到达时刻也相同,但费用小于等于ρ_{new}的,因此ρ'支配ρ_{new},则ρ_{new}可以被剪枝掉(第13行).

除了上述两个映射以外,还利用费用值下限进行剪枝.第13行的$c_{\perp}(w,d)$表示点w到d的费用下限.可以从d出发调用一次逆向的Dijkstra算法,扩展路径时不考虑边与边之间的时间顺序约束,计算出图中其他顶点到达d的费用下限.如果ρ_{new}的费用(c^*+c加上ρ_{new}的终点w到达d的费用的下限已超过θ,则ρ_{new}不可能生成解,不需要加进H.

引理2. H最多包含$|E|$条路径,其中,$|E|$表示图的边数.

证明:只需证明加进H的路径的最后一条边各不相同.因为H中的路径是按照费用值升序的顺序出堆的,因此在所有的最后一条边是e的路径中,最早被发现的路径具有最小费用,加进H;之后被发现的路径受第13行的条件$c^*+c\geqslant\mathcal{C}(e)$的约束,不会加进$H$.因此$H$的路径总数不超过$|E|$. □

算法中采用费用值而不是布尔类型变量记录映射$\mathcal{C}(e)$,是为了优化带费用限制的最短耗时路径查询,将在第2.2节中加以说明.

引理3. 算法Dijk-CCMTP的时间复杂度是$O(|E|\cdot(\log|E|+\Delta_{\max}))$,其中,$|E|$表示图的边数,$\Delta_{\max}$表示顶点的最大出度.

证明:H用二项堆实现,第6行ρ出堆的复杂度是$O(\log|E|)$.第10行枚举点的出边,最多循环$\Delta_{\max}$次,因此,H中处理每条路径的时间复杂度是$O(\log|E|+\Delta_{\max})$.H最多包含$|E|$条路径,因此时间复杂度是$O(|E|\cdot(\log|E|+\Delta_{\max}))$. □

2.2 基于Dijk-CCMTP的查询算法

本节首先讨论如何调用Dijk-CCMTP查询带费用限制的最早到达路径.考虑一个从s到d的、出发时刻$\geqslant t$、费用不超过θ的查询.可以将Dijk-CCMTP算法的第4行的条件"出发时刻为t"改成"出发时刻$\geqslant t$".由引理1可以证明,这个改动仍然能够保证算法的正确性.

带费用限制的最晚出发路径查询与带费用限制的最早到达路径查询是对称的.带费用限制的最晚出发路径查询从终点d出发做反向搜索扩展回起点,即扩展路径时访问顶点关联的入边.在最小堆H中,费用小的路径优先出堆;费用相等的路径出发时刻晚的优先出堆.

带费用限制的最短耗时路径查询并不能调用一次Dijk-CCMTP求解,这是因为路径的耗时与路径的出发时刻、到达时刻均有关系,因此需要枚举s的不同的出发时刻,多次调用Dijk-CCMTP求解.留意到路径P只可能被比出发时刻不早于它的路径所支配,所以可以按出发时刻从晚到早的顺序调用Dijk-CCMTP.每次调用Dijk-CCMTP时重用前一轮调用Dijk-CCMTP时设置的$\mathcal{C}(e)$值,即调用了Dijk-CCMTP求解在t_1时刻出发的路径以后,下一轮求在t_2时刻($t_2<t_1$)出发的路径时不需要将$\mathcal{C}(e)$的值重置为∞.$\mathcal{C}(e)$的含义和引理1保证了正确性.

定理1. Dijk-CCMTP的变种算法查询带费用限制的最早到达路径和带费用限制的最晚出发路径的时间复杂度是$O(|E|\cdot(\log|E|+\Delta_{\max}))$,查询带费用限制的最短耗时路径的时间复杂度是$O(\Delta_{\max}\cdot|E|\cdot(\log|E|+\Delta_{\max}))$,其中,$|E|$表示图的边数,$\Delta_{\max}$表示顶点的最大度数.

定理1可以由引理1和引理3推导证明.

3 索引ACCTL的框架

本节介绍索引ACCTL的设计思想.ACCTL的构建基于一个顶点的全序关系,全序关系用来衡量顶点在图中的重要程度.在公共交通网络中,一些交通枢纽很重要,因为相距较远的两个地方通常在交通枢纽换乘.表现在图中,一个顶点所在的带费用限制的最小时态路径越多,它就越重要,称它等级越高.本文用$o(v)$表示将顶点按重要程度从高到低排序后,顶点v在序列中的位置.若$o(v)<o(u)$,则表明v的重要程度排在u的前面,即v的等级比u高.o的选取并不影响查询结果的正确性,但会影响索引的大小,进而影响查询效率.下文假定已知o,将在第

5.3 节中讨论 o 的计算.

3.1 ACCTL的设计思想

设 P 是一条从点 s 到点 d 的路径,v_T 是 P 上的顶点中等级最高的点.v_T 在 P 中所处的位置有 3 种可能:(1) $v_T=s$;(2) $v_T=d$;(3) $v_T\neq s$ 且 $v_T\neq d$.设 P 上从 s 到 v_T 的子路记为 P_+,从 v_T 到 d 的子路记为 P_-.情况(1)和情况(2)可视为情况(3)的特例,即 P_+ 或 P_- 为空.如果预先计算出一些从 s 到比 s 高级的顶点的路径(这些路径的集合记为 S_+)和一些从比 d 高级的顶点到 d 的路径(这些路径的集合记为 S_-),那么对于一个从 s 到 d 的查询,可以从 S_+ 中查找 P_+、从 S_- 中查找 P_-,使得 P_+ 和 P_- 能够连接成一条路径生成解.这种查找方法类似于数据库的表连接操作.如果设一张数据库表 T 记录任意查询 Q 的解,ACCTL 的作用就是把 T 分解成两张表 T_1 和 T_2,使得 T 中的记录可以通过 T_1 和 T_2 连接操作生成.于是,ACCTL 避免在原图 G 上遍历,从而减少搜索量.由引理 1,任何查询的解均可以由两条非支配路径生成,因此 ACCTL 只需要保存一些基本路径即可.

定义 8(基本路径). P 是一条基本路径,如果 P 满足以下两个条件:

(1) 等级约束:P 上的顶点中,起点或终点的等级最高;

(2) 支配约束:P 是非支配路径.

假设图 1 中的顶点的等级由高到低依次为 v_0~v_5.$\langle e_2,e_3\rangle$ 是一条基本路径;$\langle e_2,e_3,e_4\rangle$ 违反了等级约束,不是基本路径.

随着图的规模增大,图中基本路径的数目会暴涨,因此,ACCTL 需要借助近似因子 α 去掉部分基本路径,允许生成近似解.生成近似解的一个关键概念是路径的 α-支配.

定义 9(路径的 α-支配). 设 P_1 和 P_2 是两条连接相同起点和终点的路径,α 是近似因子.称 P_1 α-支配 P_2,或 P_2 被 P_1 α-支配,如果满足:(1) $c(P_1)\leqslant\alpha\cdot c(P_2)$;(2) P_1 的出发时刻 $\geqslant P_2$ 的出发时刻;(3) P_1 的到达时刻 $\leqslant P_2$ 的到达时刻.

定义 9 与定义 7 的区别在于,对路径的费用稍放宽至近似因子 α 倍范围内.如图 1 所示,假设路径 $P_1=\langle e_1,e_3\rangle$,$P_2=\langle e_2,e_3\rangle$,$\alpha$ 取 1.1,则 P_2 α-支配 P_1.结合定义 4~定义 6 和定义 9,可以得到以下引理.

引理 4. 设 P 是查询 Q 的精确解,P' α-支配 P,则 P' 是 Q 的近似解.

表 2 是 ACCTL 的雏形.第 2 列的每条标签 $\langle u,t_d,t_a,c,l_{ptr}\rangle$ 表示路径在 t_d 时刻从 v 出发,在 t_a 时刻到达 u,费用 c,l_{ptr} 指向表示前缀路径的标签(表中每条标签的最后一个分量(l_{ptr})为 null 表示该标签对应的路径不存在前缀路径,为*表示前缀路径的标签不在表中),其具体含义将在定义 10 中说明.例如,$l_0=\langle v_0,2,6,7,\text{null}\rangle$ 表示在 2 时刻从 v_2 出发,6 时刻到达 v_0,费用是 7.第 3 列的标签表示从 u 到 v 的路径.假设查询从 v_4 到 v_2 的费用上限是 30 的最早到达路径,要求出发时刻 $\geqslant$3.查询过程将标签 $l_7,l_8,\ldots,l_{11}$ 和 l_2,l_3,l_4 进行匹配.标签 l_a 和 l_b 相匹配的意思是:(1) l_a 表示的路径的终点与 l_b 表示的路径的起点相同;(2) l_b 的出发时刻 $\geqslant l_a$ 的到达时刻;(3) l_a 和 l_b 的费用之和不超过费用上限.本例中,查询返回 l_8 和 l_2 匹配的路径,即 $\langle e_2,e_3,e_4\rangle$,在 3 时刻出发,14 时刻到达,费用是 30.

Table 2 Some canoical paths from/to v_2~v_4 against Fig.1 with α=1.1

表 2 图 1 的 v_2~v_4 出发/到达的部分基本路径,α=1.1

v	从 v 出发的路径	到达 v 的路径
v_2	$l_0=\langle v_0,2,6,7,\text{null}\rangle$,$l_1=\langle v_1,2,9,9,*\rangle$	$l_2=\langle v_0,10,14,8,\text{null}\rangle$,$l_3=\langle v_1,3,10,10,*\rangle$,$l_4=\langle v_1,5,11,9,*\rangle$
v_3	$l_5=\langle v_0,6,10,10,\text{null}\rangle$	$l_6=\langle v_0,7,11,10,\text{null}\rangle$
v_4	$l_7=\langle v_0,2,9,21,*\rangle$,$l_8=\langle v_0,3,10,22,l_5\rangle$,$l_9=\langle v_1,2,4,10,\text{null}\rangle$ $l_{10}=\langle v_3,2,5,10,\text{null}\rangle$,$l_{11}=\langle v_3,3,6,12,\text{null}\rangle$	$l_{12}=\langle v_0,7,14,23,l_6\rangle$,$l_{13}=\langle v_1,9,10,10,\text{null}\rangle$ $l_{14}=\langle v_3,12,14,13,\text{null}\rangle$

3.2 标签去冗余

表 2 的标签存在冗余,影响到索引的存储空间大小和查询效率.例如 l_3 和 l_4 均表示从 v_1 到 v_2 的路径,顶点 v_1 只需要保存一份即可,无需在 l_3 和 l_4 中均保存.另外,仅具有共同端点的标签才有可能匹配上.例如查询从 v_4 到 v_2 的路径时,可能与 l_9 匹配的标签只有 l_3 和 l_4,因为它们的起点 v_1 等于 l_9 的终点.l_2 的起点与 l_9 的终点不相同,因此查询时无需扫描 l_2.基于上述考虑,ACCTL 将表示连接相同顶点的路径的标签分成一个标签组.表 3 显示了分组的标签,这也是 ACCTL 的存储格式.

定义 10(ACCTL 索引). 给定近似因子α,图 G 的 ACCTL 索引预先对每个顶点 v 计算两个标签组集合$\mathcal{L}_+(v)$和$\mathcal{L}_-(v)$,满足:

(1) $\mathcal{L}_+(v)=\{L_v^{u_0},L_v^{u_1},...,L_v^{u_{k-1}}\}$ 是一个标签组的集合.对于 $0\leqslant i<k$,u_i 是一个顶点,且有 $o(u_i)<o(v)$. $L_v^{u_i}\in\mathcal{L}_+(v)$ 是一个标签的集合,$L_v^{u_i}$ 内的标签表示从 v 到 u_i 的基本路径.不失一般性,假设 $L_v^{u_0},L_v^{u_1},...,L_v^{u_{k-1}}$ 已按 u_i 的等级排序,有 $o(u_0)<o(u_1)<\ldots<o(u_{k-1})$.标签 $l=\langle t_d,t_a,c,l_{ptr}\rangle\in L_v^{u_i}$ 表示一条从 v 到 u_i 的基本路径 P,在 t_d 时刻出发,t_a 时刻到达,费用是 c,l_{ptr} 指向表示 P 上从 v 的直接后继顶点到 u_i 的那段子路的标签. $L_v^{u_i}$ 内的标签按照出发时刻升序、出发时刻相等的按到达时刻升序排列.

(2) $\mathcal{L}_-(v)=\{L_{u_0}^v,L_{u_1}^v,...,L_{u_{k-1}}^v\}$ 是一个标签组的集合.对于 $0\leqslant i<k$,u_i 是一个顶点,且有 $o(u_i)<o(v)$. $L_{u_i}^v\in\mathcal{L}_-(v)$ 是一个标签的集合,$L_{u_i}^v$ 内的标签表示从 u_i 到 v 的基本路径.不失一般性,假设 $L_{u_0}^v,L_{u_1}^v,...,L_{u_{k-1}}^v$ 已按 u_i 的等级排序,有 $o(u_0)<o(u_1)<\ldots<o(u_{k-1})$.标签 $l=\langle t_d,t_a,c,l_{ptr}\rangle\in L_{u_i}^v$ 表示一条从 u_i 到 v 的基本路径 P,在 t_d 时刻出发,t_a 时刻到达,费用是 c,l_{ptr} 指向表示 P 上从 u_i 到 v 的直接前驱顶点那段子路的标签. $L_{u_i}^v$ 内的标签按照到达时刻升序、到达时刻相等的按出发时刻升序排列.

(3) 对于任意的从顶点 s 到顶点 d 的带费用限制的最小时态路径查询 Q,设 P_{opt} 是 Q 的精确解,则在$\mathcal{L}_+(s)$内的某个标签组 L_s^u 中,存在标签 $l_+=\langle t_d^+,t_a^+,c^+,\cdot\rangle$ (下文中,对于标签中暂不需关注的项采用符号·表示)的对应路径 P_+,在$\mathcal{L}_-(d)$内的某个标签组 L_v^d 中,存在标签 $l_-=\langle t_d^-,t_a^-,c^-,\cdot\rangle$ 的对应路径 P_-,以下 3 种情况之一成立.

(3.1) $u=d$,且 $P_+$$\alpha$-支配 P_{opt};

(3.2) $v=s$,且 $P_-$$\alpha$-支配 P_{opt};

(3.3) $u=v$, $t_a^+\leqslant t_d^-$,且 P_+和 P_-连接起来的路径α-支配 P_{opt}.

Table 3 $\mathcal{L}_+(v)$ and $\mathcal{L}_-(v)$ label sets of v_2~v_4 of ACCTL against Fig.1 with α=1.1

表 3 图 1 的 ACCTL 索引中 v_2~v_4 的$\mathcal{L}_+(v)$和$\mathcal{L}_-(v)$集合,α=1.1

v	$\mathcal{L}_+(v)$		$\mathcal{L}_-(v)$	
v_2	$L_{v_2}^{v_0}$	l_0=⟨2,6,7,null⟩	$L_{v_0}^{v_2}$	l_2=⟨10,14,8,null⟩
	$L_{v_2}^{v_1}$	l_1=⟨2,9,9,*⟩	$L_{v_1}^{v_2}$	l_3=⟨3,10,10,*⟩,l_4=⟨5,11,9,*⟩
v_3	$L_{v_3}^{v_0}$	l_5=⟨6,10,10,null⟩	$L_{v_0}^{v_3}$	l_6=⟨7,11,10,null⟩
v_4	$L_{v_4}^{v_0}$	l_7=⟨2,9,21,*⟩,l_8=⟨3,10,22,l_5⟩	$L_{v_0}^{v_4}$	l_{12}=⟨7,14,23,l_6⟩
	$L_{v_4}^{v_1}$	l_9=⟨2,4,10,null⟩	$L_{v_1}^{v_4}$	l_{13}=⟨9,10,10,null⟩
	$L_{v_4}^{v_3}$	l_{10}=⟨2,5,10,null⟩,l_{11}=⟨3,6,12,null⟩	$L_{v_3}^{v_4}$	l_{14}=⟨12,14,13,null⟩

下文为了表述清晰,首先在第 4 节介绍查询算法,然后在第 5 节讨论索引的构建方法,因为索引构建方法比查询方法复杂.

4 查询算法

查询从点 s 到点 d 的带费用限制的最小时态路径分个两阶段:首先,检索$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$得到符合查询要求的标签;然后,将标签还原回一条图 G 上的路径.下文分别介绍查找标签和还原路径的过程.

4.1 查找标签

本节讨论如何用 ACCTL 查询从 s 到 d、出发时刻≥t、费用上限为θ的最早到达路径近似解,另外两种查询可作近似的处理.定义 10 的条件(3)大致给出了查询过程.$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$内的标签分别记录了路径的前半部分和后半部分的信息,查询过程就是找出$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$内的相匹配的标签,生成候选路径,再从候选路径中挑选具有最早到达时刻的路径作为解.

算法2描述了带费用限制的最早到达路径的查询算法.$t_\perp$表示当前找到的路径的最早到达时刻,在查询过程中,利用 $t_\perp$减少不必要的标签匹配.初始时,$t_\perp=\infty$.c_T是一个扩大α倍的上限,允许近似解的费用稍微超过θ.算法第2行~第5行处理定义10中条件(3)的情况(3.1)和情况(3.2).

算法 2. EAPQuery.

输入:起点 s,终点 d,时刻 t,费用上限θ.

输出:构成费用不超过θ的最早到达路径的标签 l_+^* 和 l_-^*.

1. 初始化 $t_\perp=\infty$;$c_T=\theta\times\alpha$; l_+^* 和 l_-^* 设为 null;
2. **if** $\mathcal{L}_+(s)$中包含标签组 L_s^d **then**
3. 如果 L_s^d 内存在费用不超过 c_T的标签,令 l_+^* 为具有最早到达时刻的那条标签.更新 $t_\perp$为 l_+^* 的到达时刻;
4. **if** $\mathcal{L}_-(d)$中包含标签组 L_s^d **then**
5. 如果 L_s^d 内存在费用不超过 c_T的标签,令 l_-^* 为具有最早到达时刻的那条标签.如果 l_-^* 的到达时刻比 $t_\perp$ 更早,更新 $t_\perp$为 l_-^* 的到达时刻;
6. i=0, j=0;
7. **while** $i<|\mathcal{L}_+(s)|$且 $j<|\mathcal{L}_-(d)|$ **do**
8. //检查标签组 $L_s^{u_i}$ 和 $L_{u_j}^d$
9. **if** $o(u_i)<o(u_j)$ **then** $i=i+1$;
10. **else if** $o(u_i)>o(u_j)$ **then** $j=j+1$;
11. **else**
12. 设 $l_+=\langle t_d^+,t_a^+,c^+,\cdot\rangle$ 是 $L_s^{u_i}$ 中的第 1 条 $t_d^+\geqslant t$ 的标签;
13. **while** $L_s^{u_i}$ 内的标签还没遍历完 **do**
14. 令 $l_-=\langle t_d^-,t_a^-,c^-,\cdot\rangle$ 指向 $L_{u_j}^d$ 的首条标签;
15. **while** $L_{u_j}^d$ 内的标签还没遍历完 **do**
16. **if** $t_a^-\geqslant t_\perp$ **then break**;
17. **if** $t_a^+\leqslant t_d^-$ 且 $c^+ + c^-\leqslant c_T$ **then**
18. 记录 $l_+^*=l_+$;$l_-^*=l_-$;$t_\perp=t_a^-$; **break**;
19. 令 l_-指向下一条标签;
20. 令 l_+指向下一条标签;
21. $i=i+1$, $j=j+1$;
22. 返回 $t_\perp$, l_+^* 和 l_-^*;

接下来,算法处理等级最高的点出现在路径内部的情况,对应定义10中条件(3)的情况(3.3).指针 i 和 j 分别指示当前正在扫描$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$的第几组标签.算法对$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$内的标签组进行线性扫描.假设当前正在扫描$\mathcal{L}_+(s)$内的标签组 $L_s^{u_i}$ 和$\mathcal{L}_-(d)$内的标签组 $L_{u_j}^d$.如果 $o(u_i)<o(u_j)$,则表明 $L_s^{u_i}$ 内的标签与 $L_{u_j}^d$ 的不匹配,而在$\mathcal{L}_+(s)$中可能与 $L_{u_j}^d$ 内的标签匹配的标签组排在 $L_s^{u_i}$ 之后,于是令 i 指向下一组标签(第 9 行).对称地,如果 $o(u_i)>o(u_j)$令 j 指向下一组标签(第 10 行).

当 $o(u_i)=o(u_j)$,即 $u_i=u_j$ 时, $L_s^{u_i}$ 内的标签有可能与 $L_{u_j}^d$ 内的标签匹配.留意到 $L_s^{u_i}$ 内的标签按出发时刻升序排列,所以可调用二分查找快速定位到 $L_s^{u_i}$ 中的第 1 条 $t_d^+\geqslant t$ 的标签 $l_+=\langle t_d^+,t_a^+,c^+,\cdot\rangle$ (第 12 行),符合 $t_d^+\geqslant t$ 要求的标签必定都排在 l_+之后.在匹配 l_+和 $L_{u_j}^d$ 内的标签时,从 $L_{u_j}^d$ 内的第 1 条标签开始按顺序遍历.记 l_-为当前扫描的 $L_{u_j}^d$ 内的标签.当 l_-的到达时刻$\geqslant t_\perp$时,对 $L_{u_j}^d$ 的遍历便可终止,因为 $L_{u_j}^d$ 内的标签按到达时刻升序排列,排在 l_-之后的

标签不可能生成到达时刻早于 $t_\perp$的解.

假设在表 3 的 ACCTL 索引中查找从 v_4 到 v_2、在 3 时刻或之后出发的、费用不超过 30 的最早到达路径.初始时 $t_\perp=\infty$.首先,$\mathcal{L}_+(v_4)$中没有终点是 v_2 的标签组,$\mathcal{L}_-(v_2)$中也没有起点是 v_4 的标签组;接着,从$\mathcal{L}_+(v_4)$的终点为 v_0 的标签组找到第 1 条出发时刻≥3 的标签 $l_8=\langle 3,10,22,l_5\rangle$;接着,检查$\mathcal{L}_-(v_2)$中以 v_0 为起点的第 1 条标签 $l_2=\langle 10,14,8,\text{null}\rangle$,于是得到一条由 l_8 和 l_2 拼接的候选路径,更新 $t_\perp=14$;接下来扫描$\mathcal{L}_+(v_4)$的下一个标签组 $L_{v_4}^{v_1}$,由于 $L_{v_4}^{v_1}$ 中不存在出发时刻≥3 的标签,所以继续检查下一个标签组 $L_{v_4}^{v_3}$.由于$\mathcal{L}_-(v_2)$不存在以 v_3 为起点的标签组,所以此时$\mathcal{L}_-(v_2)$的标签组扫描完毕,查询结束,返回 l_8 和 l_2,表示结果为从 3 时刻出发、在 14 时刻到达、费用为 30 的路径.

4.2 路径还原

算法 2 仅获得表示查询的解的标签,需要将标签还原成图 G 上的一条路径.出于对称性,下文仅讨论如何将 l_-^* 还原.假设 $l_-^*=\langle t_d,t_a,c,l_{ptr}\rangle$ 表示一条从顶点 u 到 d 的路径 P_-.设 v_{pred} 表示 P_-上 d 的直接前驱顶点.由定义 10,l_{ptr} 表示 P_-上从 u 到 v_{pred} 的那一段子路(记为 P_{pred}).于是,可通过还原 l_{ptr} 获得 P_{pred}.假设路径由 k 条边构成,则还原路径的复杂度是 $O(k)$.

本文为了表述清晰,用 l_{ptr} 指代表示路径前缀的标签.在实际编码中,l_{ptr} 通过记录 v_{pred} 和 pos 实现,其中,pos 表示 P_{pred} 的标签在$\mathcal{L}_-(v_{pred})$中的标签组 $L_u^{v_{pred}}$ 中的存放位置.需要说明的是,$L_u^{v_{pred}}$ 必定包含了 l_{ptr}.这是因为:(1) P_{pred} 上的顶点中,起点 u 的等级最高,理由是 P_-上 u 的等级最高;(2) 下文将在第 5.1 节中解释,ACCTL 采用 Dijk-CCMTP 算法构造 P_-,P_{pred} 是非支配路径;并且在第 5.2 节中解释 l_{ptr} 会保留在 ACCTL 中,不会被删除.

5 ACCTL 索引的构建

本节解释如何构建 ACCTL 索引.ACCTL 的正确性依赖于两点:(1) 对于任意查询 Q,设精确解是 P_{opt},总可以在 ACCTL 找到路径 P,使得 P 是 α-支配 P_{opt},由引理 4,P 是 Q 的近似解;(2) 从 ACCTL 中查找到的标签总可以还原成图 G 上的一条路径.因此,构建 ACCTL 的难点在于:(1) 如何有效地、不遗漏地计算出所有基本路径?(2) 如何尽量多地去掉一些路径,但仍能保证路径被还原?对应地,ACCTL 的构建有两个步骤:一是生成基本路径;二是剪枝掉一些基本路径.下文分别在第 5.1 节和第 5.2 节解释这两个步骤,在第 5.3 节讨论顶点的等级序 o 的计算.

5.1 计算基本路径

计算基本路径的一种直观的做法是:枚举符合等级约束的路径,从中选择非支配路径,得到基本路径.初始时,设 $G_0=G$,等级最高的顶点记为 v_0,G_0 中所有从 v_0 出发和到达 v_0 的路径均满足等级约束.接下来,将 v_0 从 G_0 中删除,生成图 G_1.在 G_1 中,等级最高的顶点记为 v_1,从 v_1 出发和到达 v_1 的路径均满足等级约束,….重复上述步骤,即可计算出符合等级约束的路径.算法 3 描述了构建索引算法 BuildIndex.第 3 行~第 26 行的 for 循环迭代地计算从 v_i 出发的和到达 v_i 的基本路径,并选择部分路径生成标签加进 ACCTL 中.

算法 3. BuildIndex.

输入:图 $G=(V,E)$,顶点等级序 o,近似因子α.

输出:生成所有顶点 $v\in V$ 的$\mathcal{L}_+(v)$和$\mathcal{L}_-(v)$,构成 ACCTL 索引.

1. $G_0=G$;
2. 设 $v_0,v_1,...,v_{N-1}$ 是 G 中的顶点按等级 o 从高到低排序后的顶点序列;
3. **for** $i=0,1,...,N-1$ **do**
4. 设 $T=\{t_{k-1},t_{k-2},...,t_0\}$是 G_i 中从 v_i 出发的时刻的集合,$t_{k-1}>t_{k-2}>...>t_0$;
5. 对于 G_i 中的所有边 e,设$\mathcal{C}(e)=\infty$;
6. **for** $t=t_{k-1},t_{k-2},...,t_0$ **do**
7. $H=\varnothing$;对于 G_i 中的所有顶点 u,令$\mathcal{B}(u)=\varnothing$;

8. **foreach** 图 G_i 中从 v_i 出发的、出发时刻为 t 的边 e **do**
9. 生成一条仅包含 e 的路径ρ,加进 H 中;
10. **while** $H\neq\varnothing$ **do**
11. 设$\rho=\langle u,t_a^*,c^*,\rho_{pred}\rangle$是 H 的堆顶的路径,摘除ρ;
12. 设最近添加进$\mathcal{B}(u)$的路径是ρ'. **if** ρ'的到达时刻$\leqslant t_a^*$ **then continue**;
13. 将ρ添加进$\mathcal{B}(u)$;
14. **foreach** 图 G_i 中从 u 出发的每条边 $e=\langle u,w,t_d,t_a,c\rangle$ **do**
15. **if** $t_d<t_a^*$ **then continue**;
16. 生成路径$\rho_{new}=\langle w,t_a,c^*+c,\rho\rangle$;
17. **if** $c^*+c\geqslant\mathcal{C}(e)$ **then continue**; **else** 令$\mathcal{C}(e)=c^*+c$,将ρ_{new}加进 H;
18. **foreach** G_i 中的顶点 u **do**
19. 设 $L_{v_i}^u$ 是$\mathcal{L}_-(u)$中表示从 v_i 到 u 的路径的标签组.调用 *MarkPrune*($\mathcal{B}(u)$, $L_{v_i}^u$);
20. **foreach** $\rho=\langle u,t_a,c,\rho_{pred}\rangle\in\mathcal{B}(u)$,其中,$u$ 为 G_i 中的顶点 **do**
21. **if** ρ标记为保留但ρ_{pred}标记为删除 **then**
22. 标记ρ_{pred}为保留.追溯检查ρ_{pred}的前缀路径 ρ'_{pred},直到 ρ'_{pred} 标记为保留;
23. **foreach** $\rho=\langle u,t_a,c,\rho_{pred}\rangle\in\mathcal{B}(u)$,其中,$u$ 为 G_i 中的顶点 **do**
24. 设 $L_{v_i}^u$ 是$\mathcal{L}_-(u)$中表示从 v_i 到 u 路径的标签组.生成标签$\langle t,t_a,c,l_{ptr}\rangle$加进 $L_{v_i}^u$,其中,l_{ptr} 指向表示ρ_{pred}的标签;
25. **foreach** G_i 中的顶点 u **do** 对$\mathcal{L}_-(u)$中的标签组 $L_{v_i}^u$ 内的标签排序;
26. 仿照第 2 行~第 25 行生成到达 v_i 的基本路径,选择部分路径生成标签集合;
27. 将 v_i 从 G_i 中删掉,生成 G_{i+1};
28. 返回所有顶点 $v\in V$ 的$\mathcal{L}_+(v)$和$\mathcal{L}_-(v)$;

如何从满足等级约束的路径中有效地计算出非支配路径?假设计算从 v_i 出发的基本路径.由于在 t 时刻出发的路径仅可能被在t'时刻($t'\geqslant t$)出发的路径支配,所以按 v_i 的出发时刻降序的顺序调用 Dijk-CCMTP(第 6 行).第 7 行~第 17 行的处理与 Dijk-CCMTP 类似,但做了以下改动.

(1) 保存在堆 H 中的路径$\rho=\langle u,t_a,c,\rho_{pred}\rangle$增加一个分量$\rho_{pred}$,表示$\rho$的前缀路径.

(2) 对于 G_i 中的每个顶点 u,用一个包$\mathcal{B}(u)$收集从 v_i 到 u 的非支配路径.ρ从 H 摘除后(第 11 行),仅当ρ不被$\mathcal{B}(u)$中的路径支配时才将ρ加进$\mathcal{B}(u)$.回顾 H 中的路径,按照费用升序出堆;另一方面,第 10 行~第 17 行的 while 循环计算的路径有相同的起点(s)和出发时刻(t),可推出路径按到达时刻降序的顺序加入$\mathcal{B}(u)$,所以只需要比较ρ和最近一条加入$\mathcal{B}(u)$的路径的到达时刻,即可得出ρ是否被$\mathcal{B}(u)$内的路径支配(第 12 行).

(3) 由于终点 d 和费用上限θ在构建索引时未知,所以在 BuildIndex 中不利用 d 和θ进行剪枝.

当在 t 时刻从 v_i 出发的路径遍历完后,对于每个可达顶点 u,它的包$\mathcal{B}(u)$包含了到达 u 的非支配路径.第 19 行的 MarkPrune 算法从$\mathcal{B}(u)$中选择部分路径加入 ACCTL,选择细节将在第 5.2 节中讨论.MarkPrune 独立地选择$\mathcal{B}(u)$中的路径加进 ACCTL,即是说,MarkPrune 在选择$\mathcal{B}(u)$中的路径时并不考虑另一个顶点 v 的包$\mathcal{B}(v)$内的路径,其后果可能导致还原路径时出错.假设在选择$\mathcal{B}(u)$中的路径时,路径$\rho=\langle\cdot,\cdot,\cdot,\rho_{pred}\rangle$被选中,生成相应的标签 l 加进 ACCTL.为了保证标签 l 能正确被还原,表示路径ρ_{pred}的标签 l_{ptr} 必须在 ACCTL 中.然而 l_{ptr} 可能不在 ACCTL 中,因为 MarkPrune 在选择$\mathcal{B}(v_{pred})$内的路径(假设 v_{pred} 是ρ上 u 的直接前驱顶点)时并没有考虑ρ.因此,在调用完 MarkPrune 以后,需要自底向上地检查每条被保留的路径是否可被还原.加进 ACCTL 的标签是否可被还原,取决于它对应的路径ρ的前缀路径ρ_{pred}是否也生成对应的标签加进 ACCTL.对于每条被 MarkPrune 算法标记为保留的路径ρ,如果ρ_{pred}也被标记成保留,则对ρ的检查结束;否则,将ρ_{pred}标记成保留,并追溯ρ_{pred}的前缀路径是否标记

为保留,直到遇到前缀路径被标记为保留为止.假设 G_i 中所有顶点 u 的$\mathcal{B}(u)$集合的路径总数为 m,第 20 行~第 22 行自底向上检查的时间复杂度是 $O(m)$.由引理 2,m 不超过$|E|$.实际上,受出发时刻和路径支配约束,$m<<|E|$.因此,第 20 行~第 22 行不会产生太大耗费.

对于每条标记为保留的路径,第 23 行、第 24 行生成标签 l 并加进相应的标签组.最后,第 26 行调用反向 Dijk-CCMTP 算法计算到达 v_i 的基本路径,计算过程与第 4 行~第 25 行相似.

5.2 选择路径

本节关注算法 3 第 19 行的 MarkPrune 算法如何选择$\mathcal{B}(u)$中的路径加进 ACCTL.$\mathcal{B}(u)$包含了在 t 时刻出发的从 v_i 到 u 的路径.关键问题是:如何在$\mathcal{B}(u)$中选择尽量少的路径加进 ACCTL,同时能够保证对于任意的查询 Q,都能在 ACCTL 中找到标签生成近似解?换句话说,对于任意路径$\rho\in\mathcal{B}(u)$,如果表示ρ的标签没有加进 ACCTL,则在 ACCTL 中存在另一条标签对应路径ρ',使得$\rho'\alpha$-支配ρ.精确计算出保留的路径的最小集合是一个 NP-难问题,MarkPrune 采用贪心的策略选择路径,如算法 4 所示.

算法 4. MarkPrune.

输入:在 t 时刻出发从 v_i 到 u 的基本路径的集合$\mathcal{B}(u)$,$\mathcal{L}_{-}(u)$中表示从 v_i 到 u 的路径的标签组 $L^u_{v_i}$.

输出: $\mathcal{B}(u)$内的标签标记为保留或删除.

1. 假设$\mathcal{B}(u)$内的路径的加入顺序依次是$\rho_0,\rho_1,\ldots,\rho_{k-1}$,初始时对于 $0\leqslant i<k$,设 $d(\rho_i)=\rho_i$;
2. 标记ρ_{k-1}为保留;令 $x=k-1$;
3. **for** $y=k-2,k-3,\ldots,1,0$ **do**
4. **if** $\rho_x\alpha$-支配ρ_y **then** 标记ρ_y为删除;记录 $d(\rho_x)=\rho_y$;
5. **else** 标记ρ_y为保留;令 $x=y$;
6. **foreach** 标记为保留的路径$\rho\in\mathcal{B}(u)$ **do**
7. **if** $L^u_{v_i}$ 内存在标签表示路径ρ',使得$\rho'\alpha$-支配 $d(\rho)$ **then** 标记ρ为删除;
8. 返回;

假设加入$\mathcal{B}(u)$的路径的顺序依次是$\rho_0,\rho_1,\ldots,\rho_{k-1}$.由于路径按费用升序生成,受路径支配的限制,$\rho_0,\rho_1,\ldots,\rho_{k-1}$自动按到达时刻降序排列.因此,$\rho_i$ 仅可能α-支配比它早加入$\mathcal{B}(u)$的路径$\rho_j(j<i)$.MarkPrune 中,令 x 指向最近被标记为保留的路径,初始时令ρ_{k-1}标记为保留,x 指向ρ_{k-1}.接下来,从ρ_{k-2}开始依次往前扫描路径,令 y 指向当前被检查的路径.如果$\rho_x\alpha$-支配ρ_y,则ρ_y 标记为删除;否则,ρ_y 标记为保留.然后,继续检查ρ_y 可否α-支配更早期加入$\mathcal{B}(u)$的路径,令 x 指向ρ_y.

回顾在 BuildIndex 算法中,从 v_i 出发的路径按出发时刻从晚到早的顺序生成,因此当前 $L^u_{v_i}$ 内的标签表示的路径的出发时刻均晚于$\mathcal{B}(u)$内的路径,也可能α-支配$\mathcal{B}(u)$内的某些路径.直观来说,如果在 $L^u_{v_i}$ 内存在某条标签对应的路径$\rho'\alpha$-支配$\mathcal{B}(u)$内的某条路径ρ,则可以把ρ删除.但这种判断会引起错误.假设α=1.1,$\mathcal{B}(u)$中存在两条路径ρ_{10} 和ρ_{11},费用分别是 10 和 11,且有$\rho_{11}\alpha$-支配ρ_{10}.于是保留ρ_{11},删除ρ_{10}.假设 $L^u_{v_i}$ 中存在某条标签表示路径ρ_{12},费用 12,且有$\rho_{12}\alpha$-支配ρ_{11}.如果因为ρ_{12}的存在删除ρ_{11},则没有路径α-支配ρ_{10}.因此,MarkPrune 采用 $d(\rho)$记录ρ的支配范围,即ρ能够α-支配的费用最小的路径是哪条. $L^u_{v_i}$ 中,某条标签表示的路径ρ'能够删除$\mathcal{B}(u)$中的某条路径ρ,仅当$\rho'\alpha$-支配 $d(\rho)$.

定理 2. BuildIndex 算法生成的 ACCTL 索引是正确的.

证明:假设 P_{opt} 是一个从顶点 s 到顶点 d 的查询 Q 的精确解,且 P_{opt} 上任意一段子路都是非支配路径.由引理 1,这样的路径 P_{opt} 是存在的.设 v_T是 P_{opt} 上等级最高的点,P_{opt} 上从 s 到 v_T的路段记为 P^+_{opt}, 从 v_T到 d 的路段记为 P^-_{opt}. 首先证明:在 ACCTL 中能找到标签生成路径 P,使得 $P\alpha$-支配 P_{opt}.BuildIndex 的第 6 行~第 17 行枚举了所有从 v_T出发的基本路径,保证生成了 P^-_{opt}; 第 18 行、第 19 行保证 ACCTL 存在路径 P_{-},$P_{-}\alpha$-支配 P^-_{opt}. 对称地,

ACCTL 也存在路径 P_+,$P_+$$\alpha$-支配 P_{opt}^+ 在费用方面,$c(P_+)+c(P_-)\leqslant\alpha\cdot c(P_{opt}^+)+\alpha\cdot c(P_{opt}^-)=\alpha\cdot c(P_{opt})$.在时间方面,$P_+$的到达时刻$\leqslant P_{opt}^+$ 的到达时刻$\leqslant P_{opt}^-$ 的出发时刻$\leqslant P_-$的出发时刻,所以 P_+与 P_-能够拼接成路径;同时,P_+的出发时刻$\geqslant P_{opt}^+$ 的,P_-的到达时刻$\leqslant P_{opt}^-$ 的,所以 P_+和 P_-拼接的路径α-支配 P_{opt}.最后,第 20 行~第 22 行保证了每条路径的前缀路径的标签都在 ACCTL 中,于是能够将标签正确还原成 G 上的一条路径. □

定理 3. 设查询 Q 的费用限制是θ,P_{opt} 是 Q 的精确解,且 P_{opt} 上任意一段子路都是非支配路径.设 P 是用 ACCTL 查询所得解.

(1) 若$\alpha\cdot c(P_{opt})\leqslant\theta$,则 P 是精确解;

(2) 若$\theta<\alpha\cdot c(P_{opt})$,则 P 有可能是精确解,也可能是近似解;若 P 是近似解,满足:

(2.1) $c(P)\leqslant\alpha\cdot c(P_{opt})$;

(2.2) P 不早于 P_{opt} 出发且不晚于 P_{opt} 到达.

证明:设 P_{opt}^+ 和 P_{opt}^- 的含义如定理 2 中的证明所述.

- 用反证法证明:(1) 当$\alpha\cdot c(P_{opt})\leqslant\theta$时,$P$ 是精确解.因为 P_{opt} 上任意一段子路都是非支配路径,所以 P_{opt}^+ 是非支配路径,于是在 ACCTL 中存在标签 l_+^*,使得 l_+^* 表示的路径α-支配 P_{opt}^+;同理,ACCTL 存在标签 l_-^*,使得 l_-^* 表示的路径α-支配 P_{opt}^-. 于是,l_+^* 和 l_-^* 拼接的路径 P 满足费用限制:$c(P)\leqslant\alpha\cdot c(P_{opt}^+)+\alpha\cdot c(P_{opt}^-)=\alpha\cdot c(P_{opt})\leqslant\theta$.另一方面,由$\alpha$-支配的定义,$P$ 的出发时刻$\geqslant P_{opt}^+$ 的,到达时刻$\leqslant P_{opt}^-$ 的,与 P_{opt} 是最优解矛盾.
- (2) 当$\theta<\alpha\cdot c(P_{opt})$时,如果 ACCTL 中存在的标签 l_+^* 和 l_-^* 拼接的路径 P 满足 $c(P)\leqslant\theta$,则同情形(1)的证明,P 是精确解;若 $c(P)>\theta$,则 P 是近似解,可采用定理 2 的证明过程证明情形(2.1)和情形(2.2). □

定理 4. BuildIndex 算法的时间复杂度是 $O(|V|\cdot\Delta_{\max}\cdot|E|\cdot(\log|E|+\Delta_{\max}))$,其中,$|V|$表示顶点数,$|E|$表示边数,$\Delta_{\max}$表示顶点的最大度数.

5.3 点序o的计算

与文献[14,15,21,22]相同,尽管顶点的等级序 o 的选取不影响查询结果的正确性,但会影响索引所包含的标签数目,进而影响查询效率.要精确计算出 o,使得索引所包含的的标签数最少是一个 NP-难的问题.因此,本文采用文献[14,21]的方法,对 o 进行启发式计算.如果顶点 v 在路径 P 上,则称 v 覆盖 P.ACCTL 的作用好比把一张记录任意查询 Q 的解的数据库表 T 分解成两张表 T_1 和 T_2,使得 T 中的记录可以通过 T_1 和 T_2 的连接操作生成.T_1 好比记录了所有顶点 v 的$\mathcal{L}_+(v)$内的标签,T_2 记录了$\mathcal{L}_-(v)$内的标签.为使 T_1 和 T_2 尽量小,应该使得 T_1 和 T_2 之间能连接上的记录尽量多.这等同于:v 覆盖的带费用限制的最小时态路径越多,v 的等级就应该越高.由于带费用限制的最小时态路径太多,不可能逐一枚举,所以本文采用这样的方法确定 o:从图中随机选取一个顶点集合 $V_{sub}\subset V$,分别从 V_{sub} 的每个顶点 v 出发扩展路径.留意到路径具有时间和费用两个属性,两个点之间有多条不会被相互支配的路径.本文采用近似的方法将路径的时间和属性量化成一个值:一条路径 P 的“长度”等于 P 的费用$\times\beta+P$ 的耗时$\times(1-\beta)$,其中,β是一个[0,1)范围内的实数,在从 v 出发扩展路径之前随机生成.这样,从顶点 v 出发,采用类 Dijkstra 算法搜索全图,可得出 v 到其他顶点的最短路径,生成一棵以 v 为根的 Dijkstra 树.树上任一顶点 u 到 u 的子孙的路径也是最短路径,因此 u 在图中覆盖的最短路数目等于以 u 为根的子树包含的顶点数,包括 u 自身.u 在$|V_{sub}|$棵树覆盖的最短路径的总数视为 u 覆盖最短路径数.算法对$|V_{sub}|$棵树分别做自底向上的扫描,统计出每个顶点覆盖的最短路径数目,从中选出覆盖最短路径数最多的点 v_0 作为等级最高的顶点;然后,在每棵树中删除以 v_0 为根的子树,再更新各个顶点覆盖的最短路径数,在“剩下的”路径中,选择覆盖数目最大的顶点 v_1 作为等级次高的顶点...依此次类推.若$|V_{sub}|$棵树删除完后仍有顶点的 o 序未确定,则按顶点的度数从大到小排序确定.

6 实 验

实验采用 C++语言编写测试代码,测试平台 Windows 10,64 位操作系统,机器配置 Intel(R) Core(TM) i7-8700K CPU,64GB 内存.实验采用文献[21]提供的数据集做测试.数据采集自 GTFS[25],记录了一些地区的公共交通数据.数据集中每条边 e 只记录了出发时刻和到达时刻,实验给 e 生成一个费用值.为了模拟真实环境,一条从 u 到 v 的边的费用取值图中所有从 u 到 v 的边的耗时的平均值.实验数据的特征见表 4,$|V|$表示顶点数,$|E|$表示边数,$\Delta_{\max}$ 表示顶点最大度数,$\bar{\Delta}$ 表示平均顶点度数.下文就 ACCTL 索引的查询时间、索引大小和建立索引的时间进行分析.

Table 4 Characteristics of datasets

表 4 数据集特征

数据集	$\|V\|$	$\|E\|$	$\Delta_{\max}$	$\bar{\Delta}$	数据集	$\|V\|$	$\|E\|$	$\Delta_{\max}$	$\bar{\Delta}$
Austin	2.7K	317.5K	0.79K	118.83	Madrid	4.6K	1 912.2K	2.86K	412.52
SaltLakeCity	6.3K	329.0K	0.57K	52.74	Berlin	12.8K	1 965.8K	6.48K	162.02
Denver	9.5K	708.7K	1K	74.71	Rome	8.8K	2 267.7K	2.93K	259.91
Houston	9.8K	1 111.8K	1.57K	113.01	Toronto	10.8K	3 296.1K	2.14K	305.85
Budapest	5.5K	1 375.2K	2.24K	264.42	Sweden	51.4K	3 926.5K	7.97K	79.26
LosAngeles	15.0K	1 903.2K	1.4K	128.21	–	–	–	–	–

6.1 查询时间

查询沿用文献[21]提供的查询集.每个数据集的查询集内有 10 万个查询.每个查询包括起点 s、终点 d、时刻 t 和 t'、费用上限θ.s 和 d 从顶点集中随机产生,t 和 t'从数据集的时间域随机产生.费用上限θ是在$[\theta_{\min},\theta_{\max}]$范围内的一个随机数:$\theta_{\min}$ 表示不考虑路径上相邻的边之间的时间顺序,从 s 到 d 的最小费用;$\theta_{\max}$ 表示从 s 到 d 的某条随机的边出发的最短耗时路径的费用.带费用限制的最早到达路径查询使用 s,d,t 和θ作为查询参数;带费用限制的最晚出发路径查询使用 s,d,t'和θ;带费用限制的最短耗时路径查询使用 s,d,t,t'和θ.

实验将 ACCTL 与文献[1]提出的一种 Dijkstra 变种算法作对比.因为文献[1]解决的问题与本文的有差别(回顾文献[1]求解不超过费用上限的、路径上相邻的边满足时间顺序约束的、长度最短的路径),因此将文献[1]的 Dijkstra 变种算法修改成解决本文问题的算法,也即第 2 节讨论的 Dijk-CCMTP 的变种算法进行 3 类查询.下文中,对用基于 Dijkstra 方法求解带费用限制的最早到达路径、最晚出发路径和最短耗时路径的方法标记为 Dijk-CCEAP、Dijk-CCLDP 和 Dijk-CCSDP;对采用 ACCTL 索引查询的方法标记为 ACCTL-CCEAP、ACCTL-CCLDP 和 ACCTL-CCSDP.

图 2 显示了基于 Dijkstra 变种算法和基于 ACCTL 索引方法的查询时间对比.实验一共测试了 11 组数据,y 轴以 ms 为单位,用对数的比例显示每组数据的平均查询时间.ACCTL 索引的α取 1 求精确解,即调用 BuildIndex 时不执行第 18 行~第 22 行去掉部分α-支配路径的代码.

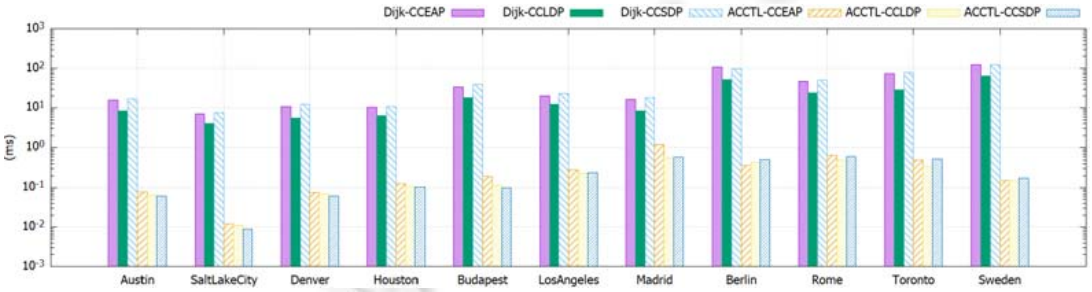
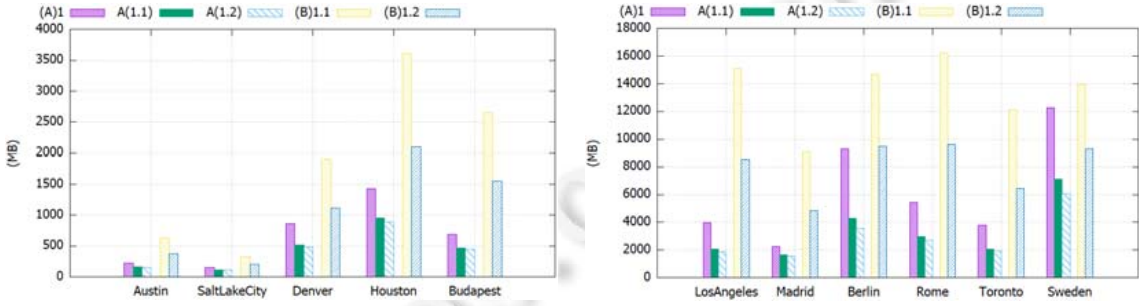


Fig.2 Query times of the Dijkstra variant method and ACCTL

图 2 基于 Dijkstra 搜索的变种算法和 ACCTL 索引的查询时间

总体看来,基于 ACCTL 索引的平均查询时间比基于 Dijkstra 的搜索快 2 个~3 个数量级.这是因为 ACTTL

在预先计算的路径集中查找路径生成解,避免在原图上盲目搜索.就各组数据的基于 ACCTL 索引的查询时间对比,查询时间与索引大小和图的度数有关.总体来说,索引越大,查询时间就越长.图 3 的各组数据的第 1 根柱子显示了α=1 时的索引大小.留意到 Madrid 的索引比 LosAngeles 和 Berlin 的都小,但查询时间长.这是因为 Madrid 的平均顶点度数大.顶点 v 的度数越大,v 到达图中其他顶点(或从图中其他顶点到达 v)的可能性越大,即$\mathcal{L}_+(s)$和$\mathcal{L}_-(d)$有拥有相同端点的标签组越多,所以需要匹配的标签组越多,查询时间越长.此外,最短耗时路径查询的速度比最早到达路径查询的稍慢,甚至在个别数据中最短耗时路径的查询比最早到达路径的更快(例如 Berlin 的 Dijk-CCEAP 与 Dijk-CCSDP、SaltLakeCity 的 ACCTL-CCEAP 与 ACCTL-CCSDP),这是因为最短耗时路径查询受到到达时刻 t'的约束,可以帮助剪枝掉部分搜索;而最早到达路径的到达时刻没有 t'的约束,所以搜索量反而更大.



(a) 前 5 组数据 (b) 后 6 组数据

Fig.3 Index size (MB) of group A and B, with α=1, 1.1, 1.2 respectively

图 3 A 组α取 1、1.1、1.2 和 B 组α取 1.1、1.2 时,各个图的索引大小(MB)

6.2 索引的大小

本节讨论α取值以及路径的费用值浮动对索引大小的影响.实验设置两大组数据集:A 组数据集的设置如上文所述,一条从 u 到 v 的边的费用取值图中所有从 u 到 v 的边的耗时的平均值;B 组数据集的图中一条边 e 的费用取值为[0.7×π,1.3×π]范围内的一个随机整数,其中,π表示通过 e 的耗时,即令边的费用取值多样化.换句话说,A 组的图的边的费用取值单一,B 组的图的边的费用取值多样.

图 3 的每组数据的前 3 根柱子显示了 A 组数据集下α取 1,1.1 和 1.2 时各个图的索引大小,以 MB 为单位,分别标记为(A)1、(A)1.1、(A)1.2.当α递增时,索引变小.因为 ACCTL 利用α-支配去掉部分基本路径,α放得越宽,可以被去掉的基本路径就越多.另一方面,α从 1.1 递增到 1.2 时,索引变小的幅度明显比α从 1 递增到 1.1 时的小.这是因为 ACCTL 需要保证原图中(α=1 时)的每条基本路径 P,在索引中都存在路径α-支配 P,而不是在α=1.1 时保留的路径的基础上去掉α-支配的路径.

图 3 的每组数据的后两根柱子表示 B 组数据集取α=1.1 和α=1.2 的索引大小,分别标记为(B)1.1 和(B)1.2.留意到 B 组的索引明显比 A 组要大.因为 B 组的图中边的费用取值多样,导致两点间路径的费用取值也多样,因此基本路径数也比 A 组要多.在 B 组数据集中,从α=1.1 递增到α=1.2 时,索引大小有 40%~50%的降幅.这是因为路径费用值取值多样,使得α变大后,有更多的路径因为α-支配被去掉.

综合上述比较,索引大小与α取值和图的费用值浮动的大小有关:α越大,索引越小,但随着α的增大,索引大小降幅变慢.另一方面,当图的路径的费用值浮动越大时,α的影响也越大.

6.3 建立索引的时间

图 4 显示了两大组数据建立索引的时间(单位:s).总体来说,A 组的建立索引时间小于 B 组,因为 A 组的基本路径数小于 B 组.两组数据的α=1.1 和α=1.2 的建立索引时间差别不大,可见,建立索引的瓶颈在于生成基本路径,不在于计算α-支配路径.这与第 5.1 节对 BuildIndex 算法的第 18 行~第 22 行的讨论一致.在 A 组数据内,α=1

时建立索引的时间略小于α=1.1 和 1.2,因为α=1 时不需要执行 BuildeIndex 的第 18 行~第 22 行.留意到图 Sweden 在α=1 时的执行时间反而大于α=1.1 和 1.2,这是因为α=1 时基本路径数多,对标签进行排序的时间不可忽略.

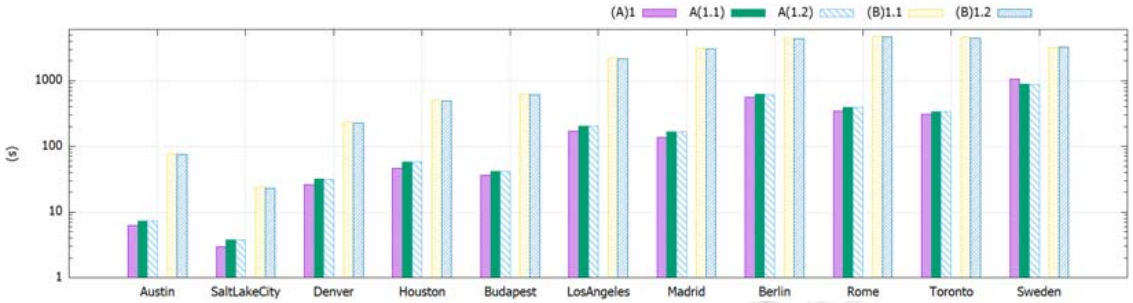


Fig.4 Index construction time of group A and B, with with α=1, 1.1, 1.2 respectively

图 4 A 组α取 1、1.1、1.2 和 B 组α取 1.1、1.2 时,各个图建立索引的时间

7 结束语

本文研究了公交网络下带费用限制的最小时态路径查询问题,提出了一种高效索引结构 ACCTL.ACCTL 通过预计算图中部分路径的信息,使得任意查询可以通过在 ACCTL 中检索路径生成近似解,避免在原图上盲目搜索,从而提高查询效率.本文对 ACCTL 的存储结构、建立索引方法和查询算法做了多方面的讨论和优化.实验验证了 ACCTL 索引支持的查询速度比在原图上采用 Dijkstra 变种算法的查询速度快 2 个~3 个数量级,并分析了 ACCTL 的存储空间和建立索引的时间的影响因素.

References:

[1] Sudip B, Arnab G, Rahul S. Restricted shortest path in temporal graphs. In: Proc. of the DEXA 2015, Part I. LNCS 9261, 2015. 13–27.

[2] Yang YJ, Gao H, Li JZ. Finding the optimal path under time-dependent cost function on graphs. Chinese Journal of Computers, 2012,35(11):2247−2264 (in Chinese with English abstract).

[3] Yang Y, Gao H, Yu JX, Li J. Finding the cost-optimal path with time constraint over time-dependent graphs. Proc. of the VLDB Endowment, 2014,7(9):673–684.

[4] Wang L, Yang LX, Gao ZY. The constrained shortest path problem with stochastic correlated link travel times. European Journal of Operational Research, 2016,255:43−57.

[5] He SX, Fan BQ, Yan L. Improved optimal path searching algorithm in transit network. Journal of University of Shanghai for Science and Technology, 2006,28(1):63−67 (in Chinese with English abstract).

[6] Wei JL, Fan XH, Liu LL, Liu Y, Ren JM, Sun QL. Solution of the optimization model of bus network time-limited free transfer based on DFS-backtracking algorithm. Science Technology and Egineering, 2017,17(10):304−307 (in Chinese with English abstract).

[7] Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-completeness. New York: W.H. Freeman & Co., 1979.

[8] Storandt S. Route planning for bicycles-exact constrained shortest paths mad pratical via contraction hierarchy. In: Proc. of the ICAPS. AAAI Press, 2012. 234−242.

[9] Geisberger R, Sanders P, Schultes D, Delling D. Contraction hierarchies: Faster and simple hierarchical routing in road networks. In: Proc. of the WEA. LNCS 5038, Springer-Verlag, 2008. 319−333.

[10] Lozano L, Medaglia AL. On an exact method for the constrained shortest path problem. Computers and Operations Research, 2013, 40(1):378−384.

[11] Ma TY, An A^* label-setting algorithm for multimodal resource constrained shortest path problem. Procedia-Social and Behavioral Science, 2014,111:330−339.

[12] Tsaggouris G, Zaroliagis CD. Multiobjective optimization: Improved FPTAS for shortest paths and non-linear objectives with applications. Theory of Computer Systems, 2009,45(1):162−186.

[13] Sedeno-NodaA, Alonso-Rodriguez S. An enhanced K-SP algorithm with pruning strategies to solve the constrained shortest path problem. Applied Mathematics and Computation, 2015,265:602−618.

[14] Sibo W, Xiaokui X, Yin Y, Wenqing L. Effective indexing for approximate constrained shortest path queries on large road networks. Proc. of the VLDB Endowment (PVLDB), 2016,10(2):61−72.

[15] Abraham I, Delling D, Goldberg AV, Werneck RF. Hierarchical hub labelings for shortest paths. In: Proc. of the European Conf. on Algorithms. Springer-Verlag, 2012. 24−35.

[16] Cooke KL, Halsey E. The shortest route through a network with time-dependent intermodal transit times. Journal of Mathematical Analysis and Applications, 1966,14(3):493−498.

[17] Geisberger R. Contraction of timetable networks with realistic transfers. In: Proc. of the Int'l Symp. on Experimental Algorithms (SEA 2010). LNCS 6049, Springer-Verlag, 2010. 71−82.

[18] Dibbelt J, Pajor T, Strasser B, Wanger D. Intriguingly simple and fast transit routing. In: Proc. of the Int'l Symp. on Experimental Algorithms (SEA 2013). LNCS 7933, Springer-Verlag, 2013. 43−54.

[19] Wu H, Cheng J, Huang S, Ke Y, Lu Y, Xu Y. Path problems in temporal graphs. Proc. of the VLDB Endowment (PVLDB), 2014,7(9):721−732.

[20] Wu H, Cheng J, Ke Y, Huang S, Huang Y, Wu H. Efficient algorithms for temporal path computation. IEEE Trans. on Knowledge and Data Engineering, 2016,28(11):2927−2924.

[21] Wang SB, Lin WQ, Yang Y, Xiao XK, Zhou SG. Efficient route planning on public transportation networks: A labelling approach. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD). New York: ACM, 2015. 967−982.

[22] Delling D, Dibbelt J, Pajor T, Wernect RF. Public transit labelling. In: Proc. of the Experimental Algorithms. Springer Int'l Publishing, 2015. 273−285.

[23] Bast H, Delling D, Goldberg AV, Hannemnn MM, Pajor T, Sanders P, Wanger D. Route planning in transportation networks. In: Kliemann L, Sanders P, eds. Proc. of the Algorithm Engineering. LNCS 9220, Cham: Springer-Verlag, 2016.

[24] Foschini L, Hershberger J, Suri S. On the complexity of time-dependent shortest paths. Algorithmica, 2014,68(4):1075−1097.

[25] GTFS. 2017. https://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds

附中文参考文献:

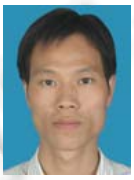
[2] 杨雅君,高宏,李建中.时间依赖函数下的最优路径查询问题研究.计算机学报,2012,35(11):2247−2264.

[5] 何胜学,范炳全,严凌.公交网络最优路径的一种改进求解算法.上海理工大学学报,2006,28(1):63−67.

[6] 魏金丽,范鑫贺,刘莲莲,刘阳,任杰睦,孙启龙.基于深度优先遍历算法-回溯算法的公交网络限时免费换乘优化模型求解.科学技术与工程,2017,17(10):304−307.

马慧(1981－),女,广东中山人,博士,副教授,CCF 专业会员,主要研究领域为数据库管理,图数据库与查询分析,算法设计.

汤庸(1961－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为学术社交网络,教育大数据服务.



梁瑞仕(1982－),男,博士,副教授,CCF 专业会员,主要研究领域为智能规划及其应用,大数据.