DOI:10.7544/issn1000-1239.20210893
59(2):362-375,2022

时态图最短路径查询方法

张天明[1] 徐一恒[1] 蔡鑫伟[2] 范 菁[1]
[1](浙江工业大学计算机科学与技术学院 杭州 310023)
[2](浙江大学计算机科学与技术学院 杭州 310013)
(tmzhang@zjut.edu.cn)

A Shortest Path Query Method over Temporal Graphs

Zhang Tianming[1], Xu Yiheng[1], Cai Xinwei[2], and Fan Jing[1]
[1](*College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou* 310023)
[2](*College of Computer Science and Technology, Zhejiang University, Hangzhou* 310013)

Abstract Shortest path query has been extensively studied for decades of years. However, most of existing works focus on shortest path query over general graphs, a paucity of studies aim at temporal graphs, where there are multi-edges between two nodes and each edge is associated with a temporal interval, recording the start time and the end time of an event. Shortest path query over temporal graphs has a plethora of applications in urban traffic route planning, social network analysis, and communication network mining, to name but a few. Traditional shortest path algorithms on general graphs are not suitable for temporal graphs because subpaths of a shortest temporal path are not guaranteed to be the optimal substructures. Hence, in this paper, we propose a Compressed Transformed Graph tree (CTG-tree) based query method, which consists of a preprocessing stage and an online query stage. In the preprocessing stage, we transform the temporal graph into a general graph, propose a lossless compression method to reduce the scale of the transformed graph, use hierarchical partitioning technique to divide the compressed directed graph into subgraphs, and then build a CTG-tree index based on the partitioned subgraphs. The nodes of CTG-tree maintain shortest paths between some vertices in the corresponding subgraphs, save shortest paths between boundaries in the subgraphs of children nodes, and record shortest paths between boundaries of the subgraphs corresponding to the children nodes and those corresponding to the current node. In the online query stage, based on the constructed CTG-tree index, we develop an efficient shortest temporal path query method. Using four real-life temporal graphs, we experimentally demonstrate that our proposed method has the best query performance compared with the state-of-the-art methods.

Key words shortest path; temporal graph; compressed directed graph; tree index; query method

摘 要 最短路径查询问题已被研究多年,然而,目前已有大部分工作主要集中在普通图上,针对时态图最短路径查询的研究工作相对较少.时态图中,2个顶点之间有多条边,每条边附带有时态区间,记录着边上代表事件的发生时间和结束时间.时态图最短路径查询在城市交通路径规划、社交网络分析、通信

收稿日期:2021-08-31;**修回日期**:2021-11-05
基金项目:国家重点研发计划项目(2018YFB1402802)
This work was supported by the National Key Research and Development Program of China (2018YFB1402802).
通信作者:范菁(fanjing@zjut.edu.cn)

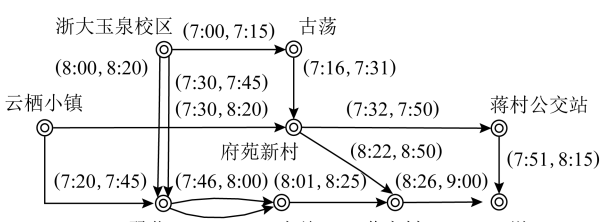
网络挖掘等领域有着广泛的应用.由于最短时态路径的子路径不能保证是最优子结构,传统的普通图最短路径计算方法不再适用于时态图.因此提出了基于压缩转化图树(CTG-tree)索引的查询方法,该方法包含预处理和在线查询2个阶段.预处理阶段将时态图转化为普通图,提出了一种无损压缩方法将转化图压缩以减小图规模,采用层次划分技术将压缩有向图分解为若干个子图,并基于子图建立CTG-tree索引.CTG-tree中的节点保存相应子图内部分顶点之间的最短路径、孩子节点对应子图的边界点之间的最短路径、孩子节点对应子图的边界点与当前节点相应子图的边界点之间的最短路径信息.在线查询阶段基于构建的CTG-tree索引,提出了一种高效的最短路径查询方法.基于4个真实的时态图数据集实验结果表明,与现有方法相比,提出的方法具有更优的查询性能.

关键词 最短路径;时态图;压缩有向图;树索引;查询方法

中图法分类号 TP311.131

图,作为一种通用的数据结构,用于表示各种对象之间的复杂关联关系.图上最短路径计算作为基础功能函数,有着广泛的应用.例如,路径规划[1]、城市交通路线优化[2]、社交网络分析[3]等.现有的最短路径算法主要聚焦在普通图,然而,真实场景中,图中常常附有时态信息,2个顶点之间的边关联的事件在某一时间点发生并持续一段时间结束,此种类型的图称之为时态图.

图1列举了时态图应用在公交时刻图的例子.图中顶点表示公交站点,时态边上附带有时态区间,表示公交车从一个站点到另一个站点的出发时间和到达时间.例如,一班89路公交车早上7:00从浙大玉泉校区始发站出发,7:15到达古荡站点,停靠站1 min,7:16出发,经府苑新村,蒋村公交站,最终在8:15到达三墩终点站.此例中,时态图最短路径查询有助于乘客查询指定时态区间内的2个站点间的最短路径,从而为乘客智能推荐公交线路.给定时态图$G_T=(V_T,E_T)$,查询源点u,查询目的点v,时态区间I,本文研究的时态图最短路径旨在图G_T中查询u到v在时态区间I内的最短时态路径距离.时



线路:
89路:浙大玉泉校区→古荡→府苑新村→蒋村公交站→三墩
79路:浙大玉泉校区→翠苑→大关
121路:云栖小镇→府苑新村
70路:府苑新村→董家村
12专线:云栖小镇→翠苑→大关→董家村→三墩

Fig. 1 Illustration of bus timetable

图1 公交时刻图示例

态图G_T中的边除了关联时态区间外,还附加有权重值,在交通路网中可以表示距离;在公交/地铁/航班时刻图中可以表示费用;在社交网络中可以表示用户之间的亲密度.与已有的最短路径查询问题相比,最短时态路径更具挑战性.

首先,计算最短时态路径时需要考虑边与边的时序流转特性,否则如果运用现有的普通图最短路径计算方法,会产生错误的结果.例如,图1中,假设边上权值均为1,乘客想要查询云栖小镇站到三墩站在时态区间[7:00,9:00]的时态最短路径距离.如果不考虑边之间的时序流转特性,则计算得到的结果为3,对应的最短时态路径为(云栖小镇,府苑新村,蒋村公交站,三墩)或(云栖小镇,府苑新村,董家村,三墩).实际这2条路径均不能到达三墩站.对于第一条路径,121路云栖小镇到府苑新村时间为8:20,其不能赶上7:32从府苑新村出发经由蒋村公交站最终到达三墩站的89路公交车.对于第二条路径,121路换乘70路8:50到董家村,同样赶不上8:26到三墩站的12专线.鉴于此,时态最短路径计算过程中需要考虑时序流转特性,此例计算得到的结果应为4,对应的最短时态路径应为(云栖小镇,翠苑,大关,董家村,三墩).

其次,最短时态路径并不满足子路径最优性质,即顶点u到顶点v的最短时态路径的子路径并不一定是最短时态路径.上例中,子路径(云栖小镇,翠苑,大关,董家村)并不是时态区间[7:00,9:00]内的最短时态路径,云栖小镇到董家村在时态区间[7:00,9:00]内最短时态路径应为(云栖小镇,府苑新村,董家村).这导致计算时态最短路径时更复杂,因为需要考虑所有的路径.

目前,已有一些工作[4-5]研究时态图最短路径查询.然而,它们一般假设输入时态图为FIFO(first-

in-first-out)时间依赖图,FIFO 时间依赖图是指图输入边上的时间间隔表示为出发时间的函数,这个函数具有 FIFO 属性,即若出发时间早,则到达时间也早.这类问题具有一定的特殊性,实际交通工具出行时刻图或社交接触网络不一定是 FIFO 时间依赖图.与本文工作最相关的是文献[6-7]提出的 one-pass 算法,其将时态图转化为普通图,而后在普通图上采用广度优先搜索和 Dijkstra 算法计算最短时态路径,但其在较大规模数据集上效率较低.鉴于此,本文研究时态图最短路径问题并提出了基于层次索引的计算方法,其主要包含预处理和在线查询 2 个阶段.预处理阶段本文提出了一种无损压缩方法和层次索引 CTG-tree(compressed transformed graph tree).首先将时态图转化为普通图,对普通图进行压缩以减小规模,将时态图上最短路径计算转化为在压缩结果图上执行.而后在压缩结果图上构建 CTG-tree,它将 G-tree[8] 的思想扩展到压缩有向图上,首先采用 Metis 划分将压缩图层次划分为若干个子图,得到每个子图的入边界点与出边界点集合.CTG-tree 为每个叶子节点对应子图计算其内部顶点与出边界点的最短路径、入边界点到子图内顶点的最短路径和出边界点到入边界点的最短路径;非叶节点计算其对应子图入边界点到其所有孩子节点对应子图的入边界点之间的最短路径距离、所有孩子节点对应子图的出边界点到当前非叶节点对应子图的出边界点之间的最短路径距离、以及所有孩子节点对应子图的出边界点到所有孩子节点对应子图的入边界点之间的最短路径距离作为索引.查询阶段本文基于层次索引 CTG-tree 设计了高效的最短路径查询算法.概括而言,本文的主要贡献有 4 个方面:

1) 提出了一种基于层次索引的两阶段(预处理阶段和查询阶段)方法以高效地计算时态图最短路径;

2) 提出了一种无损压缩方法和层次索引 CTG-tree,将时态图上最短路径计算结果转化为在压缩结果图上执行.CTG-tree 将 G-tree 的思想扩展到压缩有向图上;无损压缩在减小图处理规模的同时还保证了压缩结果图上最短路径计算的准确性;

3) 提出了基于 CTG-tree 索引的查询算法以高效地支持时态图最短路径查询;

4) 基于 4 个真实的时态图数据集,进行了充分的实验评估,验证了本文提出的基于层次索引的最短时态路径算法的高效性.

1 相关工作

本节分别概述已有的普通图和时态图最短路径查询算法.

1.1 普通图最短路径查询算法

综述文献[1,9]将普通图最短路径问题划分为点到点的最短路径计算、单源最短路径计算和所有点对最短路径计算 3 类问题.点到点的最短路径计算指定点对之间的最短路径;单源最短路径计算一个点到其他所有点的最短路径距离;所有点对最短路径计算图中所有点对之间的最短路径距离,其可通过单源最短路径计算得到.已有的最短路径查询算法大多是基于经典的 Dijkstra 方法,文献[10]从理论角度分析了单源最短路径基于堆的优先级队列算法的时间复杂度是 $O(m \log n)$;采用斐波那契堆的时间复杂度是 $O(m+n \log n)$.文献[11]提出了基于 Dijkstra 的变体方法 Sky-Dijk.为了缩小搜索空间,双向搜索算法[12-13] 被提出.由于 Dijkstra 方法及其变体算法在大图上运行时间较长,因此研究人员提出了一系列基于索引的最短路径算法[8,14-20] 以提高查询效率.例如,文献[14]提出了基于标志点(landmark)的算法,其预先选择一些顶点作为标志点;并且预先计算图中每个顶点与标志点之间的最短路径距离.图中任意一对顶点之间的最短路径距离上下界可以运用预先计算好的距离计算得到;文献[15]提出了基于剪枝的标志点标签,通过剪枝的广度优先搜索预计算所有顶点的距离标签,通过距离标签计算任意点对的最短路径距离.文献[16]提出了收缩层级结构;将图中的顶点或者边按照重要度排序,再根据排序的结果迭代,最终生成层级嵌套索引.文献[8]提出了 G-tree 索引,其首先将路网层次划分为若干个子图,再计算每个子图内的顶点与边界点之间的距离,子图间的边界点与边界点之间的距离作为索引,基于 G-tree 索引提高查询效率.基于索引的最短路径算法关键在于平衡查询时间、索引时间、索引空间;为了减小索引空间,文献[17-18]还提出了索引压缩方法.

针对动态图,文献[19]提出了 DCHvcs,其扩展收缩层级结构以支持动态图.文献[20]在收缩层级最短路径索引的基础上,构建辅助的图结构以及权重传播机制来处理流式和批量的路网权重更新.文献[21]提出了 CRP 方法以支持最短路径索引的动态更新.针对大规模输入图,文献[22]基于分布式流

处理系统 Yahoo S4,提出了 2 种异步通信算法以支持动态最短路径;文献[23]探索了边受限的最短路径查询问题.此外,一些研究人员还致力于近似最短路径计算方法研究[24-25].文献[24]提出了距离 Oracle 数据结构,对于$(2k-1)$-近似能在$O(k)$的时间内给出任意节点对之间的近似最短路径.文献[25]将大图近似为一个规模相对小的 spanner 稀疏子图,而后在子图上做近似最短路径计算.概括而言,上述算法均针对普通图,并没有考虑边上附带的时态信息,因此不能直接有效地应用在时态图上.

1.2 时态图最短路径查询算法

综述文献[1]总结了时间依赖图的最短路径查询算法.时间依赖图中边延迟函数计算一条边中源点到目的点所需时间,时间函数可分为离散或连续的.

文献[2,6-7,26]针对离散时间下的最短路径进行了研究.文献[2]提出了 TD-G-tree 索引以支持时间依赖路网上最短路径查询,但是其与本文研究问题不同,并没有考虑路径内边之间的时序关系.文献[26]提出了时间表标签(time table labelling, TTL)索引;文献[6-7,27]提出将时态图转化为普通图,基于此,文献[27]研究了时间依赖的可达性查询,其提出了 TopChain 标签索引方法以解决时态图上的可达性查询、最早到达时间查询、最小间隔时间查询.与其研究的问题不同,本文旨在解决时态图上的最短路径查询.文献[6-7]在转化的普通图上采用广度优先搜索和 Dijkstra 算法计算 2 点之间给定时间段内的最短路径.本文基于上述工作,将时态图转化为普通图后,利用压缩以及层次索引技术高效支持最短时态路径查询.

文献[3-5]针对连续时间下的最短路径进行了研究.文献[4]提出了基于 Dijkstra 算法返回耗时最少旅行时间的最优出发时间;文献[3]首先定位给定时态区间段内子图,而后在子图中采用 Dijkstra 计算最短路径.文献[5]在文献[4]的基础上,研究的时态图中边除了边延迟函数外,还添加了权重代价函数,利用这 2 种函数的结构属性设计最短路径算法.这类工作通常假设输入时态图为 FIFO 时间依赖图,输入边上时间间隔表示为出发时间的函数,出发时间早,则到达时间也早,这类问题具有一定的特殊性.而在我们的问题中,输入边有出发时间和到达时间,输入图(例如交通工具出行时刻图或者社交接触网络)不一定是 FIFO 时间依赖图,这导致计算时态最短路径时更复杂,最短路径计算不满足最优子结构特征,因为需要考虑边与边的时序流转特性,计算难度更高.

2 问题定义

本节主要介绍时态图、时态路径相关概念,最后给出问题定义.

定义 1. 时态图[6].本文定义的时态图是复杂有向图,表示为$G_T=(V_T,E_T)$,其中V_T表示顶点集合;$E_T\subseteq V_T\times V_T$是有向边的集合,具体地,连接顶点$u\in V_T$到$v\in V_T/\{u\}$的有向边$e_i\in E_T$表示为一个五元组$(u,v,w_i,s_{ti},a_{ti})$,表示$u$与$v$之间的事件发生起始时间是$s_{ti}$,终止时间是$a_{ti}$.时间间隔是$I=(s_{ti},a_{ti})$,持续时间为$|I|=a_{ti}-s_{ti}$,$w_i$表示每条时态边$e_i$的非负权重值,表示耗费时间或者路程等.

不同于简单图,时态图中 2 个顶点之间可能有多条时态边.与普通图路径定义不同,时态图中时态路径的定义如下:

定义 2. 时态路径.从顶点u到v的时态路径p,表示为$p(u,v)=\langle u,e_1,v_1,\cdots,v_{m-1},e_m,v\rangle$,其中$\langle u,v_1,v_2,\cdots,v_{m-1},v\rangle$表示顶点序列,$\langle e_1,e_2,\cdots,e_{m-1},e_m\rangle$表示时态边序列,$e_1=(u,v_1,w_1,s_{t1},a_{t1})$,$e_i=(v_{i-1},v_i,w_i,s_{ti},a_{ti})(1<i<m)$,$e_m=(v_{m-1},v,w_m,s_{tm},a_{tm})$,使得对于任意的$1\leqslant i\leqslant m$,$a_{ti}\leqslant s_{ti+1}$.令$S_p=s_{t1}$表示$p$的开始时间,$E_p=a_{tm}$表示$p$的到达时间,$D_p=a_{tm}-s_{t1}$表示$p$的时间间隔.$W_p=\sum_{i=1}^{m}w_i$表示$p$的路径权重.

给定时态图G_T,源点s,目的点s',时间间隔$I=[t_s,t_a]$,定义函数$TPSet_G(s,s',I=[t_s,t_a])$为从顶点$s$到顶点$s'$的所有满足$S_p\geqslant t_s$,$E_p\leqslant t_a$的$G_T$中的时态路径$p$构成的集合.基于此给出定义 3.

定义 3. 最短时态路径查询.给定时态图G_T,源点s,目的点s',时间间隔$I=[t_s,t_a]$,最短时态路径查询返回s在时间间隔I内到达s'的时态路径权重最小值$d(s,s',I)=\min_{p\in TPSet_G(s,s',I)}W_p$.

本文定义的最短时态路径查询问题可用于城市交通路线查询与推荐.例如,给定图 1 所示的公交时刻图G_T,源点s设置为浙大玉泉校区站,目的点s'设置为三墩站,时态区间$I=[7:00,9:00]$,则根据定义 1,定义 2 和$TPSet_G$函数定义得到$TPSet_G$(浙大玉泉校区站,三墩站,[7:00,9:00])={⟨浙大玉泉校区,(浙大玉泉校区,翠苑,1,7:30,7:45),翠苑,(翠苑,大关,1,7:46,8:00),大关,(大关,董家村,1,

8:01,8:25),董家村,(董家村,三墩,1,8:26,9:00),三墩〉,〈浙大玉泉校区,(浙大玉泉校区,古荡,1,7:00,7:15),古荡,(古荡,府苑新村,1,7:16,7:31),府苑新村,(府苑新村,蒋村公交站,1,7:32,7:50),蒋村公交站,(蒋村公交站,三墩,1,7:51,8:15),三墩〉}.对应2条线路:分别是79路—121专线;89路.根据定义3,浙大玉泉校区站到三墩站的最短时态路径距离为4.

3 CTG-Tree构建

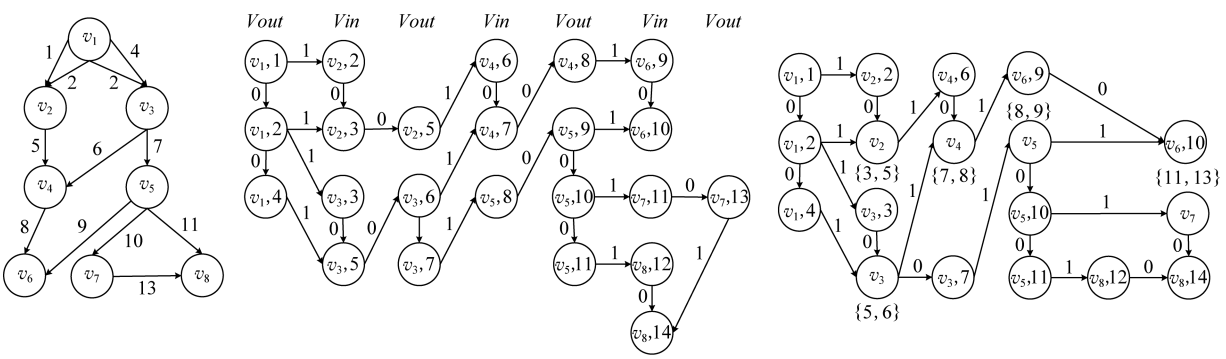
本文提出了基于层次索引的计算方法,主要包含预处理和在线查询2个阶段.本节详细阐述预处理阶段提出的无损压缩方法和层次索引CTG-tree.

3.1 基于转化图的无损压缩

在时态图上计算最短时态路径的直接方法是进行广度优先搜索或者深度优先搜索,但是由于时态最短路径不满足子路径最优性质,因此在搜索过程中需要保存计算遍历到的所有路径,此种方法效率较低.因此本文的主要思想是先将时态图转化为普通图,而后进行压缩,再利用索引加速查询.具体地,本文利用文献[6]提出的方法将时态图 $G_T=(V_T,E_T)$ 转化为普通图 $G=(V,E)$,其中转化图 G 已被证明为有向无环图[6],具体的转化过程如下.

每个顶点 $u\in V_T$ 转化为 V 中的2个集合 $Vin(u)$ 和 $Vout(u)$,$Vin(u)=\{\langle u,a_{ti}\rangle|1\leqslant i\leqslant h\}$,$Vout(u)=\{\langle u,s_{tj}\rangle|1\leqslant j\leqslant m\}$,其中 a_{ti} 是 u 入边中到达 u 的不同时间实例,s_{tj} 是 u 出边中从 u 出发的不同时间实例.h 和 m 分别表示不同的到达时间实例数目和出发时间实例数目.边转化过程涉及到3步.1)每条边 $e_i=(u,v,w_i,s_t,a_t)$ 转化为 $\langle u,s_t\rangle\in Vout(u)$ 到顶点 $\langle v,a_t\rangle\in Vin(v)$ 的边,边的权重为 w_i.2)令 $Vin(v)=\{\langle v,a_{t1}\rangle,\langle v,a_{t2}\rangle,\cdots,\langle v,a_{th}\rangle\}$,对于 $1\leqslant i\leqslant h-1$,创建从 $\langle v,a_{ti}\rangle$ 到 $\langle v,a_{ti+1}\rangle$ 的边,边的权重为0.同样的,对于 $Vout(v)$ 集合,采用相同的方式创建边.3)对于顶点 $\langle v,a_{tin}\rangle\in Vin(v)$,按照 $Vin(v)$ 中顶点的逆序顺序,创建从 $\langle v,a_{tin}\rangle$ 到 $\langle v,s_{tout}\rangle\in Vout(v)$ 的边,边的权重为0.其中 $s_{tout}=\min\{s_{t'}|\langle v,s_{t'}\rangle\in Vout(v),s_{t'}\geqslant a_{tin}\}$,并且没有 $\langle v,a_{tin}\rangle\in Vin(v)$ 到 $\langle v,s_{tout}\rangle$ 的边创建.

图2给出了转化图示例.图2(a)为输入的时态图 G_T,边上附带的值为边事件的出发时间,假设边权重、边时态区间间隔均为1.图2(b)为转化后的有向图 G.以图2(a)中的顶点 v_3 为例,v_3 转化为 $Vin(v_3)=\{\langle v_3,3\rangle,\langle v_3,5\rangle\}$,$Vout(v_3)=\{\langle v_3,6\rangle,\langle v_3,7\rangle\}$.以图2(a)中的边 $(v_3,v_4,6,7)$ 为例,其转化为 $\langle v_3,6\rangle\in Vout(v_3)$ 到 $\langle v_4,7\rangle\in Vin(v_4)$ 的边.对于 $Vin(v_3)$,创建 $\langle v_3,3\rangle$ 到 $\langle v_3,5\rangle$ 的边.对于 $Vout(v_3)$,创建 $\langle v_3,6\rangle$ 到 $\langle v_3,7\rangle$ 的边.此外,需要创建 $\langle v_3,5\rangle\in Vin(v_3)$ 到 $\langle v_3,6\rangle\in Vout(v_3)$ 的边.



(a) 时态图G_T (b) 转化图G (c) 压缩图G'

Fig. 2 Illustrations of temporal graph, transformed graph, and compressed graph

图2 时态图、转化图和压缩图示例

转化图的规模通常为时态图的几十倍甚至上百倍,如果根据文献[6-7],直接在转化图上采用广度优先搜索或Dijkstra算法计算最短路径,则搜索空间较大,耗费时间.可以观察到,转化图规模庞大的原因是时态图中的一个顶点拆分为 Vin,$Vout$ 集合中的多个顶点.Vin,$Vout$ 集合中的顶点连接的边权重均为0,并且一些顶点对具有相同的邻居,鉴于此,本文提出了一种无损压缩规则:原始时态图中的顶点 v 转化的 Vin 或 $Vout$ 集合中的顶点 $\langle v,t_1\rangle$ 与 $\langle v,t_2\rangle$,$t_1<t_2$,如果其满足 $\langle v,t_1\rangle$ 与 $\langle v,t_2\rangle$ 的入度邻居和出度邻居除 $\langle v,t_1\rangle$ 与 $\langle v,t_2\rangle$ 外相同,则顶点 $\langle v,t_1\rangle$ 与 $\langle v,t_2\rangle$ 压缩为 $\langle v,T\rangle$,$T=\{t_1,t_2\}$.

图 2(c)给出了图 2(b)按照上述无损压缩规则得到的压缩图 G'.转化图 G 中的$\langle v_4,7\rangle\in Vin(v_4)$与$\langle v_4,8\rangle\in Vout(v_4)$压缩为$\langle v_4,\{7,8\}\rangle\in G'$;$G$ 中的$\langle v_3,6\rangle\in Vout(v_3)$与$\langle v_3,7\rangle\in Vout(v_3)$压缩为$\langle v_3,\{6,7\}\rangle\in G'$;$G$ 中的$\langle v_5,8\rangle\in Vin(v_5)$与$\langle v_5,9\rangle\in Vout(v_5)$压缩为$\langle v_5,\{8,9\}\rangle\in G'$.

具体的压缩算法伪码如算法 1 所示.其中,$N_{in}(\langle v,t\rangle)=\{\langle u,t'\rangle\mid(\langle u,t'\rangle,\langle v,t\rangle)\in E\}$和$N_{out}(\langle v,t\rangle)=\{\langle m,t'\rangle\mid(\langle v,t\rangle,\langle m,t'\rangle)\in E\}$分别表示转化图 G 中顶点$\langle v,t\rangle$的入度邻居和出度邻居集合.算法 1 首先遍历转化图 G 中的 $Vin(v)$或 $Vout(v)$集合中的所有顶点(行①),如果集合中的任意 2 个顶点$\langle v,t_1\rangle$和$\langle v,t_2\rangle$满足上述无损压缩规则,则$\langle v,t_1\rangle$与$\langle v,t_2\rangle$合并为$\langle v,\{t_1,t_2\}\rangle$(行②~④);如此得到最终的无损压缩图 G'.

算法 1. 基于转化图的无损压缩算法.

输入:基于时态图 $G_T=(V_T,E_T)$的转化图 $G=(V,E)$;

输出:无损压缩图 $G'=(V',E')$.

① for each $v\in G_T$,$\langle v,t_1\rangle,\langle v,t_2\rangle\in Vin(v)\cup Vout(v)$

② if $N_{in}(\langle v,t_1\rangle)/\langle v,t_2\rangle=N_{in}(\langle v,t_2\rangle)/\langle v,t_1\rangle$且 $N_{out}(\langle v,t_1\rangle)/\langle v,t_2\rangle=N_{out}(\langle v,t_2\rangle)/\langle v,t_1\rangle$

③ $\langle v,t_1\rangle$与$\langle v,t_2\rangle$合并为$\langle v,\{t_1,t_2\}\rangle$;/* 根据规则进行压缩 */

④ end if

⑤ end for

⑥ 输出无损压缩图 $G'=(V',E')$.

明显地,算法 1 的时间复杂度为 $\sum_{(v,t)\in V}(|Vin(v)|+|Vout(v)|)^2(|N_{in}(v,t)|+|N_{out}(v,t)|)$,空间复杂度为 $O(|V|+|E|)$.

3.2 CTG-tree 索引

基于上述无损压缩图,构建层次索引 CTG-tree.分为 2 个步骤:

1) 图层次划分.由于 Metis 划分保证每次划分的子图顶点数目相近,交叉边数目较少,因此本文采用 Metis 层次划分,首先将无损压缩图 G'划分为若干子图 $G_1,G_2,\cdots,G_l(l\geqslant 2)$,而后在每个子图 $G_i(1\leqslant i\leqslant l)$上进行迭代划分直至划分的子图顶点数目不超过指定的阈值 θ;最终形成平衡树 CTG-tree.G'为 CTG-tree 的根节点,第 i 次迭代划分的子图为 CTG-tree 第 i 层节点,叶子节点对应子图的顶点数目不超过阈值 θ. Metis 划分属于边划分方法,每次迭代划分的子图之间没有交集(例 $G_1\cup G_2\cup\cdots\cup G_l=G'$,$G_1\cap G_2\cap\cdots\cap G_l=\varnothing$),但是会产生跨子图的交叉边 $e(u,v)$,$u\in G_i$,$v\in G_j$.基于此,本文将子图内顶点分为 3 类:入边界点、出边界点和内部顶点.令 $G_i.B_i$,$G_i.B_o$,$G_i.V_I$ 分别表示子图 $G_i=(V_i,E_i)$的入边界点、出边界点和内部顶点的集合,则

① $V_i=G_i.B_i\cup G_i.B_o\cup G_i.V_I$;

② $\forall u\in G_i.B_o$,至少存在一条跨分区的交叉边 $e=(u,m)$,u 与 m 隶属不同子图,即 $m\notin G_i$;

③ $\forall v\in G_i.B_i$,至少存在一条跨分区的交叉边 $e=(s,v)$,s 与 v 隶属不同子图,即 $s\notin G_i$;

④ $\forall v'\in G_i.V_I$,v'的所有入度邻居.出度邻居都隶属子图 G_i,即 $N_{in}(v')\subseteq V_i$,$N_{out}(v')\subseteq V_i$;

在图划分步骤中,每个子图记录相应的入边界点,出边界点.

2) 最短路径距离矩阵计算.对于 CTG-tree 中的叶子节点对应的子图 G_i,需要计算 3 个最短路径距离矩阵,即游出距离矩阵 $G_i.\boldsymbol{D}_o$、游入距离矩阵 $G_i.\boldsymbol{D}_i$ 和边界点距离矩阵 $G_i.\boldsymbol{D}_b$. $G_i.\boldsymbol{D}_o$ 保存 G_i 中所有顶点到每个出边界点的最短路径距离;$G_i.\boldsymbol{D}_i$ 保存 G_i 中每个入边界点到所有顶点的最短路径距离;$G_i.\boldsymbol{D}_b$ 保存 G_i 中所有出边界点到入边界点的最短路径距离.对于 CTG-tree 中的非叶子节点对应的子图 G_j,需要计算出边界点游入距离矩阵 $G_j.\boldsymbol{D}_{Bi}$、入边界点游入距离矩阵 $G_j.\boldsymbol{D}_{Ii}$和边界点游出距离矩阵 $G_j.\boldsymbol{D}_{Bo}$;$G_j.\boldsymbol{D}_{Bi}$保存 G_j 所有孩子节点对应子图的出边界点到 G_j 所有孩子节点对应子图的入边界点之间的最短路径距离;$G_j.\boldsymbol{D}_{Ii}$保存 G_j 入边界点到 G_j 所有孩子节点对应子图的入边界点之间的最短路径距离;$G_j.\boldsymbol{D}_{Bo}$保存 G_j 所有孩子节点对应子图的出边界点到 G_j 出边界点之间的最短路径距离.

具体的 CTG-tree 索引构建算法伪码如算法 2 所示.

算法 2. CTG-tree 索引构建算法.

输入:无损压缩图 $G'=(V',E')$,Metis 每层划分的子图数 l,叶子子图顶点数阈值 θ;

输出:CTG-tree 索引.

① CTG-tree←$Metis(G',l,\theta)$;

② for each layer(自底向上) of CTG-tree

③ if leaf node← /* 叶子节点层计算 */

④ $G_i.\boldsymbol{D}_o,G_i.\boldsymbol{D}_i,G_i.\boldsymbol{D}_b\leftarrow Dijkstra(G')$;

⑤ else /* 非叶子节点层计算 */

⑥ $G_j.\boldsymbol{D}_{\mathrm{Bi}} \leftarrow Dijkstra(G')$; $G_j.\boldsymbol{D}_{\mathrm{Ii}} \leftarrow Dijkstra(G')$; $G_j.\boldsymbol{D}_{\mathrm{Bo}} \leftarrow Dijkstra(G')$;

⑦ end if

⑧ end for

⑨ 输出 CTG-tree 索引.

算法 2 首先在给定的无损压缩图 G' 上进行 Metis 层次划分,得到 CTG-tree(行①),需要说明的是,无损压缩图中,边权重为 0 会影响 Metis 划分过程,因此需要适当增权处理.

接着,对 CTG-tree 自底向上遍历计算索引矩阵(行②).对于叶子节点,利用 Dijkstra 算法在图 G' 上计算叶子节点对应子图 G_i 的游出距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{o}}$、游入距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{i}}$ 和边界点距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{b}}$(行③~④).对于非叶节点,利用 Dijkstra 算法在图 G' 上计算对应子图 G_j 的出边界点游入距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Bi}}$、边界点游出距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Bo}}$ 和入边界点游入距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Ii}}$,值得注意的是,此过程可以利用顶点拓扑序加速计算过程,由于转化图是有向无环图,压缩图即为有向无环图,因此在运用 Dijkstra 算法前,可先计算顶点拓扑序,若在计算顶点 u 到 v 的最短路径时,u 的拓扑序大于 v 的拓扑序,则顶点 u 到 v 的最短路径一定为∞.

图 3 所示为图 2(c)的 CTG-tree 索引构建过程.图 3(a)表示 Metis 划分过程,其中 $l=2$,$\theta=5$.首先,G' 被划分为 G_1 和 G_2,接下来 G_1 被划分为 G_3 和 G_4;G_2 被划分为 G_5 和 G_6;因为 $|V_3|=|V_6|=4<\theta$,$|V_4|=|V_5|=5\leqslant\theta$,所以 Metis 终止划分.构建的 CTG-tree 得到的每个子图的入边界点、出边界点如图 3(b)所示.例如 $G_4.B_{\mathrm{i}}=\{\langle v_1,4\rangle,\langle v_3,3\rangle,\langle v_4,6\rangle\}$;$G_4.B_{\mathrm{o}}=\{\langle v_3,\{5,6\}\rangle,\langle v_4,\{7,8\}\rangle\}$;$G_2.B_{\mathrm{i}}=\{\langle v_3,7\rangle,\langle v_6,9\rangle\}$;$G_2.B_{\mathrm{o}}=\varnothing$.

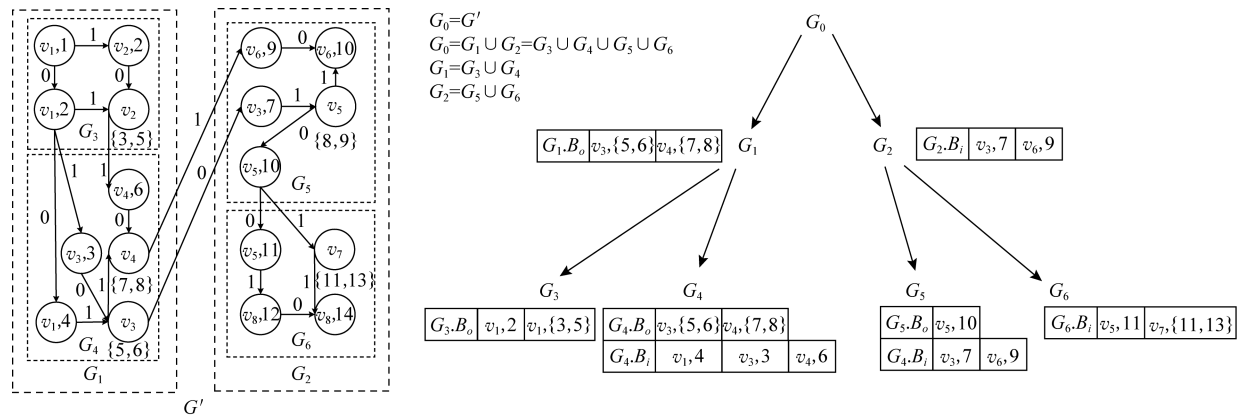


Fig. 3 Metis Partitioning and CTG-tree Construction

图 3 Metis 划分以及 CTG-tree 构建过程

随后,计算 CTG-tree 中的叶子节点对应子图 G_3,G_4,G_5 和 G_6 的游出距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{o}}$、游入距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{i}}$ 和边界点距离矩阵 $G_i.\boldsymbol{D}_{\mathrm{b}}$.

例如,对于子图 G_4,$G_4.\boldsymbol{D}_{\mathrm{o}}$ 计算 G_4 所有点$\langle v_1,4\rangle$,$\langle v_3,3\rangle$,$\langle v_4,6\rangle$,$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$到所有出边界点$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$的最短路径距离;$G_4.\boldsymbol{D}_{\mathrm{i}}$ 计算 G_4 所有入边界点$\langle v_1,4\rangle$,$\langle v_3,3\rangle$,$\langle v_4,6\rangle$到所有点$\langle v_1,4\rangle$,$\langle v_3,3\rangle$,$\langle v_4,6\rangle$,$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$的最短路径距离;$G_4.\boldsymbol{D}_{\mathrm{b}}$ 计算 G_4 所有出边界点$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$到入边界点$\langle v_1,4\rangle$,$\langle v_3,3\rangle$,$\langle v_4,6\rangle$的最短路径距离.

而后,自底向上计算非叶节点对应子图 G_1,G_2 和 G_0 的出边界点游入距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Bi}}$、入边界点游入距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Ii}}$ 和边界点游出距离矩阵 $G_j.\boldsymbol{D}_{\mathrm{Bo}}$.

例如,对于子图 G_1,$G_1.\boldsymbol{D}_{\mathrm{Bi}}$ 计算 G_3 和 G_4 的所有出边界点$\langle v_1,2\rangle$,$\langle v_2,\{3,5\}\rangle$,$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$到 G_3 和 G_4 的所有入边界点$\langle v_1,4\rangle$,$\langle v_3,3\rangle$,$\langle v_4,6\rangle$最短路径距离;$G_1.\boldsymbol{D}_{\mathrm{Bo}}$ 计算 G_3 和 G_4 的所有出边界点$\langle v_1,2\rangle$,$\langle v_2,\{3,5\}\rangle$,$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$到 G_1 的出边界点$\langle v_3,\{5,6\}\rangle$,$\langle v_4,\{7,8\}\rangle$之间的最短路径距离.对于子图 G_2,$G_2.\boldsymbol{D}_{\mathrm{Ii}}$ 计算 G_2 入边界点$\langle v_3,7\rangle$,$\langle v_6,9\rangle$到 G_5 和 G_6 的所有入边界点$\langle v_3,7\rangle$,$\langle v_6,9\rangle$,$\langle v_5,11\rangle$,$\langle v_7,\{11,13\}\rangle$之间的最短路径距离;最终 CTG-tree 中每个节点保存的距离矩阵索引如图 4 所示.

CTG-tree 索引保存 3 部分信息:CTG-tree 节点、边界点以及距离矩阵.CTG-tree 叶子节点对应子图的顶点数目不超过阈值 θ,每层节点数为 l,

$G_0.\boldsymbol{D}_{\mathrm{Bi}}$

	v_3,7	v_3,9
v_3,{5,6}	0	2
v_4,{7,8}	∞	1

$G_1.\boldsymbol{D}_{\mathrm{Bo}}$

	v_3,{5,6}	
v_1,2	1	2
v_2,{3,5}	∞	1
v_3,{5,6}	0	1
v_4,{7,8}	∞	0

$G_1.\boldsymbol{D}_{\mathrm{Bi}}$

	v_1,4	v_3,3	v_4,6
v_1,2	0	1	2
v_2,{3,5}	∞	∞	1
v_3,{5,6}	∞	∞	∞
v_4,{7,8}	∞	∞	∞

$G_2.\boldsymbol{D}_{\mathrm{Bi}}$

	v_3,7	v_5,11	v_6,9	v_7,{11,13}
v_5,10	∞	0	∞	1

$G_2.D_{\mathrm{li}}$

	v_3,7	v_5,11	v_6,9	v_7,{11,13}
v_3,7	0	1	∞	2
v_6,9	∞	∞	0	∞

$G_3.\boldsymbol{D}_{\mathrm{o}}$

	v_1,2	v_2,{3,5}
v_1,1	0	1
v_1,2	0	1
v_2,2	∞	0
v_2,{3,5}	∞	0

$G_4.\boldsymbol{D}_{\mathrm{o}}$

	v_3,{5,6}	v_4,{7,8}
v_1,4	1	2
v_3,3	0	1
v_3,{5,6}	0	1
v_4,6	∞	0
v_4,{7,8}	∞	0

$G_4.\boldsymbol{D}_{\mathrm{i}}$

	v_1,4	v_3,3	v_3,{5,6}	v_4,6	v_4,{7,8}
v_1,4	0	∞	1	∞	2
v_3,3	∞	0	0	∞	1
v_4,6	∞	∞	∞	0	0

$G_4.\boldsymbol{D}_{\mathrm{b}}$

	v_1,4	v_3,3	v_4,6
v_3,{5,6}	∞	∞	∞
v_4,{7,8}	∞	∞	∞

$G_5.\boldsymbol{D}_{\mathrm{o}}$

	v_5,10
v_3,7	1
v_5,{8,9}	0
v_5,10	0
v_6,9	∞
v_6,10	∞

$G_5.\boldsymbol{D}_{\mathrm{i}}$

	v_3,7	v_5,{8,9}	v_5,10	v_6,9	v_6,10
v_3,7	0	1	1	∞	2
v_6,9	∞	∞	∞	0	0

$G_5.\boldsymbol{D}_{\mathrm{b}}$

	v_3,7	v_6,9
v_5,10	∞	∞

$G_6.\boldsymbol{D}_{\mathrm{i}}$

	v_5,11	v_7,{11,13}	v_8,12	v_8,14
v_5,11	0	∞	1	1
v_7,{11,13}	∞	0	∞	1

Fig.4 Distance Matrix Index in CTG-tree

图 4 CTG-tree 中的距离矩阵索引

所以 CTG-tree 高度为 $\log_l(|V|/\theta)+1$,CTG-tree 节点总数为 $l|V|/(l-1)\theta$.CTG-tree 边界点总数为

$$\sum_{0\leqslant j\leqslant(2^{\log_l(|V|/\theta)+1}-2)}(|G_j.B_{\mathrm{i}}|+|G_j.B_{\mathrm{o}}|).$$

CTG-tree 距离矩阵大小为

$$\sum_{0\leqslant j\leqslant\left(\frac{|V|}{\theta(l-1)}-2\right)}(|G_j.\boldsymbol{D}_{\mathrm{Bi}}|+|G_j.\boldsymbol{D}_{\mathrm{Bo}}|+|G_j.\boldsymbol{D}_{\mathrm{Ii}}|)+\sum_{\left(\frac{|V|}{\theta(l-1)}-1\right)\leqslant j\leqslant\left(\frac{l}{l-1}\times\frac{|V|}{\theta}-2\right)}(|G_j.\boldsymbol{D}_{\mathrm{i}}|+|G_j.\boldsymbol{D}_{\mathrm{o}}|+|G_j.\boldsymbol{D}_{\mathrm{b}}|).$$

所以 CTG-tree 索引空间复杂度为

$$O\Big(\frac{l}{l-1}\times\frac{|V|}{\theta}+\sum_{0\leqslant j\leqslant(2^{\log_l(|V|/\theta)+1}-2)}(|G_j.B_{\mathrm{i}}|+|G_j.B_{\mathrm{o}}|)+\sum_{0\leqslant j\leqslant\left(\frac{|V|}{\theta(l-1)}-2\right)}(|G_j.\boldsymbol{D}_{\mathrm{Bi}}|+|G_j.\boldsymbol{D}_{\mathrm{Bo}}|+|G_j.\boldsymbol{D}_{\mathrm{Ii}}|)+\sum_{\left(\frac{|V|}{\theta(l-1)}-1\right)\leqslant j\leqslant\left(\frac{l}{l-1}\times\frac{|V|}{\theta}-2\right)}(|G_j.\boldsymbol{D}_{\mathrm{i}}|+|G_j.\boldsymbol{D}_{\mathrm{o}}|+|G_j.\boldsymbol{D}_{\mathrm{b}}|)\Big).$$

4 基于 CTG-tree 索引的最短路径计算方法

本节详细阐述在线查询阶段提出的基于层次索引 CTG-tree 的最短路径计算方法.

给定时态图 G_{T},源点 s,目的点 s',时间间隔 $I=[t_s,t_a]$,顶点 s 在时间间隔 I 内到达 s'的最短时态路径 $d(s,s',I)=\min_{p\in TPSet_G(s,s',I)}W_p$ 转化为在压缩图 $G'=(V',E')$计算$\langle s,T_s\rangle$到$\langle s',T_e\rangle$的最短路径 $d(\langle s,T_s\rangle,\langle s',T_e\rangle)=\min_{p\in PSet_{G'}(\langle s,T_s\rangle,\langle s',T_e\rangle)}W_p$.其中,$T_s$ 包含 $Vout(s)$中大于或等于 t_s 的最小时间值;T_e 包含 $Vin(s')$中小于或等于 t_a 的最大时间值;函数$PSet_{G'}(\langle s,T_s\rangle,\langle s',T_e\rangle)$为 G'中从顶点$\langle s,T_s\rangle$到顶点$\langle s',T_e\rangle$的路径 p 构成的集合.

具体的基于 G'构建的 CTG-tree 索引的最短路径查询算法伪码如算法 3 所示.

算法 3. 最短路径查询算法 $CTGQ(T,s,s',I=[t_s,t_a])$.

输入:基于 G'构建的 CTG-tree 索引 T,查询源点 s,目的点 s',时间间隔 $I=[t_s,t_a]$;

输出:s 在时间间隔 I 内到达 s'的最短时态路径距离.

① $locate\langle s,T_s\rangle,\langle s',T_e\rangle$ in T; /* T_s 包含 $Vout(s)$中大于或等于 t_s 的最小时间值;T_e 包含 $Vin(s')$中小于或等于 t_a 的最大时间值 */

② $determine\langle s,T_s\rangle.leaf,\langle s',T_e\rangle.leaf$ in T; /* $\langle s,T_s\rangle.leaf,\langle s',T_e\rangle.leaf$ 分别表示顶点$\langle s,T_s\rangle,\langle s',T_e\rangle$所属子图对应的叶子节点 */

③ if $\langle s,T_s\rangle.leaf(G')=\langle s',T_e\rangle.leaf(G')=G_f$ /* $\langle s,T_s\rangle.leaf(G'),\langle s',T_e\rangle.leaf(G')$分别表示顶点$\langle s,T_s\rangle,\langle s',T_e\rangle$所属叶子节点对应的子图 */

④ $d(\langle s,T_s\rangle,\langle s',T_e\rangle)=\min\{Dijkstra(\langle s,T_s\rangle,\langle s',T_e\rangle,G_f),\min_{b_{\mathrm{o}}\in G_f.B_{\mathrm{o}},B_{\mathrm{i}}\in G_f.B_{\mathrm{i}}}(d(\langle s,T_s\rangle,b_{\mathrm{o}})+d(b_{\mathrm{o}},b_i)+d(b_i,\langle s',T_e\rangle))\}$;

⑤ else if $\langle s,T_s\rangle.leaf(G')=G_s$, $\langle s',T_e\rangle.leaf(G')=G'_s$, $G_s\neq G'_s$ /*$\langle s,T_s\rangle$与$\langle s',T_e\rangle$属于不同叶子节点对应的子图*/

⑥ $d(\langle s,T_s\rangle,\langle s',T_e\rangle)=\min\limits_{u_o\in G_l.B_o,u_i\in G_r.B_i}(d(\langle s,T_s\rangle,u_o)+d(u_o,u_i)+d(u_i,\langle s',T_e\rangle))$;

⑦ end if

⑧ 输出 $d(\langle s,T_s\rangle,\langle s',T_e\rangle)$.

首先,算法在CTG-tree上查找$\langle s,T_s\rangle$,$\langle s',T_e\rangle$,将原始时态图上最短时态路径计算转化为压缩图上点$\langle s,T_s\rangle$到点$\langle s',T_e\rangle$的最短路径计算(行①).

接下来,确定顶点$\langle s,T_s\rangle$,$\langle s',T_e\rangle$所属子图对应的CTG-tree中的叶子节点$\langle s,T_s\rangle.leaf$,$\langle s',T_e\rangle.leaf$(行②).如果$\langle s,T_s\rangle$,$\langle s',T_e\rangle$同属一个子图$G_f$,则顶点$\langle s,T_s\rangle$到$\langle s',T_e\rangle$的最短路径有2种情况:

1) $\langle s,T_s\rangle$到$\langle s',T_e\rangle$的最短路径中的点均在子图G_f中,则在子图G_f中运用Dijkstra算法计算$d(\langle s,T_s\rangle,\langle s',T_e\rangle)=Dijkstra(\langle s,T_s\rangle,\langle s',T_e\rangle,G_f)$;

2) $\langle s,T_s\rangle$到$\langle s',T_e\rangle$的最短路径中的点部分属于其他子图$G_x(\neq G_f)$,则最短路径必通过子图G_f的出边界点和入边界点,所以:

$$d(\langle s,T_s\rangle,\langle s',T_e\rangle)=\min_{b_o\in G_f.B_o,b_i\in G_f.B_i}(d(\langle s,T_s\rangle,b_o)+d(b_o,b_i)+d(b_i,\langle s',T_e\rangle));$$

其中,$d(\langle s,T_s\rangle,b_o)$,$d(b_o,b_i)$和$d(b_i,\langle s',T_e\rangle)$可分别通过查找CTG-tree索引中子图$G_f$的游出距离矩阵$G_i.\boldsymbol{D}_o$,边界点距离矩阵$G_i.\boldsymbol{D}_b$和游入距离矩阵$G_i.\boldsymbol{D}_i$得到(行③~④).

如果$\langle s,T_s\rangle.leaf(G')=G_s$,$\langle s',T_e\rangle.leaf(G')=G'_s$,$G_s\neq G'_s$,即$\langle s,T_s\rangle$,$\langle s',T_e\rangle$隶属于不同叶子节点对应子图,则顶点$\langle s,T_s\rangle$到$\langle s',T_e\rangle$的最短路径:

$$d(\langle s,T_s\rangle,\langle s',T_e\rangle)=\min_{u_o\in G_l.B_o,u_i\in G_r.B_i}(d(\langle s,T_s\rangle,u_o)+d(u_o,u_i)+d(u_i,\langle s',T_e\rangle));$$

令G_a是G_s对应叶子节点和G'_s对应叶子节点的最小公共祖先对应的子图,则G_l,G_r是G_a所在节点的孩子节点对应的子图,且$\langle s,T_s\rangle\in G_l$,$\langle s',T_e\rangle\in G_r$.上式中,$d(u_o,u_i)$可通过查找CTG-tree索引中图$G_a$的出边界点游入距离矩阵$G_a.\boldsymbol{D}_{Bi}$得到;$d(\langle s,T_s\rangle,u_o)$根据下式计算:

$$d(\langle s,T_s\rangle,u_o)=\min_{u_x\in G_s.B_o}(d(\langle s,T_s\rangle,u_x)+\min_{u_{x-1}\in G_{s-1}.B_o}d(u_x,u_{x-1})+\cdots+\min_{u_1\in G_1.B_o}d(u_1,u_0)),$$

其中,G_{s-1}对应节点是G_s对应节点的父节点,G_{s-2}对应节点是G_{s-1}对应节点的父节点,以此类推,G_1对应节点是G_1对应节点的父节点.$d(\langle s,T_s\rangle,u_x)$,$d(u_{x-1},u_x)$,…,$d(u_1,u_o)$可分别通过$G_s$的游出距离矩阵$G_s.\boldsymbol{D}_o$,$G_{s-1}$的边界点游出距离矩阵$G_{s-1}.\boldsymbol{D}_{Bo}$,…,$G_1$的边界点游出距离矩阵$G_1.\boldsymbol{D}_{Bo}$得到.

$d(u_i,\langle s',T_e\rangle)$根据下式计算:

$$d(u_i,\langle s',T_e\rangle)=\min_{v_1\in G'_1.B_i}((d(u_i,v_1)+\min_{v_2\in G'_2.B_i}d(v_1,v_2)+\cdots+\min_{v_x\in G'_s.B_i}(d(v_{x-1},v_x)+d(v_x,\langle s',T_e\rangle)))),$$

其中,G'_1对应节点是G_r对应节点的孩子节点,G'_2对应节点是G'_1对应节点的孩子节点,以此类推,G'_s对应节点是G'_{s-1}对应节点的孩子节点.$d(u_i,v_1)$,$d(v_1,v_2)$,…,$d(v_{x-1},v_x)$,$d(v_x,\langle s',T_e\rangle)$可分别通过$G_r$的入边界点游入距离矩阵$G_r.\boldsymbol{D}_{Ii}$,$G'_1$的入边界点游入距离矩阵$G'_1.\boldsymbol{D}_{Ii}$,…,$G'_{s-1}$的入边界点游入距离矩阵$G'_{s-1}.\boldsymbol{D}_{Ii}$和$G'_s$的游入距离矩阵$G'_s.\boldsymbol{D}_i$得到.

以图2(a)所示时态图为例,查询顶点v_5到v_6在时间区间$I=[1,10]$内的最短时态路径.根据算法3,首先在G'中查找对应的点为$\langle v_5,\{8,9\}\rangle$和$\langle v_6,10\rangle$.而后在$G'$上计算$\langle v_5,\{8,9\}\rangle$到$\langle v_6,10\rangle$的最短路径.由于$\langle v_5,\{8,9\}\rangle$与$\langle v_6,10\rangle$隶属于同一叶子节点对应子图$G_5$中,则计算$Dijkstra(\langle v_5,\{8,9\}\rangle,\langle v_6,10\rangle,G_5)=1$;$\min\{d(\langle v_5,\{8,9\}\rangle,\langle v_5,10\rangle)+d(\langle v_5,10\rangle,\langle v_3,7\rangle)+d(\langle v_3,7\rangle,\langle v_6,10\rangle),d(\langle v_5,\{8,9\}\rangle,\langle v_5,10\rangle)+d(\langle v_5,10\rangle,\langle v_6,9\rangle)+d(\langle v_6,9\rangle,\langle v_6,10\rangle)\}=\infty$;所以$d(v_5,v_6,[1,10])=1$.

若查询v_1到v_8在时间区间$I=[1,15]$内的最短时态路径,则转化为在G'上计算$\langle v_1,1\rangle$到$\langle v_8,14\rangle$的最短路径,由于$\langle v_1,1\rangle\in G_3$,$\langle v_8,14\rangle\in G_6$,$\langle v_1,1\rangle$与$\langle v_8,14\rangle$隶属于不同子图.则根据算法3有:

$$d(\langle v_1,1\rangle,\langle v_8,14\rangle)=\min_{u_o\in\{\langle v_3,\{5,6\}\rangle,\langle v_4,\{7,8\}\rangle\},u_i\in\{\langle v_3,7\rangle,\langle v_6,9\rangle\}}(d(\langle v_1,1\rangle,u_o)+d(u_o,u_i)+d(u_i,\langle v_8,14\rangle));$$

其中

$$\min_{u'_o\in\{\langle v_1,2\rangle,\langle v_2,\{3,5\}\rangle\},u_o=\langle v_3,\{5,6\}\rangle}(d(\langle v_1,1\rangle,u'_o)+d(u'_o,\langle v_3,\{5,6\}\rangle))=1;$$

$$\min_{u'_o\in\{\langle v_1,2\rangle,\langle v_2,\{3,5\}\rangle\},u_o=\langle v_4,\{7,8\}\rangle}(d(\langle v_1,1\rangle,u'_o)+d(u'_o,\langle v_4,\{7,8\}\rangle))=2;$$

$$\min_{u_i=\langle v_3,7\rangle,u'_i\in\{\langle v_5,11\rangle,\langle v_7,\{11,13\}\rangle\}}(d(\langle v_3,7\rangle,u'_i)+d(u'_i,\langle v_8,14\rangle))=2;$$

$$\min_{u_i=\langle v_6,9\rangle,u'_i\in\{\langle v_5,11\rangle,\langle v_7,\{11,13\}\rangle\}}(d(\langle v_3,7\rangle,u'_i)+d(u'_i,\langle v_8,14\rangle))=\infty;$$

所以最终,$d(\langle v_1,1\rangle,\langle v_8,14\rangle)=d(\langle v_1,1\rangle,\langle v_1,2\rangle)+d(\langle v_1,2\rangle,\langle v_3,\{5,6\}\rangle)+d(\langle v_3,\{5,6\}\rangle,\langle v_3,7\rangle)+d(\langle v_3,7\rangle,\langle v_5,11\rangle)+d(\langle v_5,11\rangle,\langle v_8,14\rangle)=3$.

算法 3 中,$d(s,s',I)$的计算转化为压缩图上$d(\langle s,T_s\rangle,\langle s',T_e\rangle)$的计算,分为 2 种情况:1)当$\langle s,T_s\rangle,\langle s',T_e\rangle$同属一个子图时,时间复杂度为$O(\theta\log\theta)$;2)当$\langle s,T_s\rangle,\langle s',T_e\rangle$不属于同一子图时,需要迭代遍历$\langle s,T_s\rangle$所在叶子节点对应子图$G_s$的游出距离矩阵,$G_s$对应节点的前驱节点相应子图$G_{s-1},G_{s-2},\cdots,G_1$的边界点游出距离矩阵,$G_a$的出边界点游入距离矩阵,$G_r$的入边界点游入距离矩阵,$G_r$对应节点的后继节点相应子图$G'_1,G'_2,\cdots,G'_{s-1}$的入边界点游入距离矩阵和$\langle s',T_e\rangle$所在叶子节点对应子图$G'_s$的游入距离矩阵;所以时间复杂度为

$$O\Big(\sum_{1\leqslant j\leqslant s-1}(|G_j.\boldsymbol{D}_{\mathrm{Bo}}|+|G'_j.\boldsymbol{D}_{\mathrm{Ii}}|)+|G_a.\boldsymbol{D}_{\mathrm{Bi}}|+|G_1.\boldsymbol{D}_{\mathrm{Bo}}|+|G_r.\boldsymbol{D}_{\mathrm{Ii}}|+|G_s.\boldsymbol{D}_{\mathrm{o}}|+|G_{s'}.\boldsymbol{D}_{\mathrm{i}}|\Big).$$

5 实验分析

本节在真实的数据集上对所提出的 CTGQ 方法进行实验测试,并与 2 种流行方法 1-pass[7] 和 TDFS 进行对比,以验证 CTGQ 的效率.

5.1 实验设置

本文使用 4 个真实数据集进行实验测试,分别是 GTFS 数据集 Transit①,以及 SNAP 公开的数据集 Bitcoin②,Mathoverflow③,Askubuntu④. Transit 数据集记录了美国科罗拉多州丹佛县班车数据,包括班车号、班车出发时间、到达时间,停靠站点等信息.Bitcoin 数据集是一个 who-trust-whom 网络,网络中的顶点表示在一个名为比特币 OTC 交易平台上使用比特币进行交易的用户;由于比特币用户是匿名的,为防止欺诈和风险用户交易,用户之间会进行声誉评分,因此网络中的边记录一个用户给另一个用户的声誉评分以及评分时间.Mathoverflow 和 Askubuntu 数据集均为 Stack Exchange 下的互动时态网络,顶点表示用户,从用户 u 到用户 v 的边包含 3 种类型的互动关系:用户 u 在时间 t 回答了 v 的问题;用户 u 在时间 t 评论了 v 的问题;用户 u 在时间 t 评论了 v 的回答.表 1 给出了实验测试中用到数据集的统计信息.其中$|V_T|,|E_T|,|T(G_T)|$分别表示时态图的顶点数、边数和时态区间数.对于每个数据集,随机选取 500 组查询,查询时间间隔默认为 $I=[0,+\infty]$,报道每个算法的平均查询时间.

Table 1 Statistics of the Datasets

表 1 数据集统计信息

数据集	$\lvert V_T\rvert$	$\lvert E_T\rvert$	$\lvert T(G_T)\rvert$
Transit	143	6 999	6 995
Bitcoin	5 881	35 592	35 445
Mathoverflow	21 688	107 581	90 489
Askubuntu	137 517	280 102	262 106

本文将 CTGQ 与 1-pass[7] 和 TDFS 进行对比. 1-pass 算法和和 TDFS 均无需构建索引,1-pass 算法在转化图上运用 Dijkstra 算法计算最短时态路径;TDFS 算法在原始时态图上进行深度优先遍历计算最短时态路径;实验测试过程中,为取得较好的索引性能和查询性能,4 个数据集 Transit,Bitcoin,Mathoverflow,Askubuntu 默认参数 $l=16$;θ 分别设置为 128,64,256,128.本文所有实验程序均使用 C++语言编写,实验测试环境为一台配置为英特尔至强 CPU 处理器 E5-2650 v4,128 GB 内存,1 TB 硬盘的服务器.

5.2 实验结果与分析

表 2 给出了原始数据集采用 3.1 节提出的规则进行转化和压缩后得到的转化图和压缩图的大小.与表 1 数据进行对比,可以看出转化图的顶点数是

Table 2 Statistics of the Transformed Graph and the Compressed Graph

表 2 转化图和压缩图统计信息

数据集	转化图 G		压缩图 G'	
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert V'\rvert$	$\lvert E'\rvert$
Transit	13 972	27 494	7 159	14 212
Bitcoin	71 063	117 461	49 585	88 429
Mathoverflow	215 160	316 125	195 392	293 504
Askubuntu	560 180	719 343	520 041	673 130

① http://www.rtd-denver.com/services

② https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

③ https://snap.stanford.edu/data/sx-mathoverflow.html

④ https://snap.stanford.edu/data/sx-askubuntu.html

原始图顶点数的 4～97 倍,转化图的边数是原始图边数的 2～4 倍.相较于转化图,压缩图顶点数是转化图顶点的 3～50 倍,压缩图边数是转化图边数的 2～3 倍,说明了本文提出压缩规则的有效性.

表 3 给出了 CTG-tree 索引构建时间和空间代价.可以看出,Transit 数据集上构建 CTG-Tree 索引需要时间不足 1 s;Askubuntu 数据集上构建 CTG-tree 索引需要时间接近 8 min.在 4 个真实数据集上,CTG-tree 索引大小在 20 MB 到 8.2 GB 之间.

Table 3 CTG-tree Construction Cost and Space Overhead

表 3 CTG-tree 索引构建时间和空间代价

数据集	构建时间/s	存储空间/MB
Transit	0.597	24
Bitcoin	29.491	1 286
Mathoverflow	203.149	6 857
Askubuntu	453.466	8 344

表 4 给出了 CTGQ,1-pass 和 TDFS 算法在 4 个数据集上的平均查询时间.由表 4 可以观察出,CTGQ 查询时间比 1-pass 平均快 1 个数量级,比 TDFS 平均快 2 个数量级.这是因为 1-pass 算法与 TDFS 算法均需要在线遍历搜索,1-pass 算法需要在转化图上运用 Dijkstra 算法计算 2 点之间的最短时态路径;TDFS 算法在计算最短时态路径时,由于最短时态路径的子路径不满足最优子结构性质,所以 TDFS 需要遍历的时态路径是指数级的,耗时更长;而 CTGQ 利用了 CTG-tree 索引进行查询,查询过程中无需再遍历压缩图,只需要根据 CTG-tree 索引矩阵即可得到结果,因此效率更高.

Table 4 Query Time

表 4 查询时间 ms

数据集	CTGQ	1-pass	TDFS
Transit	2	181	596
Bitcoin	89	1 692	6 028
Mathoverflow	264	4 004	8 398
Askubuntu	453	7 726	13 649

本文实验验证了 4 个不同的时间间隔 I_i(1≤i≤4)对最短时态路径查询时间的影响.I_1 是数据集的最大时间间隔,I_{i+1}(1≤i≤3)是将 I_i 划分为 2 个相等子区间后的第一个子区间.图 5 给出了 CTGQ,1-pass 和 TDFS 算法在 Transit,Bitcoin,Mathoverflow,Askubuntu 数据集上的实验结果.

从图 5 可以看到,CTGQ,1-pass 和 TDFS 的查询时间随着时态区间的缩减而降低,TDFS 下降趋势最为显著.这是因为时态区间缩减时,TDFS 所需遍历的时态路径数目大幅度减少;而 1-pass 和 CTGQ 算法在压缩图上计算,时态区间减小时,1-pass 算法

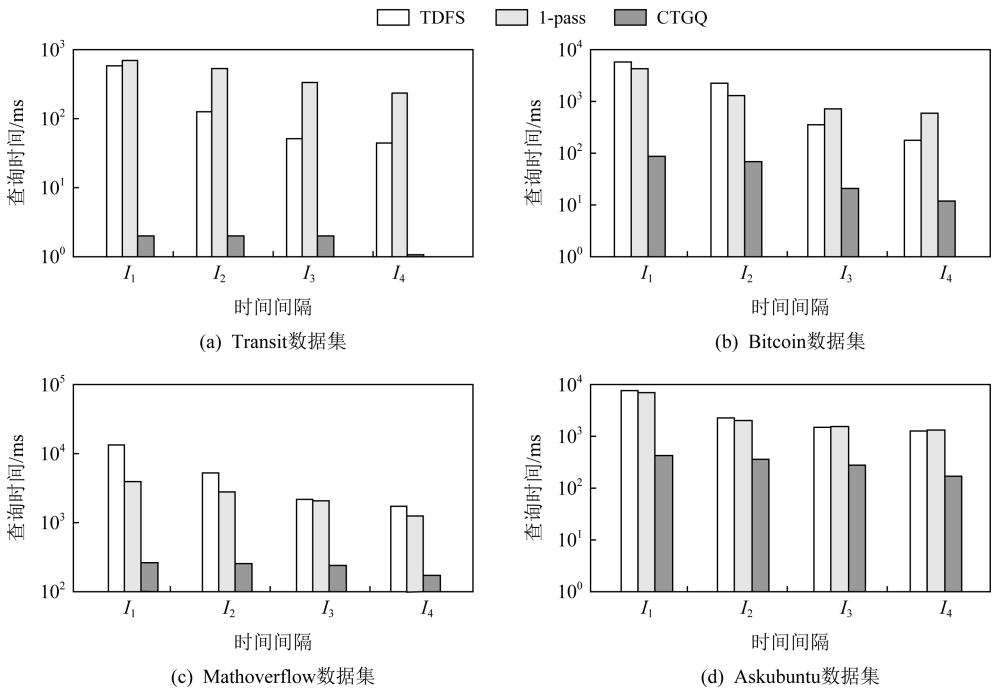


Fig. 5 Effect of Time Interval

图 5 时间区间的影响

遍历的路径长度减小,CTGQ 算法遍历的 CTG-tree 矩阵索引计算量减少.

另外,从图 5 还可以观察到不同查询时态区间下,CTGQ 算法的性能始终远远优于 1-pass 和 TDFS 算法的性能,这与前述实验分析结论一致.

本文考察了叶子子图顶点数目阈值 θ、迭代划分子图数目 l 对 CTG-tree 构建和 CTGQ 算法运行性能的影响.表 5 给出了 4 个数据集 Transit,Bitcoin,Mathoverflow,Askubuntu 在 θ 值分别设置为 2,4,8,16,l 值分别设置为 32,64,128,256,512 情况下的 CTG-tree 索引构建时间和 CTGQ 算法运行时间结果.

Table 5 Effects of θ and l

表 5 θ 和 l 的影响 ms

数据集	l	CTG-tree 构建时间				平均查询时间			
		$\theta=2$	$\theta=4$	$\theta=8$	$\theta=16$	$\theta=2$	$\theta=4$	$\theta=8$	$\theta=16$
Transit	32	452	408	524	565	10	5	3	2
	64	429	414	483	606	10	6	2	2
	128	377	416	378	597	10	6	3	2
	256	360	365	386	616	11	6	2	2
	512	303	316	359	440	11	6	2	2
Bitcoin	32	26.0	21.4	21.5	29.6	495	313	215	91
	64	24.1	18.0	21.3	29.4	494	312	197	89
	128	21.5	18.7	18.2	29.2	496	314	198	96
	256	18.8	16.0	18.1	25.1	511	315	203	92
	512	17.1	15.9	18.0	24.7	480	311	196	89
Mathoverflow	32	303	217	214	264	749	726	525	291
	64	275	185	166	205	1 093	799	451	283
	128	244	184	162	205	1 070	767	550	285
	256	215	147	167	203	972	854	517	264
	512	179	150	129	201	908	769	483	275
Askubuntu	32	943	596	501	560	1.52	0.87	1.08	0.50
	64	828	557	501	556	1.28	0.86	1.06	0.49
	128	705	465	391	453	1.31	1.03	1.07	0.45
	256	650	458	389	402	0.87	0.85	1.07	0.50
	512	541	375	399	408	1.45	0.88	1.07	0.50

首先由表 5 可以看到,当 l 取固定值时,CTG-tree 索引构建时间随着 θ 值增加呈减少趋势.这是因为 θ 值增加,CTG-tree 叶子节点对应子图的顶点数目增多,迭代划分产生的总子图数目减少,相应的入边界点、出边界点总数减少,索引矩阵计算量减少,因此 CTG-tree 索引构建时间降低.

其次,由表 5 可以看出,当 θ 取固定值时,CTG-tree 索引构建时间随着 l 值增大基本呈先降低再升高的趋势.这是因为随着 l 值的增大,CTG-tree 每层节点数目增多,入边界点、出边界点总数先减少再增多,导致索引矩阵计算量先减小后增加.

此外,当 θ 取固定值时,CTGQ 算法运行时间随着 l 值增大基本呈降低趋势.这是因为,l 值增大,CTG-tree 高度降低,CTGQ 算法自底向上,自顶向下遍历层数减少.

鉴于上述观察,为取得较好的索引性能和查询性能,4 个数据集默认参数 $l=16$;θ 在数据集 Transit,Bitcoin,Mathoverflow,Askubuntu 上的值分别设置为 128,64,256,128.

6 总 结

本文研究了时态图最短路径问题并提出了基于 CTG-tree 索引的计算方法,其主要包含预处理阶段

和在线查询阶段.在预处理阶段,本文提出了一种无损压缩方法和层次索引 CTG-tree.其首先将时态图转化为普通图,对普通图进行无损压缩以减小图处理规模,将时态图上最短路径计算转化为在压缩图上执行;而后对压缩图采用 Metis 划分将压缩图层次划分为若干个子图,得到每个子图的入边界点与出边界点集合,并基于此构建 CTG-tree.CTG-tree 为每个叶子节点对应子图计算游入距离矩阵、游出距离矩阵和边界点距离矩阵;为每个非叶节点对应子图计算出边界点游入距离矩阵,入边界点游入距离矩阵和边界点游出距离矩阵.查询阶段本文基于 CTG-tree 索引设计了高效的最短路径查询算法.最后,在真实的时态图数据集上进行了全面的实验,实验结果验证了本文所提出的基于 CTG-tree 索引的算法的效率.此外,设计高效的分布式时态图最短路径方法与增量计算算法也是一个重要且值得研究的问题,后续工作计划对其进行深入地探索.

作者贡献声明:张天明负责问题定义、方法设计、数据分析与论文撰写修改工作;徐一恒负责数据收集与整理、实验结果可视化工作;蔡鑫伟负责部分实验测试和整理工作;范菁负责指导论文写作并提出修改建议.

参 考 文 献

[1] Bast H, Delling D, Goldberg A, et al. Route planning in transportation networks [M] //Algorithm Engineering. Berlin: Springer, 2016: 19-80

[2] Wang Yong, Li Guoliang, Tang Ning. Querying shortest paths on time dependent road networks [J]. Proceedings of the VLDB Endowment, 2019, 12(11): 1249-1261

[3] Huo Wenyu, Tsotras V J. Efficient temporal shortest path queries on evolving social graphs [C] //Proc of the Int Conf on Scientific and Statistical Database Management. Berlin: Springer, 2014: 38:1-38:4

[4] Ding Bolin, Yu J X, Qin Lu. Finding time-dependent shortest paths over large graphs [C] //Proc of the Int Conf on Extending Database Technology: Advances in Database Technology. New York: ACM, 2008: 205-216

[5] Yuan Ye, Lian Xiang, Wang Guoren, et al. Constrained shortest path query in a large time-dependent graph [J]. Proceedings of the VLDB Endowment, 2019, 12(10): 1058-1070

[6] Wu Huanhuan, Cheng J, Huang Silu, et al. Path problems in temporal graphs [J]. Proceedings of the VLDB Endowment, 2014, 7(9): 721-732

[7] Wu Huanhuan, Cheng J, Ke Yiping, et al. Efficient algorithms for temporal path computation [J]. IEEE Transactions on Knowledge & Data Engineering, 2016, 28(11): 2927-2942

[8] Zhong Ruicheng, Li Guoliang, Tan K L, et al. G-tree: An efficient and scalable index for spatial search on road networks [J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(8): 2175-2189

[9] Panda M, Mishra A. Asurvey of shortest-path algorithms [J]. International Journal of Applied Engineering Research, 2018, 13(9): 6817-6820

[10] Zwick U. Exact and approximate distances in graphs—A survey [C] //Proc of European Symp on Algorithms. Berlin: Springer, 2001: 33-48

[11] Hansen P. Bicriterion path problems [M] //Multiple criteria decision making theory and application. Berlin: Springer, 1980: 109-127

[12] Dantzig G B, Thapa M N. Linear programming 2: Theory and extensions [M]. Berlin: Springer, 2006

[13] Batz G V, Geisberger R, Neubauer S, et al. Time-dependent contraction hierarchies and approximation [C] //Proc of the Int Symp on Experimental Algorithms. Berlin: Springer, 2010: 166-177

[14] Potamias M, Bonchi F, Castillo C, et al. Fast shortest path distance estimation in large networks [C] //Proc of the 18th ACM Int Conf on Information and Knowledge Management. New York: ACM, 2009: 867-876

[15] Akiba T, Iwata Y, Yoshida Y. Fast exact shortest-path distance queries on large networks by pruned landmark labeling [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2013: 349-360

[16] Geisberger R, Sanders P, Schultes D, et al. Contraction hierarchies: Faster and simpler hierarchical routing in road networks [C] //Proc of the Int Workshop on Experimental and Efficient Algorithms. Berlin: Springer, 2008: 319-333

[17] Botea A. Ultra-fast optimal pathfinding without runtime search [C] //Proc of the Conf on Artificial Intelligence and Interactive Digital Entertainment. Palo Alto, CA: AAAI, 2011: 122-127

[18] Botea A, Harabor D. Path planning with compressed all-pairs shortest paths data [C] //Proc of the Int Conf on Automated Planning and Scheduling. Palo Alto, CA: AAAI, 2013: 293-297

[19] Geisberger R, Sanders P, Schultes D, et al. Exact routing in large road networks using contraction hierarchies [J]. Transportation Science, 2012, 46(3): 388-404

[20] Ouyang Dian, Yuan Long, Qin Lu, et al. Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees [J]. Proceedings of the VLDB Endowment, 2020, 13(5): 602-615

[21] Delling D, Goldberg A V, Pajor T, et al. Customizable route planning in road networks [J]. Transportation Science, 2017, 51(2): 566-591

[22] Zhang Dongxiang, Yang Dingyu, Wang Yuan, et al. Distributed shortest path query processing on dynamic road networks [J]. The VLDB Journal, 2017, 26(3): 399-419

[23] Hassan M S, Aref W G, Aly A M. Graph indexing for shortest-path finding over dynamic sub-graphs [C] //Proc of the 2016 Int Conf on Management of Data. New York: ACM, 2016: 1183-1197

[24] Thorup M, Zwick U. Approximate distance oracles [J]. Journal of the ACM, 2005, 52(1): 1-24

[25] Elkin M, Peleg D. $(1+\varepsilon, \beta)$-spanner constructions for general graphs [J]. SIAM Journal on Computing, 2004, 33(3): 608-631

[26] Wang Sibo, Lin Wenqing, Yang Yi, et al. Efficient route planning on public transportation networks: A labelling approach [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2015: 967-982

[27] Wu Huanhuan, Huang Yuzhen, Cheng J, et al. Reachability and time-based path queries in temporal graphs [C] //Proc of the Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2016: 145-156

Zhang Tianming, born in 1988. PhD, lecturer. Member of CCF. Her main research interests include graph data management and deep learning.

张天明,1988 年生.博士,讲师,CCF 会员.主要研究方向为图数据管理和深度学习.

Xu Yiheng, born in 2000. Bachelor candidate. His main research interests include graph querying and processing.

徐一恒,2000 年生.本科生.主要研究方向为图查询处理.

Cai Xinwei, born in 1998. Master candidate. His main research interests include graph analysis and mining.

蔡鑫伟,1998 年生.硕士研究生.主要研究方向为图分析与挖掘.

Fan Jing, born in 1969. PhD, professor, PhD supervisor. Distinguished member of CCF. Her main research interests include middleware, virtual reality and visualization.

范　菁,1969 年生.博士,教授,博士生导师,CCF 杰出会员.主要研究方向为中间件、虚拟现实和可视化.