时间依赖代价函数下的最优路径查询问题研究

杨雅君 高 宏 李建中
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 作者研究了时间依赖图下,具有时间限制的费用代价最优路径的查询问题.目前有关时间依赖图上的最短路径查询的研究工作解决的是最短旅行时间问题(TDSP),这些工作都利用了以下性质:到达某个顶点的最早时刻可以通过到达其邻居的最早时刻计算得出.然而,在计算具有时间限制的费用代价最优路径时,该性质并不成立.因此,目前解决 TDSP 问题的方法均不能解决文中面对的问题.对此作者提出一个新的算法用于计算时间依赖图模型上的满足时间限制的费用代价最优路径.该算法适用于有向图和无向图.作者证明了算法的时间复杂度和空间复杂度分别为 $O(kn\log n+mk^2\log k)$ 和 $O((n+m)k)$.最后,作者通过真实数据集上的实验,验证了该算法的有效性.

关键词 时间依赖图;代价函数;最优路径
中图法分类号 TP311 **DOI 号**: 10.3724/SP.J.1016.2012.02247

Finding the Optimal Path under Time-Dependent Cost Function on Graphs

YANG Ya-Jun GAO Hong LI Jian-Zhong
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

Abstract Shortest path query is an important problem in graphs and has been well-studied on static graphs. However, in real applications, the costs of edges in graphs always change over time. We call such graphs as time-dependent graphs. In this paper, we study how to find the optimal path with the minimum cost under time constraint on large time-dependent graphs. Most existing works about time-dependent shortest path problem (TDSP) focus on finding the shortest path with the minimum travel time. All these works utilize the following property: the earliest arriving time of vertex v can be computed by the earliest arriving time of v's neighbors. Unfortunately, this property does not hold in our problem. In this paper, we propose a novel algorithm to compute the optimal path with the minimum cost under time constraint. We show the time and space complexity are $O(kn\log n+mk^2\log k)$ and $O((n+m)k)$ respectively. We confirm the effectiveness and efficiency of our algorithms using real-life datasets in experiments.

Keywords time-dependent graph; cost function; optimal path

1 引 言

图模型被广泛地用来刻画现实世界中各种各样实体间的复杂关系,如交通网、生物信息学、XML 数据库以及社交网络等等.随着信息技术的发展,各个应用领域的信息量都呈现了爆炸性增长趋势.因此,图模型数据的规模也变得越来越大.最短路径查询

收稿日期:2012-06-30;最终修改稿收到日期:2012-09-02.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316200)、国家自然科学基金(60903016,61003046,60533110,60773063,61173022)、黑龙江省自然科学基金(F201031)、中国博士后科学基金(20110491064)、黑龙江省博士后基金(LBH-Z09140)、哈尔滨工业大学科研创新基金"中央高校基本科研业务费专项基金"(HIT.NSRIF.2010060)资助.**杨雅君**,男,1983 年生,博士研究生,主要研究方向为图数据库和图挖掘、海量数据处理. E-mail: yjyang@hit.edu.cn. **高 宏**,女,1966 年生,教授,博士生导师,主要研究领域为无线传感器网络、物联网、海量数据管理和数据挖掘等. **李建中**,男,1950 年生,教授,博士生导师,主要研究领域为数据库、海量数据处理、物联网和无线传感器网络等.

是图上一类经典的查询问题并有着重要的应用.目前,大多数的最短路径查询都是定义在静态图上的.然而,在现实应用中,图模型往往不是固定不变的,而是随着时间发生着演绎与变化.例如,在交通网中,很多的交通管理系统能够向用户提供实时的交通信息.这些系统包括:车辆信息和通信系统(VICS)和欧洲交通信息频道(TMC).VICS 被应用于日本的城市交通管理,TMC 则被广泛应用于欧洲、北美和澳大利亚的城市交通管理.这些交通系统所管理的交通网都可以被刻画为大规模的图数据模型.在这些模型中,通过一条边(道路)的代价不是一成不变的,而是会随着时间的变化而变化.因此,如何在与时间相关的动态图模型上计算最短路径,就成为了一个有着十分重要研究意义的问题.

我们考虑物流网中的一个应用.快递公司需要从城市 A 运输一批货物到城市 B,假设出发时间为 t_1,且客户要求在时刻 t_2 之前收到货物.在该物流网中,通过每条公路有一个时间开销和价格开销.根据时间开销我们可以计算某条路径 p 是否满足时间限制,即是否能在时刻 t_2 之前到达城市 B.满足时间限制的路径可能有很多,因此,从这些满足时间限制的路径中,找到价格花费最少的路径,是一个有着十分重要应用意义的问题.

一个交通物流网可以刻画为一个大规模图模型.图模型上的边(道路)可以用时间代价 $w_{i,j}$ 和费用代价 $f_{i,j}(t)$ 来描述.时间代价表示通过这条边所需要花费的时间,费用代价表示通过这条边所要花费的费用(金钱).对图中的任意一条边 (v_i,v_j),通过 (v_i,v_j) 的费用代价 $f_{i,j}(t)$ 会随着 v_i 上的出发时刻 t 的变化而变化,即 $f_{i,j}(t)$ 为出发时刻 t 的函数.我们称这样的图模型为时间依赖图模型.因此,上面的查询问题可以表示为:给定起点 v_s 和终点 v_e,最早出发时刻 t_d 和最晚到达时刻 t_a,计算一条由 v_s 到 v_e 的路径 p,该路径 p 需要满足两个条件:(1)时间限制.即要求路径 p 在时刻 t_d 之后从 v_s 出发,在时刻 t_a 之前到达 v_e;(2)在所有满足条件(1)的路径中,路径 p 的通行费用总和最小.

目前,关于时间依赖图上的最短路径查询问题(Time-Dependent Shortest Path,TDSP)已经有了很多的研究工作[1-6].然而,这些工作主要解决的是最短旅行时间问题,即给定一个出发时刻 t_0,计算一个最早到达终点的时刻并找到其对应的路径;或者给定一个出发时间区间 T,找到一条最优路径,其出发时间 $t_s\in T$,使得从起点 v_s 到终点 v_e 的旅行时间总和最小.这些工作仅仅假设图中的边只存在一个时间代价函数 $w_{i,j}(t)$,即通过图中某条边的时间和该边起点上的出发时刻有关.在用时间代价函数刻画边上代价的时间依赖图模型中,我们用 t_i 表示到达顶点 v_i 的最早时刻,则 t_i 满足以下递归关系:$t_i=\min\{t_j+w_{j,i}(t)\mid v_j\in N^-(v_i)\}$.$v_j\in N^-(v_i)$ 表示 v_j 是 v_i 的入边邻居,t_j 表示 v_j 的最早到达时刻.该递归关系表示:顶点 v_i 的最早到达时刻可以根据其入边邻居的最早到达时刻计算得出.目前,解决 TDSP 问题的研究工作都利用了这一性质.然而,在费用代价函数存在的时间依赖图模型上,该关系并不成立(详见 2.2 节).因此,目前解决 TDSP 问题的方法均不能解决本文面对的问题.

运筹学领域中的一些工作研究了时间依赖图上的费用代价最优路径查询问题[7-11].然而,这些工作假定时间模型是离散的.在离散时间模型下,时间区间被表示为一系列离散时间点的集合$[t_1,t_2,\cdots,t_T]$.对图上任意一条边 (v_i,v_j),用户只能选择在某个给定时刻 t_i 从 v_i 出发.这些工作的主要问题有:(1)离散时间模型下不能找到精确的最优解.比如用户在时刻 t 到达顶点 v_i,$t_{i-1}<t<t_i$,t_{i-1} 和 t_i 是给定离散时间区间内的两个相邻的离散时间点.用户能选择的通过边 (v_i,v_j) 的最早出发时刻为 t_i.然而,可能在时刻 t' 出发才会使得通过边 (v_i,v_j) 的代价最小,这里 $t<t'<t_i$;(2)这些方法只是从理论上给出计算最优路径的方法,并没有从提高算法实际运行效率的角度来设计算法和优化算法,也没有应用数据处理中的优化技术.所以,这些方法需要承受较高的时间和空间开销.因此,运筹学领域中的这些工作并不能很好解决本文所要面对的问题.

本文中,我们研究了时间依赖图模型上的满足时间限制的费用代价最优路径的查询问题,本文中的时间模型是连续的,且允许存在等待时间以期待找到费用代价最小的路径.本文的主要贡献如下:

(1)给出了一个新的算法计算时间依赖图模型上的满足时间限制的费用代价最优路径.该算法适用于有向图和无向图.

(2)证明了算法的时间复杂度为 $O(kn\log n+mk^2\log k)$,空间复杂度为 $O((n+m)k)$,这里 n 和 m 分别为图中顶点和边的数量;k 为到达时间-最小代价函数上的分段区间数量(详见 3.1 节).

(3)通过真实数据集上的实验,验证了我们算法的有效性.

本文第 2 节介绍问题定义、现有的最有效的方

法以及它们存在的主要问题；第 3 节介绍如何计算时间限制下的费用代价最优的路径，该算法分为两个阶段：最小费用代价计算阶段和路径选择阶段；第 4 节通过真实数据集上的实验验证方法的有效性；第 5 节讨论相关工作；最后，总结本文.

2　问题定义

本节中，将介绍时间依赖图的定义和时间依赖图上的具有时间限制的费用代价最优路径查询问题的定义.

2.1　时间依赖图模型的概念

定义 1.　时间依赖图. 时间依赖图是一个有向图，记作 $G_T(V,E,W,F)$，其中 $V=\{v_i\}$ 是图上的顶点集合；$E=V\times V$ 是图上的边集合；W 是边上的时间代价取值集合；F 是边上的费用代价函数集合. 对图中每条边 $(v_i,v_j)\in E$，存在一个时间代价 $w_{i,j}\in W$ 和一个费用代价函数 $f_{i,j}(t)\in F$. $w_{i,j}$ 是一个常数，它表示通过边 (v_i,v_j) 所要花费的时间. $f_{i,j}(t)$ 是一个分段常量函数，$f_{i,j}(t)$ 的直观意义是：如果在时刻 t 从顶点 v_i 出发，则通过边 (v_i,v_j) 所要花费的费用代价为 $f_{i,j}(t)$. 无向图可以转化为有向图，即一条无向边等价于两条方向相反的有向边.

本文中我们假设通过图中任意一条边的时间代价 $w_{i,j}\geq 0$. 该假设是合理的，因为在现实世界中，不存在小于零的时间开销. 类似地，我们假设 $f_{i,j}(t)$ 在任意时刻 t 的取值为正值，因为现实世界中不存在小于零的费用开销.

本文中，$f_{i,j}(t)$ 定义为分段常量函数，其形式如下：

$$f_{i,j}(t)=\begin{cases} c_1, & t^0\leq t\leq t^1 \\ c_2, & t^1<t\leq t^2 \\ \cdots & \\ c_p, & t^{p-1}<t\leq t^p \end{cases},$$

这里，$[t^0,t^p]$ 是函数 $f_{i,j}(t)$ 的定义域；常数 c_k 为函数 $f_{i,j}(t)$ 在时间片段 $[t^{k-1},t^k]$ 上的取值. 对 $f_{i,j}(t)$ 的分段常量假设是合理的. 现实世界中费用代价函数取值是分段常量的情况十分普遍. 比如，道路网中的某一条公路，在每天不同的时间区间内的收费标准不同. 因此，通过该公路的费用代价函数即是一个分段常量函数.

给定起点 v_s 和终点 v_e，最早出发时刻 t_d 和最晚到达时刻 t_a，本文的目的是找到一条路径 p^*，使得 (1) p^* 在时刻 t_d 之后离开 v_s，且在时刻 t_a 之前到达 v_e；(2) 在所有满足条件 (1) 的路径中，p^* 的费用代价总和最小.

给定一条路径 p，其到达终点 v_e 的费用代价与时间有关. 对于任意一条边 $(v_i,v_j)\in p$，离开 v_i 的时刻不同，通过边 (v_i,v_j) 的费用也就不同. 因此，路径 p 的费用代价也不同. 本文中，允许等待时间，即到达顶点 v_i 后，允许在 v_i 等待一段时间 $\omega(v_i)$，以期待路径 p 的费用代价更小. 下文中，我们用 $arrive(v_i)$ 和 $depart(v_i)$ 分别表示到达 v_i 的时刻和离开 v_i 的时刻. 对 $\forall v_i\in V$，有

$$depart(v_i)=arrive(v_i)+\omega(v_i).$$

令路径 $p=v_1\rightarrow v_2\rightarrow\cdots\rightarrow v_h$，给定最早出发时刻 t_d，和顶点 v_i 上的等待时间 $\omega(v_i)$，有

$$arrive(v_1)=t_d,$$
$$arrive(v_2)=depart(v_1)+w_{1,2},$$
$$\cdots$$
$$arrive(v_h)=depart(v_{h-1})+w_{h-1,h}.$$

因此，给定最早出发时刻 t_d 和顶点 v_i 上的等待时间 $\omega(v_i)$，对路径 p 上的每个顶点 v_i，沿路径 p 到达 v_i 的代价 $cost_p(v_i)$ 可递归的定义为

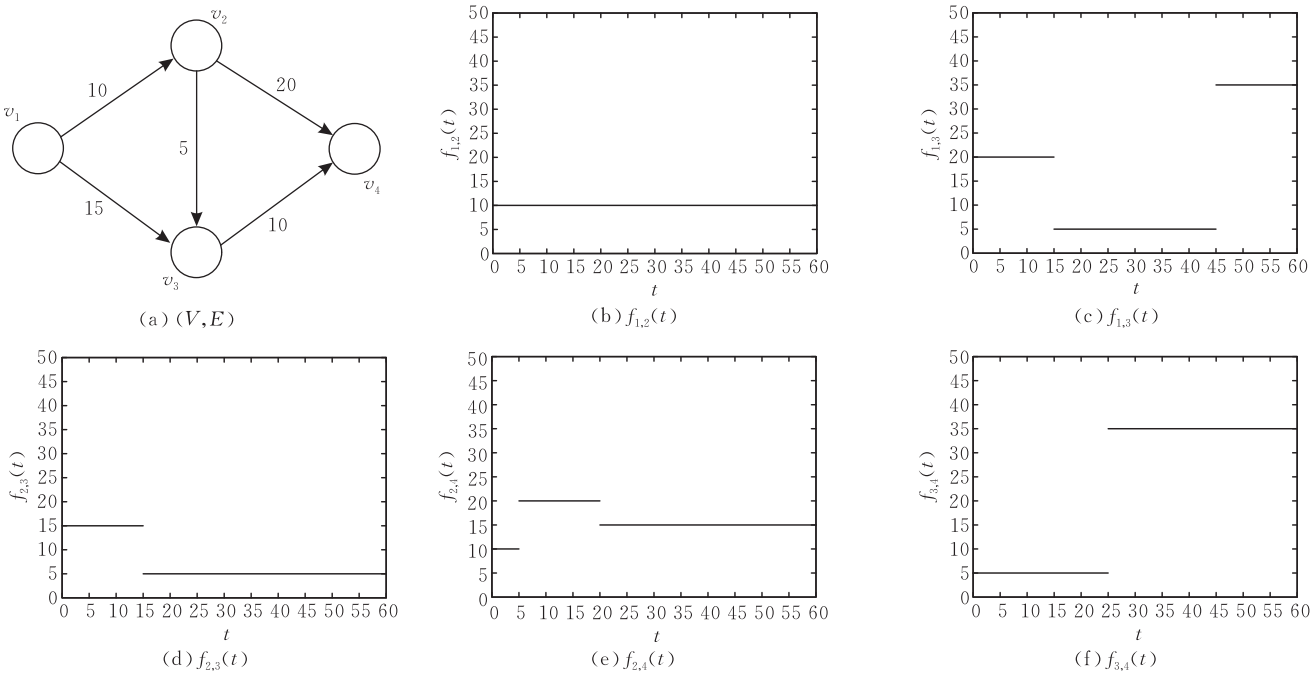
$$cost_p(v_1)=0,$$
$$cost_p(v_2)=cost_p(v_1)+f_{1,2}(depart(v_1)),$$
$$\cdots$$
$$cost_p(v_h)=cost_p(v_{h-1})+f_{h-1,h}(depart(v_h)).$$

我们定义路径 p 的代价 $cost(p)=cost_p(v_h)$. 下面，我们给出时间依赖图上最优路径查询问题的定义.

定义 2.　时间依赖图上的具有时间限制的费用代价最优路径查询问题. 给定一个时间依赖图 $G_T(V,E,W,F)$，起点 v_s，终点 v_e，最早出发时刻 t_d 和最晚到达时刻 t_a. 时间依赖图上具有时间限制的费用最优路径查询问题的目的是找到一条路径 p^* 以及 p^* 上每个顶点 v_i 的最佳等待时间 $\omega^*(v_i)$，使得 (1) $depart(v_s)\geq t_d\wedge arrive(v_e)\leq t_a$；(2) $cost(p^*)$ 在所有满足条件 (1) 的 v_s 到 v_e 的路径中最小.

图 1 给出了一个时间依赖图的示例，图 1(a) 是一个时间依赖图 G_T，该图包含 4 个顶点和 5 条边. 边 (v_i,v_j) 上的数字表示该边的时间代价 $w_{i,j}$，即通过边 (v_i,v_j) 要花费的时间. 图 1(b)～(f) 分别给出了边 (v_1,v_2)，(v_1,v_3)，(v_2,v_3)，(v_2,v_4) 和 (v_3,v_4) 的费用代价函数 $f_{i,j}(t)$.

给出一个图 G_T 上的费用最优路径查询，$v_s=v_1$，$v_e=v_4$，$t_d=0$，$t_a=60$，则费用最优路径为 p^*：$v_1\rightarrow v_2\rightarrow v_3\rightarrow v_4$，在顶点 v_1，v_2 和 v_3 上的最佳等待时间分别为 $\omega^*(v_1)=0$，$\omega^*(v_2)=5$ 和 $\omega^*(v_3)=0$，路径 p^* 的代价为 $cost(p^*)=20$.



(a) (V,E) (b) $f_{1,2}(t)$ (c) $f_{1,3}(t)$

(d) $f_{2,3}(t)$ (e) $f_{2,4}(t)$ (f) $f_{3,4}(t)$

图 1 时间依赖图 G_T

2.2 现有方法介绍及存在问题分析

目前,时间依赖图模型上的最优路径查询问题主要集中于解决最短旅行时间的问题(TDSP)[1-7],即给定出发点 v_s、目的点 v_e和出发时间区间 I,找到一条由 v_s到 v_e的最优路径和最佳出发时间 $t_s \in I$,使得该路径的旅行时间最短.在该类问题中,通过任意一条边(v_i,v_j)的时间 $w_{i,j}(t)$会随着 v_i的出发时刻 t 的变化而变化,$w_{i,j}(t)$是一个与离开 v_i时刻有关的函数.我们将介绍两种计算最短旅行时间路径的最有效的方法,并分析这些方法为什么不能用于本文所要面对的问题.

A*算法. 文献[3]提出了一种基于 A* 算法的扩展算法 KDXZ. KDXZ 算法的思想是利用优先队列来维护所有待扩展的路径.假设 p_i为一条从起点 v_s到顶点 v_i的路径,需要注意的是,v_s到 v_i可能存在许多条路径,这些路径可能同时被维护在优先队列中.每一条 v_s到 v_e的路径的旅行时间代价可以用以下函数计算:$f_{p_i}(t)=g_{p_i}(t)+d_{i,e}-t$.这里,$g_{p_i}(t)$表示在时刻 t 离开 v_s,并沿路径 p_i到达 v_i的时刻;$d_{i,e}$是一个估计的 v_i到 v_e的旅行时间下界,$d_{i,e}=d_{\mathrm{ecu}}(v_i,v_e)/v_{\max}$, $d_{\mathrm{ecu}}(v_i,v_e)$表示 v_i和 v_e之间的欧氏距离,$v_{\max}$表示网络中的最大速度.$f_{p_i}(t)$表示在时刻 t 离开 v_s,并沿路径 p_i到达 v_e的估计时间开销.每一次迭代中,算法从队列中挑选一条路径 p_i,使得 $\min\limits_t\{f_{p_i}(t)\}$在被队列维护的所有路径中最小.算法在路径 p_i末尾加入边(v_i,v_j)得到一条新路径 p_j,并将 p_j加入到优先队列.当 v_s到 v_e的路径 p_e第一次从队列中弹出时,算法终止.KDXZ 算法不允许等待时间的 $\omega(v_i)$存在.

KDXZ 算法主要存在 3 方面的问题:(1) KDXZ 算法需要两点间的欧式距离和网络中的最高速度来估计旅行时间开销的下界.然而,在本文面对的问题中,边上的费用代价函数是任意的,且费用代价与时间开销不同,无法通过"距离/速度"的公式估计.因此,KDXZ 算法无法用于本文的问题;(2) KDXZ 算法的效率取决于所估计的下界对搜索空间的裁剪能力.然而,在大规模图上,KDXZ 算法很难计算出一个较准确的下界.当图的规模很大,或者起点和终点的距离很远时,KDXZ 算法的效率很低;(3) 在最坏情况下,KDXZ 算法可能要枚举每一条起点到终点的路径并将其维护在优先队列中.因此,KDXZ 算法的时间和空间复杂度是图中顶点规模的指数级.

2S 算法. 文献[5]中提出了目前解决 TDSP 问题的最有效算法 2S,该算法基于 Dijkstra 算法的思想.2S 算法主要分为:(1) 最早到达时刻函数计算阶段;(2) 路径选择阶段.在阶段(1),算法根据以下公式计算最早到达时刻函数:

$$g_i(t)=\min_{v_j\in N^-(v_i),\omega(v_j)}\{(g_j(t)+\omega(v_j))+w_{j,i}(g_j(t)+\omega(v_j))\} \tag{1}$$

$g_i(t)$是一个与出发时刻 t 相关的函数,它表示在时

刻 t 离开起点 v_s 的所有路径中，到达顶点 v_i 的最早时刻. $N^-(v_i)$ 表示顶点 v_i 的入边邻居集合，即 $N^-(v_i)=\{v_j \mid (v_j,v_i)\in E\}$. 算法用优先队列维护图中所有顶点 v_i 的最早到达时间函数 $g_i(t)$ 和一个时间区间 $[t_s,\tau_i]$. $g_i(t)$ 在区间 $[t_s,\tau_i]$ 内的取值是准确的. 当顶点 v_i 从队列中弹出时，算法根据式(1)来更新函数 $g_i(t)$ 和区间 $[t_s,\tau_i]$. 若更新后的时间区间 $[t_s,\tau_i]\neq I$，I 为给定的出发时间区间，则算法将 v_i 重新插回队列. 当终点 v_e 的最早到达时间函数 $g_e(t)$ 在整个区间 I 内取值确定后，算法结束. 在阶段(2)，算法根据 $g_e(t)$，找到起点 v_s 到终点 v_e 旅行时间最短的路径. 该算法的时间复杂度为 $O((n\log n+m)\alpha(T))$，n 为图 G_T 中顶点数量，m 为图 G_T 中边的数量，$\alpha(T)$ 为更新 $g_e(t)$ 和区间 $[t_s,\tau_e]$ 的次数.

2S 算法是一个两阶段搜索算法，它存在的主要问题有：(1) 2S 算法的核心是利用式(1)计算最早到达时间函数. 然而，式(1)所蕴含的关系在本文所面对的问题中并不成立. 我们用例子说明. 图 1 中，目标为计算 $v_1\to v_4$ 的代价最小的路径. 对顶点 v_4 的入边邻居 v_3，我们发现到达 v_3 的最小费用代价为 $g_3(15)=5$，即在时刻 15 离开 v_1，通过边 (v_1,v_3) 到达 v_3 的费用代价最小(费用为 5). 此时，到达 v_3 的时刻为 30. 因此，根据式(1)，通过顶点 v_3 到达 v_4 的最小费用代价为 $g_3(15)+w_{3,4}(30)=5+35=40$. 显然，该结果是错误的. 因为，从 v_1 出发，通过顶点 v_3 到达 v_4 的最优路径为 $v_1\to v_2\to v_3\to v_4$，其费用代价为 20. 造成该问题的原因是：一条费用代价最优路径上的子路径未必是费用代价最优的. 考虑一条 v_s 到 v_i 的费用代价最小的路径 p_i，v_j 为 v_i 的入边邻居，v_s 到 v_j 的费用代价最小的路径为 p_j，沿 p_j 到达 v_j 的时间为 t_j. 此时，可能存在另外一条 v_s 到 v_j 的路径 p_j'，其到达 v_j 的时间为 t_j'，p_j' 的费用略微大于 p_j 的费用，但在时刻 t_j 后通过边 (v_j,v_i) 的最小费用远远大于在时刻 t_j' 后通过边 (v_j,v_i) 的最小费用，即 $\min\{w_{j,i}(t)\mid t\geq t_j\}\gg\min\{w_{j,i}(t)\mid t\geq t_j'\}$. 因此，式(1)不能通过到达顶点 v_j 的最小费用计算出到达顶点 v_i 的最小费用. 如图 1 例中，v_1 到 v_4 的最优路径为 $v_1\to v_2\to v_3\to v_4$，其子路径 $v_1\to v_2\to v_3$ 并不是 v_1 到 v_3 的最优路径；(2) 该算法的更新次数 $\alpha(T)$ 没有上界保证，如果对图中顶点估计的最早到达时间函数不准确，$\alpha(T)$ 取值可能很大，从而影响了算法的效率.

综上所述，与 TDSP 问题不同，费用代价最优路径并不具备子路径最优的性质，因此解决 TDSP 问题的方法并不能解决本文所要面对的问题.

运筹学领域中的一些工作研究了时间依赖图上的具有时间限制的费用代价最优路径问题[7-11]. 然而，这些工作中假定时间模型是离散的. 在离散时间模型中，时间区间被表示为一系列离散时间点的集合 $[t_1,t_2,\cdots,t_T]$. 给定图中任意一条边 (v_i,v_j)，用户只能选择在某个给定时间点 t_i 离开 v_i. 这些工作的主要问题有：(1) 在离散时间模型下无法找到问题的最优解. 例如，假设在时刻 t 到达顶点 v_i，$t_{i-1}<t<t_i$，这里 t_{i-1} 和 t_i 是给定离散时间区间内的两个相邻的离散时间点. 离散时间模型下，通过边 (v_i,v_j) 离开 v_i 的最早时刻为 t_i. 然而，在时刻 t' 出发才能使得通过边 (v_i,v_j) 的代价最小，这里 $t<t'<t_i$，离散时间模型无法取到时刻 t'，因此也就不能找到费用最优的路径；(2) 这些方法只是从理论上给出计算费用最优路径的方法，并没有从提高算法实际运行效率的角度来设计算法和优化算法，也没有应用数据处理中的优化技术. 因此，这些方法需要承受较高的时间和空间开销.

本文中，我们给出了一种有效的计算时间依赖图上具有时间限制的费用代价最优路径的算法，该算法是一个两阶段搜索算法. 在第 1 阶段，我们计算起点 v_s 到达终点 v_e 的最小费用代价(详见 3.1 节和 3.2 节). 在第 2 阶段，我们根据第 1 阶段的结果，找到起点 v_s 到 v_e 的最优路径和路径上每个点的最佳等待时间. 我们算法的时间复杂度为 $O(kn\log n+mk^2\log k)$，空间复杂度为 $O((n+m)k)$.

3　两阶段搜索算法

在本节中，我们提出一个有效的两阶段搜索算法. 首先介绍如何根据顶点 v_j 的到达时间-最小代价函数更新其邻居 v_i 的到达时间-最小代价函数；然后，介绍算法的第 1 阶段，即计算起点 v_s 到终点 v_e 的最小费用代价；再次，介绍算法的第 2 阶段，即如何根据起点 v_s 到终点 v_e 的最小费用代价，找到满足时间限制的 v_s 到 v_e 的费用最优路径和路径上每个顶点的最佳等待时间；最后，分析算法的时间复杂度和空间复杂度. 算法 1 给出了两阶段搜索算法的流程.

算法 1.　两阶段搜索算法.

Two-Step-Search(G_T,v_s,v_e,t_d,t_a)

输入：依赖图 G_T，起点 v_s，终点 v_e，最早出发时刻 t_d，最晚到达时刻 t_a

输出：费用代价最优路径p^*，p^*上各点等待时间$\omega^*(v_i)$

1. Computing-$g_e^*(t^*)$ (G_T, v_s, v_e, t_d, t_a)；
2. if $g_e^*(t^*)\neq\infty$ then
3. Path-Selection ($g_e^*(t^*), G_T, v_s, v_e, t_d, t_a$)；
4. return $p^*, \omega^*(v_i)$ for each $v_i\in p^*$；
5. else return $\varnothing$.

3.1 更新到达时间-最小代价函数

首先介绍什么是到达时间-最小代价函数.

给定时间依赖图G_T，对任意一个顶点$v_i\in G_T$，在时刻t_d之后离开起点v_s，并且在时刻t到达顶点v_i的路径p可能存在很多，我们用$P_{s,i,t_d}(t)$表示这些路径的集合.用$g_i(t)$表示所有这些路径中的最小代价，即

$$g_i(t)=\min\{cost(p)\mid p\in P_{s,i,t_d}(t)\}.$$

需要注意的是，路径p上允许等待时间的存在，只要路径p能够满足出发时刻和到达时刻的要求，即满足$depart_p(v_s)\geq t_d$，表示路径p在时刻t_d之后离开起点v_s；$arrive_p(v_i)=t$，表示路径p在时刻t到达顶点v_i. $g_i(t)$是一个随着t变化的函数，我们称之为顶点v_i的到达时间-最小代价函数，它反映了在时刻t到达顶点v_i所要花费的最小费用.

引理 1. 给定时间依赖图G_T，若图中任意一条边上的费用代价函数$f_{i,j}(t)$是分段常量函数，则对$\forall v_i\in V$，v_i的到达时间-最小代价函数$g_i(t)$也是分段常量函数.

证明. 因为用代价函数$f_{i,j}(t)$是分段常量函数，$g_i(t)$的取值为路径所经过边上各个常量值的求和，因此，$g_i(t)$也为分段常量函数. 证毕.

下面，介绍如何计算$g_i(t)$.令顶点v_j是顶点v_i的一个入边邻居，即$v_j\in N^-(v_i)$.假设$g_j(t)$已知，我们可以根据$g_j(t)$更新$g_i(t)$.更新过程分为两步：(1)计算$g_{j\to i}(t)$，$g_{j\to i}(t)$表示：通过顶点v_j到达顶点v_i的到达时间-最小代价函数；(2)利用$g_{j\to i}(t)$更新$g_i(t)$.显然，如果不允许等待时间的存在，即$\omega(v_j)=0$，则有

$$g_{j\to i}(t)=g_j(t-w_{j,i})+f_{j,i}(t-w_{j,i}).$$

该公式的意义如下：如果在时刻$t-w_{j,i}$到达顶点v_j，其最小费用代价为$g_j(t-w_{j,i})$.因为$\omega(v_j)=0$，所以需要立即离开顶点v_j，此时通过边(v_j,v_i)的费用为$f_{j,i}(t-w_{j,i})$.因为通过边(v_j,v_i)的时间开销为$w_{j,i}$，所以通过顶点v_j且在时刻t到达顶点v_i的最小费用代价为$g_j(t-w_{j,i})+f_{j,i}(t-w_{j,i})$.

本文中，我们允许存在等待时间$\omega(v_j)$，以期待在时刻t，$g_{j\to i}(t)$能取到最小的费用代价.因此，

$$g_{j\to i}(t)=\min_{t',\omega(v_j)}\{g_j(t')+f_{j,i}(t'+\omega(v_j))\}$$
$$t=t'+\omega(v_j)+w_{j,i} \quad (2)$$

这里，$t'+\omega(v_j)$为离开顶点v_j的时间，即在时刻t'到达顶点v_j后，等待$\omega(v_j)$时间后再离开顶点v_j.因此，当在时刻t'到达顶点v_j后，如果要保证在时刻t到达顶点v_i，则需要选取合适的等待时间$\omega(v_j)$，满足$t=t'+\omega(v_j)+w_{j,i}$.在允许等待时间存在的情况下，给定时刻t，计算$g_{j\to i}(t)$的问题等价于找到一个最优的t'和$\omega(v_j)$，满足$t-w_{j,i}=t'+\omega(v_j)$，使得式(2)中$g_{j\to i}(t)$最小.

给定到达顶点v_i的时刻t，离开顶点v_j的时刻也就被确定为$t-w_{j,i}$.因此，此时通过边(v_j,v_i)的费用代价也就被确定为$f_{j,i}(t-w_{j,i})$.根据$g_{j\to i}(t)$的定义，我们知：

$$g_{j\to i}(t)=\min_{t'\leq t-w_{j,i}}\{g_j(t')+f_{j,i}(t-w_{j,i})\}$$
$$=\min_{t'\leq t-w_{j,i}}\{g_j(t')\}+f_{j,i}(t-w_{j,i}) \quad (3)$$

根据以上式(3)，计算$g_{j\to i}(t)$等价于找到最优的时刻t'，$t'\leq t-w_{j,i}$，使得$g_j(t')$最小，也就是说，对于每一个出发时刻$t-w_{j,i}$，我们只需找到一个与$f_{j,i}(t-w_{j,i})$对应的最优$g_j(t')$，并与$f_{j,i}(t-w_{j,i})$求和即可.

我们给出一个有效的算法计算函数$g_{j\to i}(t)$.给定顶点v_j的到达时间-代价函数$g_j(t)$，$g_j(t)$的定义域为$[\lambda_j,t_a]$.λ_j为在时刻t_d离开v_s，能够到达顶点v_j的最早时刻，也就是说，当在最早出发时刻t_d从起点v_s出发时，我们不可能早于时刻λ_j到达顶点v_j.进一步地，通过顶点v_j到达顶点v_i的最早时刻为$\lambda_{j\to i}=\lambda_j+w_{j,i}$.我们只需计算$g_{j\to i}(t)$在区间$[\lambda_{j\to i},t_a]$内的取值即可.对于边$(v_j,v_i)$的费用代价函数$f_{j,i}(t)$，只需考虑$f_{j,i}(t)$在区间$[\lambda_j,t_a-w_{j,i}]$内的取值即可.因为当离开顶点$v_j$的时刻晚于$t_a-w_{j,i}$时，不可能在最晚到达时刻$t_a$前到达顶点$v_i$，所以也不可能在最晚到达时刻$t_a$前到达终点$v_e$.

算法 2. 更新算法.

Updating-$g_i(t)$($g_i(t), g_j(t), f_{j,i}(t)$)

输入：$g_i(t), g_j(t), f_{j,i}(t)$

输出：$g_i(t)$

1. $\alpha_j\leftarrow t_a-w_{j,i}$；
2. while $\alpha_j\neq\lambda_j$ do
3. $g_{\alpha_j}^*\leftarrow\min\{g_j(t)\mid t\leq\alpha_j\}$；
4. $\varphi_{\alpha_j}\leftarrow\min\{t\mid g_j(t)=g_{\alpha_j}^*, t\leq\alpha_j\}$；
5. for each time sub-interval $[t^{x-1},t^x]\subseteq[\varphi_{\alpha_j},\alpha_j]$ do
6. $g_{j\to i}(t+w_{j,i})=f_{j,i}(t)+g_{\alpha_j}^*$；
7. $\alpha_j\leftarrow\varphi_{\alpha_j}$；

8. $g_i(t) \leftarrow \min\{g_i(t), g_{j\to i}(t)\}$；

9. return $g_i(t)$.

算法 2 给出了计算 $g_{j\to i}(t)$ 的流程. 我们用 $[\lambda_j, \alpha_j]$ 表示 $f_{j,i}(t)$ 的未处理区间，即区间 $[\lambda_j, \alpha_j]$ 内的 $f_{j,i}(t)$ 没有找到最优的 $g_j(t')$ 使得式(3)取值最小. 初始化阶段，$\alpha_j = t_a - w_{j,i}$. 算法 1 不断更新未处理区间 $[\lambda_j, \alpha_j]$，当 $f_{j,i}(t)$ 在全部时间区间 $[\lambda_j, t_a - w_{j,i}]$ 内都找到对应的最优 $g_j(t')$ 时，$g_{j\to i}(t)$ 在区间 $[\lambda_{j\to i}, t_a]$ 内的取值即被确定，算法结束.

我们用 $g_{\alpha_j^*}$ 表示 $g_j(t)$ 在 $t \le \alpha_j$ 时能取到的最小值，即

$$g_{\alpha_j^*} = \min\{g_j(t) \mid t \le \alpha_j\}.$$

初始化阶段，

$$g_{\alpha_j^*} = \min\{g_j(t) \mid t \le t_a - w_{j,i}\}.$$

由于 $g_j(t)$ 为分段常量函数，令 φ_{α_j} 为 $t \le \alpha_j$ 条件下使得 $g_j(t)$ 取值为 $g_{\alpha_j^*}$ 的最小的时刻 t，即

$$\varphi_{\alpha_j} = \min\{t \mid g_j(t) = g_{\alpha_j^*}, t \le \alpha_j\},$$

φ_{α_j} 实质上是 $g_j(t)$ 在取值为 $g_{\alpha_j^*}$ 的分段区间上的左端点. 因此，$g_{\alpha_j^*}$ 即为区间 $[\varphi_{\alpha_j}, \alpha_j]$ 内的函数 $f_{j,i}(t)$ 对应的最优 $g_j(t')$. 我们考虑代价函数 $f_{j,i}(t)$ 在区间 $[\varphi_{\alpha_j}, \alpha_j]$ 内的取值. 因为 $f_{j,i}(t)$ 为分段常量函数，所以 $f_{j,i}(t)$ 在区间 $[\varphi_{\alpha_j}, \alpha_j]$ 内也是分段常量函数. 不失一般性，我们设 $f_{j,i}(t)$ 在区间 $f_{j,i}(t)$ 内分为 q 个子区间 $[t^0, t^1], [t^1, t^2], \cdots, [t^{q-1}, t^q]$，这里，$t^0 = \varphi_{\alpha_j}$，$t^q = \alpha_j$. 假设 $f_{j,i}(t)$ 在子区间 $[t^{x-1}, t^x]$ 内取值为 c_x，则 $g_{j\to i}(t)$ 在区间 $[t^{x-1} + w_{j,i}, t^x + w_{j,i}]$ 内的取值为 $c_x + g_{\alpha_j^*}$.

需要注意的是，为了方便讨论，本文假设 $g_j(t)$ 和 $f_{j,i}(t)$ 上的所有分段区间均为闭区间. 开区间存在时，我们的方法仍然有效. 对 $g_j(t)$，当 $g_{\alpha_j^*}$ 对应的时间区间为开区间时，φ_{α_j} 取值为该开区间的左端点即可. 在计算 $g_{j\to i}(t)$ 时，如果 $g_{\alpha_j^*}$ 对应区间的左端或 $f_{j,i}(t)$ 上的子区间的一端为开区间，则计算出的 $g_{j\to i}(t)$ 在对应区间上的对应一端也为开区间.

当在区间 $[\varphi_{\alpha_j}, \alpha_j]$ 内的 $f_{j,i}(t)$ 都找到其对应的最优 $g_j(t')$ 后，$g_{j\to i}(t)$ 在区间 $[\varphi_{\alpha_j} + w_{j,i}, \alpha_j + w_{j,i}]$ 内的取值即计算完毕. 此时，我们令 $\alpha_j \leftarrow \varphi_{\alpha_j}$，并在区间 $[\lambda_j, \alpha_j]$ 内重复以上过程. 当 $\varphi_{\alpha_j} = \lambda_j$ 时，意味着区间 $[\lambda_j, \alpha_j]$ 内的 $f_{j,i}(t)$ 均可找到对应的最优 $g_j(t')$，因此，我们可计算出 $g_{j\to i}(t)$ 在区间 $[\lambda_j + w_{j,i}, \alpha_j + w_{j,i}]$ 内的取值. 至此，$g_{j\to i}(t)$ 在全部区间 $[\lambda_{j\to i}, t_a]$ 内的取值计算完毕，算法结束.

我们用图 2 中的例子说明 $g_{j\to i}(t)$ 的计算过程. 如图 2 所示，图中实线代表函数 $f_{j,i}(t)$ 在区间 $[\lambda_j, t_a - w_{j,i}]$ 内的取值，图中虚线代表函数 $g_j(t)$. 初始化阶段，$\alpha_j = t_a - w_{j,i}$. 当 $t \le \alpha_j$ 时，$g_j(t)$ 的最小取值为 $g_{\alpha_j^*} = 15$，使得 $g_j(t)$ 取值为 15 的最小时刻 t 为 $\varphi^1_{\alpha_j}$，即此时 $\varphi_{\alpha_j} = \varphi^1_{\alpha_j}$. 因此，函数 $f_{j,i}(t)$ 在区间 $[\varphi^1_{\alpha_j}, t_a - w_{j,i}]$ 内找到对应的最优 $g_j(t)$，我们可根据 $g_{\alpha_j^*}$ 和 $f_{j,i}(t)$ 在区间 $[\varphi^1_{\alpha_j}, t_a - w_{j,i}]$ 内取值计算 $g_{j\to i}(t)$ 在区间 $[\varphi^1_{\alpha_j} + w_{j,i}, t_a]$ 内的取值. 因此，$g_{j\to i}(t)$ 在区间 $[\varphi^1_{\alpha_j} + w_{j,i}, t_a]$ 内的取值为 $g_{\alpha_j^*} + f_{j,i}(t) = 15 + 10 = 25$. 第 2 次迭代中，算法 2 将 α_j 更新为 $\varphi^1_{\alpha_j}$. 此时，$t \le \alpha_j$ 下能取到 $g_j(t)$ 的最小值 $g_{\alpha_j^*} = 20$，其对应的 φ_{α_j} 为 λ_j. 因为 $\varphi_{\alpha_j} = \lambda_j$，算法 2 在区间 $[\lambda_j, \varphi^1_{\alpha_j}]$ 内找到 $f_{j,i}(t)$ 对应的最优 $g_j(t)$ 为 $g_{\alpha_j^*} = 20$. 算法 2 将 $g_{j\to i}(t)$ 在区间 $[\lambda_{j\to i}, \varphi^1_{\alpha_j} + w_{j,i}]$ 内的取值赋值为 $g_{\alpha_j^*} + f_{j,i}(t) = 20 + 10 = 30$. 此时，在全部区间 $[\lambda_j, t_a - w_{j,i}]$ 内，$f_{j,i}(t)$ 都找到与之对应的最优的 $g_j(t)$，进而函数 $g_{j\to i}(t)$ 在全部区间 $[\lambda_{j\to i}, t_a]$ 的取值也计算完毕，算法结束.

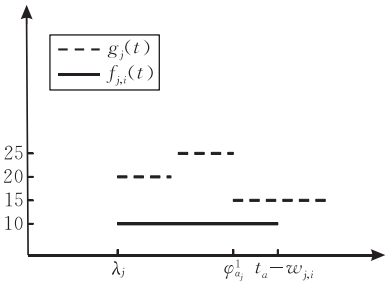


图 2　计算 $g_{j\to i}(t)$

下面，我们给出定理证明算法 2 的正确性.

定理 1.　给定时间依赖图 G_T，$v_i, v_j \in G_T$ 且 $v_j \in N^-(v_i)$. 令 $g_j(t)$ 为顶点 v_j 的到达时间-最小代价函数，则算法 2 计算的 $g_{j\to i}(t)$ 是在此 $g_j(t)$ 下最优的，即如果顶点 v_j 的到达时间-最小代价函数为 $g_j(t)$，则对 $\forall t \in [\lambda_{j\to i}, t_a]$，$g_{j\to i}(t)$ 是所有通过顶点 v_j 并且在时刻 t 到达顶点 v_i 的路径中最小的费用代价取值.

证明.　不失一般性，对 $\forall t \in [\lambda_{j\to i}, t_a]$，令 $g^*_{j\to i}(t)$ 为通过顶点 v_j 并且在时刻 t 到达 v_i 的最小费用代价. 我们只需证明 $g_{j\to i}(t) = g^*_{j\to i}(t)$ 即可. 因为 $g^*_{j\to i}(t)$ 为最优的费用代价，所以有 $g^*_{j\to i}(t) \le g_{j\to i}(t)$. 下面，我们证明 $g_{j\to i}(t) \le g^*_{j\to i}(t)$. 因为 $g^*_{j\to i}(t)$ 和 $g_{j\to i}(t)$ 所代表的路径都是在时刻 t 到达顶点 v_i，所以他们离开顶点 v_j 的时刻均为 $t - w_{j,i}$，从而他们通过边 (v_j, v_i) 所花费的费用也均为 $f_{j,i}(t - w_{j,i})$. 由 $g^*_{j\to i}(t)$ 和 $g_{j\to i}(t)$ 的定义可知：

$$g^*_{j\to i}(t)=g^*_j(t'^*)+f_{j,i}(t-w_{j,i}),$$

和

$$g_{j\to i}(t)=g_j(t')+f_{j,i}(t-w_{j,i}),$$

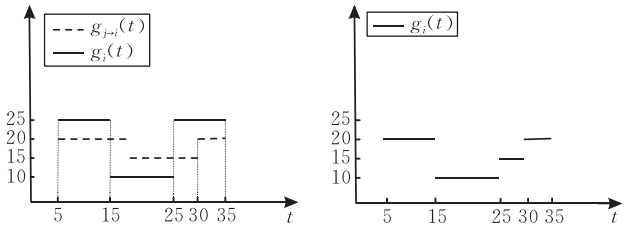
其中，$g^*_j(t'^*)$和$g_j(t')$分别为$g^*_{j\to i}(t)$和$g_{j\to i}(t)$所代表的路径中到达顶点v_j时的花费的费用代价，t'^*和t'分别为其到达顶点v_j的时刻. 显然，我们有$t'^*\le t-w_{j,i}$和$t'\le t-w_{j,i}$. 根据算法2可知，$g_j(t')$选取的是$t'\le t-w_{j,i}$条件下函数$g_j(t)$能取得的最小函数值，因此，我们有$g_j(t')\le g^*_j(t'^*)$，进而$g_{j\to i}(t)\le g^*_{j\to i}(t)$. 证毕.

定理1说明了算法2得到的$g_{j\to i}(t)$是最优的. 令$g_i(t)$为顶点v_i上现有的到达时间-代价函数(可能不是最优的)，可以根据$g_{j\to i}(t)$来更新$g_i(t)$. 对$\forall t\in[\lambda_{j\to i},t_a]$，有

$$g_i(t)=\min\{g_{j\to i}(t),g_i(t)\mid t\in[\lambda_{j\to i},t_a]\}.$$

由于$g_i(t)$和$g_{j\to i}(t)$均为分段常量函数，在对应的每个时间子区间内，只需取$g_{j\to i}(t)$和$g_i(t)$的最小值作为更新后$g_i(t)$的函数值即可.

图3中的例子展示了如何更新$g_i(t)$. 图3(a)中，虚线为$g_{j\to i}(t)$，实线为现有的$g_i(t)$. 我们发现在区间[5,15]内，$g_{j\to i}(t)$取值为20，小于$g_i(t)$取值. 因此，区间[5,15]内的$g_i(t)$取值更新为20. 区间[25,30]和[30,35]内$g_{j\to i}(t)$取值分别为15和20，均小于$g_i(t)$. 因此，区间[25,30]内$g_i(t)$更新为15，区间[30,35]内$g_i(t)$更新为20. 图3(b)中给出了更新后的$g_i(t)$函数.



(a) 更新前$g_i(t)$ (b) 更新后$g_i(t)$

图3 更新$g_i(t)$

3.2 计算到达终点的最小费用代价

在本节中，我们介绍算法1的第1阶段，即如何计算起点v_s到终点v_e的最小费用代价.

3.2.1 计算到达终点的最小费用代价

预处理阶段，我们需计算起点v_s到图中任意顶点v_i的最早到达时间λ_i. 考虑仅以时间代价$w_{i,j}$为权值的图G_T，我们在G_T上运行现有最快的单源最短路径算法，即可计算出到达各个顶点v_i的最小时间代价. 将该时间代价与最早出发时刻t_d相加，即可计算出各个顶点v_i的最早到达时刻λ_i. 目前，最快的单源最短路径算法的时间复杂度为$O(n\log n+m)$.

我们用$g^*_e(t^*)$表示终点v_e的到达时间-最小代价函数$g_e(t)$在区间$[\lambda_e,t_a]$上的最小值，则$g^*_e(t^*)$即为满足时间限制的起点v_s到终点v_e的最小费用代价. 本节的目标即是计算$g^*_e(t^*)$.

算法3给出了计算$g^*_e(t^*)$的流程. 对图中任意顶点v_i，我们用T_i表示顶点v_i最早-最晚到达时刻区间，即$T_i=[\lambda_i,t_a]$. 算法3逐步地更新区间$[\lambda_i,t_a]$内的函数$g_i(t)$. 我们用S_i表示$g_i(t)$已确定取值的区间，用τ_i表示当前的$g_i(t)$在T_i-S_i上的最小取值.

算法3. 最小费用代价计算算法.

Computing-$g^*_e(t^*)(G_T,v_s,v_e,t_d,t_a)$

输入：时间依赖图G_T，起点v_s，终点v_e，最早出发时刻t_d，最晚到达时刻t_a

输出：$g^*_e(t^*)$

1. $g_s(t)\leftarrow 0$; $\tau_s\leftarrow 0$; $S_s\leftarrow T_s$
2. Q be the priority queue initially containing V; $v_i\leftarrow v_s$;
3. while $v_i\neq v_e$ do
4. Let $[t_i^a,t_i^b]$ be the time slot τ_i corresponding to;
5. $g_i(t\mid t\in[t_i^a,t_i^b])\leftarrow\tau_i$; $S_i\leftarrow S_i\cup[t_i^a,t_i^b]$;
6. for each $v_j\in N^+(v_i)$ do
7. Updating-$g_j(t)(g_j(t),g_i(t),f_{i,j}(t))$;
8. $\tau_j\leftarrow\min\{g_j(t)\mid t\in T_j-S_j\}$;
9. if $S_i\neq T_i$ then
10. $\tau_i\leftarrow\min\{g_i(t)\mid t\in T_j-S_j\}$;
11. enqueue(Q,v_i);
12. $v_i\leftarrow$dequeue(Q);
13. $g^*_e(t^*)\leftarrow\tau_i$;
14. return $g^*_e(t^*)$.

初始化阶段，对于起点v_s，其$g_s(t)$、S_s和τ_s被初始化为$g_s(t)\leftarrow 0$，$S_s\leftarrow T_s$，$\tau_s\leftarrow 0$. 显然，如果从起点v_s出发，则在任何时刻t到达起点v_s的最小费用代价均为0. 因此，函数$g_s(t)$在全部区间上是一个取值为零的常数. 对于其它任意顶点$v_i\neq v_s$，其$g_i(t)$、S_i和τ_i分别被初始化为$g_i(t)\leftarrow\infty$，$S_i\leftarrow\varnothing$，$\tau_i\leftarrow\infty$. 这意味着在任何区间上都没有确定函数$g_i(t)$的取值.

我们使用一个优先队列Q维护图中的顶点. 优先队列Q根据τ_i的取值由小至大地弹出队列中的顶点. 初始阶段，队列Q包括全部顶点$v_i\in V$. 当终点v_e第1次从队列Q中弹出时，算法结束.

在每次迭代中，τ_i取值最小的顶点v_i从队列Q中弹出. 令τ_i取值对应的区间为$[t_i^a,t_i^b]$. 因此，顶点v_i的到达时间-最小代价函数$g_i(t)$在$[t_i^a,t_i^b]$上的取

值即为 τ_i. 这一事实我们将在下面给出定理证明. 我们将区间$[t_i^a, t_i^b]$加入顶点 v_i 的已确定区间集合 S_i 来更新 S_i，即 $S_i \leftarrow S_i \cup [t_i^a, t_i^b]$. 然后，对顶点 v_i 的所有出边邻居 $v_j \in N^+(v_i)$，根据算法 2 更新区间 $[\lambda_{i\to j}, t_a]$内的 $g_j(t)$. 同时，算法更新 τ_i.

当算法更新完顶点 v_i 的所有出边邻居 v_j 的 $g_j(t)$后，算法判断 $g_i(t)$的取值是否在全部区间上已经被确定，即是否 $S_i = T_i$. 如果 $S_i = T_i$，则函数 $g_i(t)$在全部区间 $T_i = [\lambda_i, t_a]$内的取值已经确定，这说明 $g_i(t)$在任意时刻 $t \in T_i$的取值不可能被更新得更小. 因此，我们可以将顶点 v_i从队列 Q 中移除. 如果 $S_i \neq T_i$，我们则计算当前 $g_i(t)$在区间 $T_i - S_i$内的最小值 τ_i，并将顶点 v_i重新插回队列 Q 中. 这里注意的是：当前的 $g_i(t)$不一定是最优的，当顶点 v_i 的入边邻居从队列 Q 中弹出时，$g_i(t)$可能会被更新，同时，τ_i也会被更新.

当终点 v_e第 1 次从队列 Q 中弹出时，v_e对应的 τ_e是即函数 $g_e(t)$在全部区间$[\lambda_e, t_a]$上的最小取值 $g_e^*(t^*)$(此事实我们将给出定理证明). 因此，起点 v_s 到终点 v_e的最小费用代价 $g_e^*(t^*)$已经被计算出来，算法终止.

3.2.2　例子演示说明

我们用图 1 中所示的例子来演示运算过程. 在该例中，我们给出费用最优路径查询，该查询中，$v_s = v_1$，$v_e = v_4$，$t_d = 0$，$t_a = 60$. 首先利用现有的最短路径算法计算出 G_T 中各个顶点 v_1，v_2，v_3，v_4的最早到达时间：$t_1 = 0$，$t_2 = 10$，$t_3 = 15$，$t_4 = 25$. 因此，函数 $g_1(t)$，$g_2(t)$，$g_3(t)$和 $g_4(t)$的取值区间分别为 $T_1 = [0, 60]$，$T_2 = [10, 60]$，$T_3 = [15, 60]$和 $T_4 = [25, 60]$.

图 4 给出了计算起点 v_s到终点 v_e的最小费用代价的过程. 初始化阶段，优先队列 Q 仅包含起点 v_1，其对应的 $\tau_1 = 0$，$S_1 = T_1$，$g_1(t) = 0$. 这意味着，在任何时刻 $t \in T_1$，到达 v_1的费用代价为 0. 因为，从 v_1到 v_1不需要花费任何费用. 当 v_1从队列中弹出后，对 v_1的两个出边邻居 v_2，v_3，算法分别计算 $g_{1\to 2}(t)$和 $g_{1\to 3}(t)$，并更新 $g_2(t)$和 $g_3(t)$. $g_2(t)$和 $g_3(t)$在本次迭代后的取值如图 4(a)和图 4(b)所示. 顶点 v_2和 v_3插入队列. 因为 $S_1 = T_1$，即 $g_1(t)$在全部区间上的取值已经确定，所以，v_1从队列 Q 中移除.

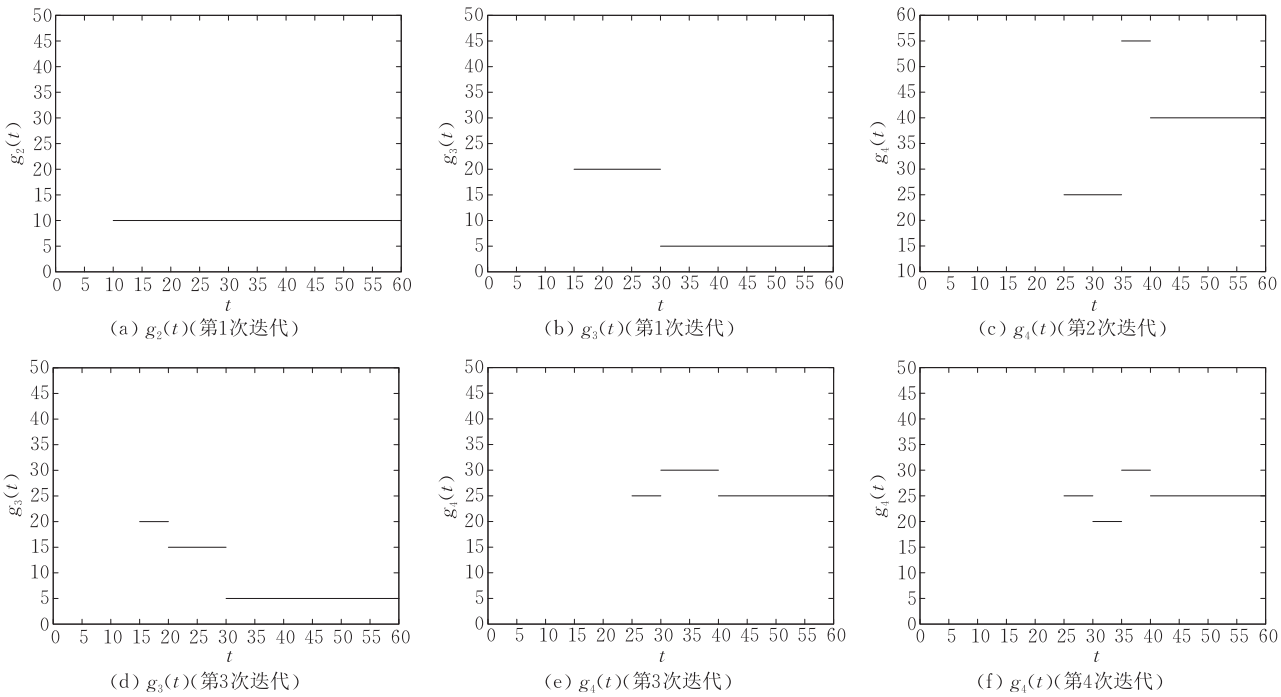


图 4　图 1 例中到达时间-最小代价函数更新过程

第 2 次迭代中，我们发现顶点 v_3在未确定区间 $T_3 - S_3$上的最小取值为 $\tau_3 = 5$. τ_3取值在队列 Q 中最小，因此顶点 v_3被弹出. 如图 4(b)所示，我们发现 $\tau_3 = 5$ 对应的取值区间为 $[30, 60]$，因此，区间 $[30, 60]$内的 $g_3(t)$取值即被确定为 5. 我们将区间 $[30, 60]$加入 S_3，即此时 $S_3 = [30, 60]$. 顶点 v_4为顶点 v_3的出边邻居，我们计算 $g_{3\to 4}(t)$并更新 $g_4(t)$. 本次迭代后的 $g_4(t)$在图 4(c)中给出. 顶点 v_4插入队列. 此时，$g_3(t)$的未确定取值区间变为 $T_3 - S_3 = [15, 30]$，且当前 $g_3(t)$在 $T_3 - S_3$上的最小值为 20. 因此，τ_3被赋值 20，顶点 v_3被重新插回队列 Q 中.

第 3 次迭代中，顶点 v_2在未确定区间 $T_2 - S_2$内

的最小取值为 $\tau_2=10$. τ_2 取值在队列 Q 中最小,因此顶点 v_2 被弹出. 如图 4(a)所示,我们发现 $\tau_2=10$ 对应的取值区间为[10,60],因此,$g_2(t)$ 在全部区间 T_2 上的取值即被确定为 10. 顶点 v_3 和顶点 v_4 为顶点 v_2 的两个出边邻居,我们分别更新 $g_3(t)$ 和 $g_4(t)$,更新后的 $g_3(t)$ 和 $g_4(t)$ 分别在图 4(d)和图 4(e)中给出. 此时,$g_3(t)$ 在未确定区间 T_3-S_3 内的最小值 τ_3 也被更新为 15. 由于 $g_2(t)$ 在全部区间 T_2 上的取值已经确定,因此,顶点 v_2 从队列 Q 中移除.

第 4 次迭代中,顶点 v_3 在未确定区间 $T_3-S_3=$[15,30]内的最小取值为 $\tau_3=15$,τ_3 取值在队列 Q 中最小,顶点 v_3 被弹出. 如图 4(d)所示,$\tau_3=15$ 对应的区间为[20,30],$g_3(t)$ 在区间[20,30]内的取值即被确定为 15. 我们将区间[20,30]加入 S_3. 此时,$S_3=$[20,60]. $g_3(t)$ 的未确定取值区间变为 $T_3-S_3=$[15,20]. 根据当前的 $g_3(t)$ 更新 v_3 的出边邻居 v_4 的 $g_4(t)$,更新结果在图 4(f)中给出. 当前 $g_3(t)$ 在未确定取值区间[15,20]内的最小值为 20,因此,τ_3 被更新为 20,顶点 v_3 被重新插入队列 Q 中.

第 5 次迭代中,顶点 v_4 在未确定区间 T_4 上的最小取值为 $\tau_4=20$,τ_4 取值在队列 Q 中最小,顶点 v_4 被弹出. 因为 v_4 为终点,算法结束. 起点 v_1 到终点 v_4 的最小代价即为 20,τ_4 对应的时间区间为[30,35].

3.2.3 正确性证明

下面证明算法 3 的正确性,我们首先给出定理 2 证明当顶点 v_i 从队列中弹出时,在 τ_i 取值对应的区间内,真正的 $g_i(t)$ 的取值即为 τ_i.

定理 2. 给定时间依赖图 G_T,令 v_i 为算法 3 第 k 次迭代过程中从队列 Q 中弹出的顶点,其 τ_i 对应区间为$[t_i^a,t_i^b]$,则顶点 v_i 的到达时间-最小代价函数 $g_i(t)$ 在区间$[t_i^a,t_i^b]$上的取值即为 τ_i.

证明. 只需证明对任意时刻 $t^0\in[t_i^a,t_i^b]$,$g_i(t^0)=\tau_i$ 即可. 因为 $g_i(t)$ 是顶点 v_i 的到达时间-最小代价函数,根据定义,我们可知 $g_i(t^0)\leq\tau_i$. 下面,证明 $\tau_i\leq g_i(t^0)$. 不失一般性,设在时刻 t^0 到达顶点 v_i 且费用代价为 $g_i(t^0)$ 的路径为

$$p:v_s\rightarrow v_1\rightarrow v_2\rightarrow\cdots\rightarrow v_h\rightarrow v_i,$$

因此,沿着该路径 p,到达顶点 $v_x(1\leq x\leq h)$ 的时刻和费用代价分别为 t_x 和 $g_x(t_x)$. 以示区分,我们用 $g_x^k(t_x)$ 表示在第 k 次迭代时顶点 v_x 的到达时间-最小代价函数. 这里需要注意的是,$g_x^k(t)$ 不一定等于顶点 v_x 真正的到达时间-最小代价函数 $g_x(t)$. 我们首先考虑顶点 v_h,在第 k 次迭代时,$g_h^k(t)$ 在时刻 t_h 的取值为 $g_h^k(t_h)$. 考虑两种情况:(1) $g_h^k(t_h)=g_h(t_h)$;(2) $g_h^k(t_h)>g_h(t_h)$. 注意:因为 $g_h(t_h)$ 为最优的到达时间-最小代价函数在时刻 t_h 的取值,则不可能出现 $g_h^k(t_h)<g_h(t_h)$ 的情况. 在情况(1)下,$f_{h,i}(t^0-w_{h,i})$ 又分为两种情况:(a) $f_{h,i}(t^0-w_{h,i})=0$;(b) $f_{h,i}(t^0-w_{h,i})\neq0$. 在情况(a)下,若时刻 t_h 的 $g_h(t)$ 的取值已经被确定,则意味着顶点 v_h 已经在 $\tau_h=g_h^k(t_h)$ 时从队列 Q 中弹出过. 因此,算法 3 已经利用 $g_h^k(t_h)$ 更新过 $g_i(t)$ 的取值. 根据算法 2 的正确性,我们有 $\tau_i\leq g_h^k(t_h)=g_h(t_h)$. 若时刻 t_h 的 $g_h(t)$ 的取值未被确定,则 $\tau_i=g_h^k(t_h)=g_h(t_h)$. 否则,$g_h^k(t_h)<\tau_i$,此时应该弹出顶点 v_h,与条件矛盾. 注意:$g_h^k(t_h)$ 不可能大于 τ_i. 因为 $g_h^k(t_h)=g_h(t_h)$,$g_h(t_h)\leq g_i(t^0)$ 和 $g_i(t^0)\leq\tau_i$,所以 $g_h^k(t_h)\leq\tau_i$. 因此,情况(a)下,我们得出 $\tau_i\leq g_h(t_h)=g_h(t_h)+f_{h,i}(t^0-w_{h,i})=g_i(t^0)$. 我们考虑情况(b),因为 $f_{h,i}(t^0-w_{h,i})\neq0$,所以 $g_h^k(t_h)=g_h(t_h)<g_i(t^0)$. 这意味着顶点 v_h 已经在 $\tau_h=g_h^k(t_h)$ 时从对列 Q 中弹出过. 因此,算法 3 已经利用 $g_h^k(t_h)$ 更新过 $g_i(t)$ 的取值. 根据算法 2 的正确性,我们有

$$\begin{aligned}\tau_i&\leq g_h^k(t_h)+f_{h,i}(t^0-w_{h,i})\\&=g_h(t_h)+f_{h,i}(t^0-w_{h,i})=g_i(t^0).\end{aligned}$$

因此,在情况(1)下,有 $\tau_i\leq g_i(t^0)$. 下面,讨论情况(2),我们证明情况(2)不可能发生. 为证明此事实,我们考虑顶点 v_{h-1},在第 k 次迭代时,$g_{h-1}^k(t_{h-1})$ 同样分为两种情况:$g_{h-1}^k(t_{h-1})=g_{h-1}(t_{h-1})$ 和 $g_{h-1}^k(t_{h-1})>g_{h-1}(t_{h-1})$. 类似证明情况(1)下的 $\tau_i=g_i(t^0)$,我们证明在 $g_{h-1}^k(t_{h-1})=g_{h-1}(t_{h-1})$ 时,有 $g_h^k(t_h)=g_h(t_h)$. 因此,只需证明 $g_{h-1}^k(t_{h-1})>g_{h-1}(t_{h-1})$ 不可能发生. 类似地,我们仅需要证明 $g_{h-2}^k(t_{h-2})>g_{h-2}(t_{h-2})$ 不可能发生,递归地,仅需证明 $g_1^k(t_1)>g_1(t_1)$ 不可能发生. 因为 $g_1(t_1)$ 在第 1 次迭代中,即起点 v_s 从队列 Q 中弹出时已经被计算出,则 $g_1^k(t_1)=g_1(t_1)$. 矛盾. 因此,情况(2)不可能发生. 综上所述,我们得出 $\tau_i\leq g_i(t^0)$. 因此,$\tau_i=g_i(t^0)$. 证毕.

定理 2 说明,每次顶点 v_i 根据 τ_i 的取值从队列 Q 中弹出时,v_i 的到达时间-最小代价函数 $g_i(t)$ 在 τ_i 对应区间上的取值即为 τ_i. 当算法 3 把全部区间上的 $g_i(t)$ 取值确定时,该 $g_i(t)$ 即为真正的 $g_i(t)$.

下面给出定理 3 证明,τ_i 即为顶点 v_i 的真正的 $g_i(t)$ 在区间 T_i-S_i 内的最小取值.

定理 3. 给定时间依赖图 G_T,令 v_i 为算法 3 第 k 次迭代时从队列 Q 中弹出的顶点,$g_i(t)$ 为顶点 v_i 的真正的到达时间-最小代价函数,则此时的 τ_i 即为

$g_i(t)$在区间 T_i-S_i上的最小值.

证明.　反证法.假设存在时刻 $t^0\in T_i-S_i$,使得 $g_i(t^0)<\tau_i$.不失一般性,我们设在时刻 t^0到达顶点 v_i且费用代价为 $g_i(t^0)$的路径为

$$p:\ v_s\rightarrow v_1\rightarrow v_2\rightarrow\cdots\rightarrow v_h\rightarrow v_i.$$

沿着该路径 p,到达顶点 $v_x(1\leq x\leq h)$的时刻和费用代价分别为 t_x和 $g_x(t_x)$.与定理 1 证明类似,我们用 $g_x^k(t)$表示在第 k 次迭代时顶点 v_x的到达时间-最小代价函数. $g_i^k(t^0)$的取值只能分为两种情况:(1) $g_i^k(t^0)=g_i(t^0)$;(2) $g_i^k(t^0)>g_i(t^0)$.在情况(1)下,因为 $g_i^k(t^0)=g_i(t^0)<\tau_i$,所以顶点 v_i会根据 $g_i^k(t^0)$取值弹出,与条件矛盾.因此,我们只需证明情况(2)不会发生.此证明过程与证明定理 1 中情况(2)的过程相同.因为情况(1)下出现矛盾,情况(2)不会发生,所以假设不正确,定理得证.　　证毕.

定理 3 说明了顶点 v_i跟据 τ_i从队列 Q 中弹出时,τ_i即为 $g_i(t)$在未确定取值区间 T_i-S_i内的最小取值.因此,当终点 v_e首次从队列 Q 中弹出时,其 τ_e即为 $g_e(t)$全部区间 T_e上的最小值,即起点 v_s到终点 v_e的最小费用代价 $g_e^*(t^*)$.

推论 1.　给定时间依赖图 G_T,对于顶点 v_i,设 τ_i^p和 τ_i^q为顶点 v_i第 p 次和第 q 次从队列 Q 中弹出时的 τ_i取值,若 $p<q$,则 $\tau_i^p\leq\tau_i^q$.

证明.　令 S_i^p和 S_i^q为顶点 v_i第 p 次和第 q 次从队列 Q 中弹出时的 S_i,因为 $p<q$,所以 $S_i^p\subset S_i^q$.因此,$T_i-S_i^p\supset T_i-S_i^q$.由定理 3,$\tau_i^p$和 τ_i^q分别为 $T_i-S_i^p$和 $T_i-S_i^q$上的最小值,所以,$\tau_i^p\leq\tau_i^q$.　　证毕.

3.3　寻找最优路径和最佳等待时间

下面介绍算法的第 2 阶段,即如何根据最小费用代价 $g_e^*(t^*)$找到起点 v_s到终点 v_e的最优路径 p^*,以及 p^*上的每个顶点的最佳等待时间$\omega^*(v_i)$,使得路径 p^*的代价等于最小费用代价$g_e^*(t^*)$.

算法 4 给出了计算最优路径 p^*和最佳等待时间 $\omega^*(v_i)$的过程.算法 4 的基本思想是:从终点 v_e开始,依次找到顶点 v_i在路径 p^*上的前驱邻居顶点 v_j.初始化阶段,我们令 $v_i\leftarrow v_e$.

算法 4.　路径选择算法.

Path-Selection $(g_e^*(t^*),G_T,v_s,v_e,t_d,t_a)$

输入:$g_e^*(t^*),G_T,v_s,v_e,t_d,t_a$

输出:p^*,p^*上各点等待时间 $\omega^*(v_i)$

1. $v_i\leftarrow v_e$; $p^*\leftarrow\varnothing$; $g_i(t_i)\leftarrow g_e^*(t^*)$; $t_i\leftarrow t^*$;
2. while $v_i\neq v_s$ do
3.　　for each $v_j\in N^-(v_i)$ do
4.　　if $\exists t_j\leq t_i-w_{j,i}, g_i(t_i)=g_j(t_j)+f_{j,i}(t_i-w_{j,i})$ then
5.　　$p^*\leftarrow p^*+v_j$; $\omega^*(v_j)=t_i-w_{j,i}-t_j$;
6.　　$v_i\leftarrow v_j$; $t_i\leftarrow t_j$; break;
7. return $p^*,\omega^*(v_i)$ for each $v_i\in p^*$.

每次迭代过程中,我们计算顶点 v_i在路径 p^*上的前驱邻居顶点 v_j.设在路径 p^*上,到达顶点 v_i的费用代价为 $g_i(t_i)$,到达顶点 v_i的时刻为 t_i.初始化阶段,$g_i(t_i)\leftarrow g_e^*(t^*)$,$t_i\leftarrow t^*$. t^* 可取为 $g_e(t)$上 $g_e^*(t^*)$取值所对应时间区间内的任意一个时刻.对于任意 $v_j\in N^-(v_i)$,即 v_j为 v_i的入边邻居,如果 $\exists t_j, t_j\leq t_i-w_{j,i}$,使得

$$g_i(t_i)=g_j(t_j)+f_{j,i}(t_i-w_{j,i}),$$

则顶点 v_j为顶点 v_i在路径 p^*上的前驱邻居顶点,t_j为路径 p^*上到达顶点 v_j的时刻.这样的 v_j一定存在,因为在算法 3 中,$g_i(t_i)$的取值是根据此 $g_j(t_j)$计算得到的.因此,在顶点 v_j上的最佳等待时间为

$$\omega^*(v_j)=t_j-w_{j,i}-t_j.$$

当算法 4 找到起点 v_s,即 $v_i=v_s$时,算法结束.此时,路径 p^*上的所有顶点和每个顶点上的等待时间均已经计算出来.

我们用图 1 和图 4 中的例子说明该计算过程.从图 4 中,我们得知到达顶点 v_4的最小费用代价为 $g_4^*(t^*)=20$,其对应区间为[30,35],我们任取其中一个时刻 $t^*=30$.因此对于顶点 v_3,离开顶点 v_3的时刻为 $t^*-w_{3,4}=30-10=20$.我们发现在 $t_3=20$时,$g_3(t_3)=15$,则

$$g_3(t_3)+f_{3,4}(t^*-w_{3,4})=g_3(20)+f_{3,4}(30-10)=20=g_4^*(t^*).$$

因此,顶点 v_3为终点 v_4在最优路径 p^*上的前驱邻居顶点,且顶点 v_3上的最佳等待时间 $\omega^*(v_3)=0$.

第 2 次迭代中,因为 $t_3=20$,所以对于顶点 v_2,离开顶点 v_2的时刻为 $t_3-w_{2,3}=20-5=15$.我们发现对于时刻 $t_2=10$,有 $g_2(t_2)=10$,且满足

$$g_2(t_2)+f_{2,3}(t_3-w_{2,3})=10+5=15=g_3(t_3),$$

因此,顶点 v_2为顶点 v_3在路径 p^*上的前驱顶点,且顶点 v_2上的最佳等待时间 $\omega^*(v_2)=5$.

同理,在第 3 次迭代中,我们找到顶点 v_2的前驱邻居顶点 v_1.因为 v_1为起点,所以算法结束.

3.4　时间复杂度和空间复杂度分析

3.4.1　时间复杂度

我们首先分别分析算法 2、算法 3 和算法 4 的时间复杂度,最后给出整体算法 1 的时间复杂度.设图中顶点数量和边的数量分别为 n 和 m,图中顶点 v_i的到达时间-最小代价函数 $g_i(t)$的平均的分段区间数量为 k.

引理 2. 算法 2 的时间复杂度为 $O(k\log k)$.

证明. 算法 2 中,需要计算 $g^*_{a_j}$,所以需要将这 k 个分段区间上的常数取值排序,该过程时间复杂度为 $O(k\log k)$. 计算 φ_{a_j} 只需找到 $g^*_{a_j}$ 对应时间区间的左端点即可,该操作需要 $O(1)$ 的时间. 在计算 $g_{j\to i}(t)$ 过程中,需要计算出 $g_{j\to i}(t)$ 在全部区间 $[\lambda_{j\to i},t_a]$ 内的不同分段上的取值. 因此,计算 $g_{j\to i}(t)$ 的过程时间复杂度为 $O(k)$. 同样地,更新 $g_i(t)$ 的时间复杂度也为 $O(k)$. 因此,算法 2 的时间复杂度为 $O(k\log k)$. 证毕.

引理 3. 算法 3 的时间复杂度为 $O(kn\log n+mk^2\log k)$.

证明. 算法 3 中的每次迭代过程中,最多有 n 个顶点同时被维护于队列 Q 中. 我们利用 Fibonacci 堆[12],则将顶点 v_i 弹出队列的操作需要花费 $O(\log n)$ 时间,而插入队列的操作需要 $O(1)$ 时间. 当顶点 v_i 从队列 Q 中弹出后,需要更新 v_i 的出边邻居 v_j 的到达时间-最小代价函数 $g_j(t)$. 令 $d^+(v_i)$ 为顶点 v_i 的出度,则这一操作的时间复杂度为 $O(d^+(v_i)k\log k)$. 因此,算法 3 中一次迭代过程的时间复杂度为 $O(\log n+d^+(v_i)k\log k)$. 由定理 3 和推论 1 可知,算法 3 依据 $g_i(t)$ 在不同分段的取值大小依次弹出顶点 v_i. 最坏情况下,顶点 v_i 被弹出的次数为 $O(k)$ 次. 因此,算法 3 的时间复杂度为

$$\begin{aligned}&O\Big(\sum_{v_i\in V}k(\log n+d^+(v_i)k\log k)\Big)\\&=O\Big(k\Big(\sum_{v_i\in V}(\log n+d^+(v_i)k\log k)\Big)\Big)\\&=O(k(n\log n+mk\log k))\\&=O(kn\log n+mk^2\log k).\end{aligned}$$

证毕.

引理 4. 算法 4 的时间复杂度为 $O(mk)$.

证明. 算法 4 的每次迭代过程中,对一个顶点 v_i,需要检查其所有入边邻居 v_j. 对每一个入边邻居 v_j,需要检查其到达时间-最小代价函数 $g_j(t)$ 在 $t\le t_i-w_{j,i}$ 时,是否存在取值满足公式 $g_i(t_i)=g_j(t_j)+f_{j,i}(t_i-w_{j,i})$,该操作的时间复杂度为 $O(k)$. 因此,每一次迭代中,对顶点 v_i,计算其前驱邻居顶点的时间复杂度为 $O(d^-(v_i)k)$. 因为最优路径 p^* 上不存在环,对图中任意一个顶点 v_i 最多计算一次其前驱邻居顶点,所以,算法 4 的时间复杂度为 $O\Big(\sum_{v_i\in V}(d^-(v_i)k)\Big)=O(mk)$. 证毕.

定理 4. 算法 1 的时间复杂度为 $O(kn\log n+mk^2\log k)$. 算法 1 是计算起点 v_s 到终点 v_e,具有时间限制的费用最优路径 p^* 和 p^* 上每个顶点 v_i 的最佳等待时间 $\omega^*(v_i)$ 的算法.

证明. 因为算法 3 和算法 4 是算法 1 的两个阶段,所以根据引理 3 和引理 4,我们可知算法 1 的时间复杂度为 $O(kn\log n+mk^2\log k)$. 证毕.

3.4.2 空间复杂度

定理 5. 算法 1 的空间复杂度为 $O((n+m)k)$.

证明. 算法 3 和算法 4 为算法 1 的两个阶段. 在优先队列 Q 中,算法 3 至多维护 n 个顶点. 对每个顶点 v_i,算法需要维护其到达时间-最小代价函数 $g_i(t)$. 对图中每条边,算法 3 和算法 4 需要维护其费用代价函数 $f_{i,j}(t)$. 因此,算法 1 的空间复杂度为 $O((n+m)k)$. 证毕.

4 实验结果和分析

我们在真实的数据集上实现了本文的算法,并与 TCSP-AWT 算法[7]进行了对比. TCSP-AWT 算法是目前解决具有时间限制的费用最优路径查询的最有效的算法,TCSP-AWT 算法用于计算离散时间模型下的费用最优路径. 所有实验均在主频为 2.5 GHz的 Intel Core i5 CPU 和内存为 4 GB 的 PC 机上完成. 我们使用的操作系统为 Windows 7.

4.1 数据集描述和实验设置

我们在以下真实数据集上进行了算法的测试:

California road network. 本数据集描述的是美国加利福尼亚州道路交通网,包括 21 047 个顶点和 21 692 条边. 其中,顶点代表道路的交叉口或者道路的端点,边代表道路片段. 本数据来自 http://www.maproom.psu.edu/dcw.

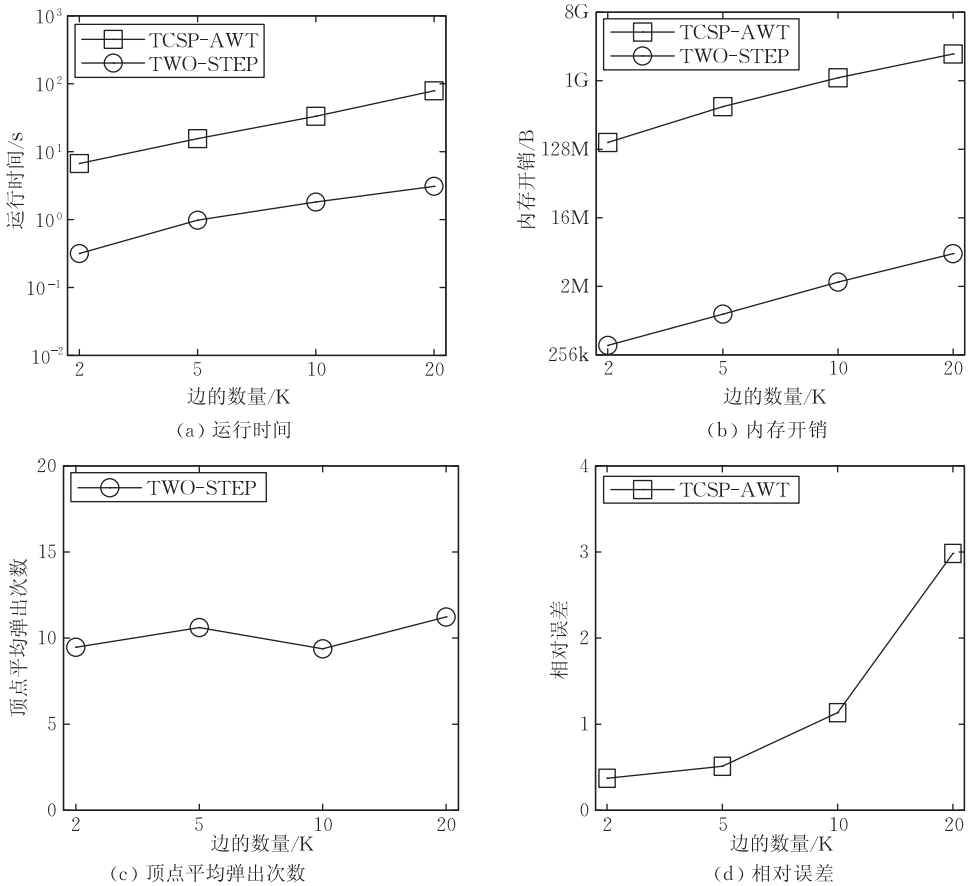
我们在该数据集上生成了 4 个规模不同的时间依赖图,其顶点规模分别为 2K,5K,10K 和 20K. 在这些时间依赖图上分别进行了算法测试,以考察算法的可扩展性. 针对数据集中的每一条边,我们随机生成其时间代价 $w_{i,j}$ 和费用代价函数 $f_{i,j}(t)$. 对于 $f_{i,j}(t)$,限定其定义域为 $[0,2000]$. 2000 表示 2000 个时间单位(秒或者分). 将 $f_{i,j}(t)$ 的定义域随机分成 k 个片段,在每一个片段上赋予一个常数取值. 因此,$f_{i,j}(t)$ 为一个分段常量函数. 对算法 TCSP-AWT,我们每隔一个时间单位采样一个离散时间点. 例如,对查询区间 $[0,1000]$,采样的离散时间点个数即为 500 个. 对每组数据集,分别生成规模为 1000 的查询集合,实验中的结果,是每个查询集合结果的平均值. 用 TWO-STEP 标记我们的算法.

我们在实验中考察了以下几方面的内容:

(1)图中顶点数量对算法性能的影响；(2)图中边的数量对算法性能的影响；(3)起点 v_s 和终点 v_e 之间距离对算法性能的影响；(4)最早出发时刻和最晚到达时刻的时间区间 $[t_d, t_a]$ 对算法性能的影响；(5)费用代价函数 $f_{i,j}(t)$ 上分段区间数量 k 对算法性能的影响. 算法性能考察的指标为：(1)算法运行时间；(2)算法运行时的内存开销；(3)每个顶点从队列中弹出的平均次数；(4)TCSP-AWT 算法的相对误差，我们将其定义为 $(c-c^*)/c^*$，这里，c 为TCSP-AWT算法结果的代价，c^* 为最优结果的代价. 我们的算法基于连续时间模型，所以不存在误差.

4.2　实验结果

实验 1. 顶点数量对算法性能的影响. 在图 5 中，我们考察顶点数量对算法性能的影响. 本组实验中，边的费用代价函数的分段数量为 $k=10$，最早出发时刻和最晚到达时刻区间 $T=[0, 1000]$，TCSP-AWP 算法的时间点个数 $l=500$. 如图 5(a)和图 5(b)所示，我们的算法的运行时间和内存开销均远远小于 TCSP-AWT 算法. 这是因为 TCSP-AWT 算法需要计算在每个时刻到达图中每个顶点的代价，并对这些代价进行组合以求得最小代价. 我们的算法比 TCSP-AWT 算法快了 20 倍，内存开销少了近 500 倍. 我们发现，算法的运行时间和内存开销都随着顶点数量的增加而增大. 在图 5(c)中，我们发现每个顶点从队列中弹出的平均次数与顶点数量变化无关. 在图 5(d)中，我们发现 TCSP-AWT 算法的相对误差随着顶点数量的增加而增大. 这是因为，顶点规模变大时，起点和终点之间的距离增大. 因此，在路径上累积的误差值就会变大.

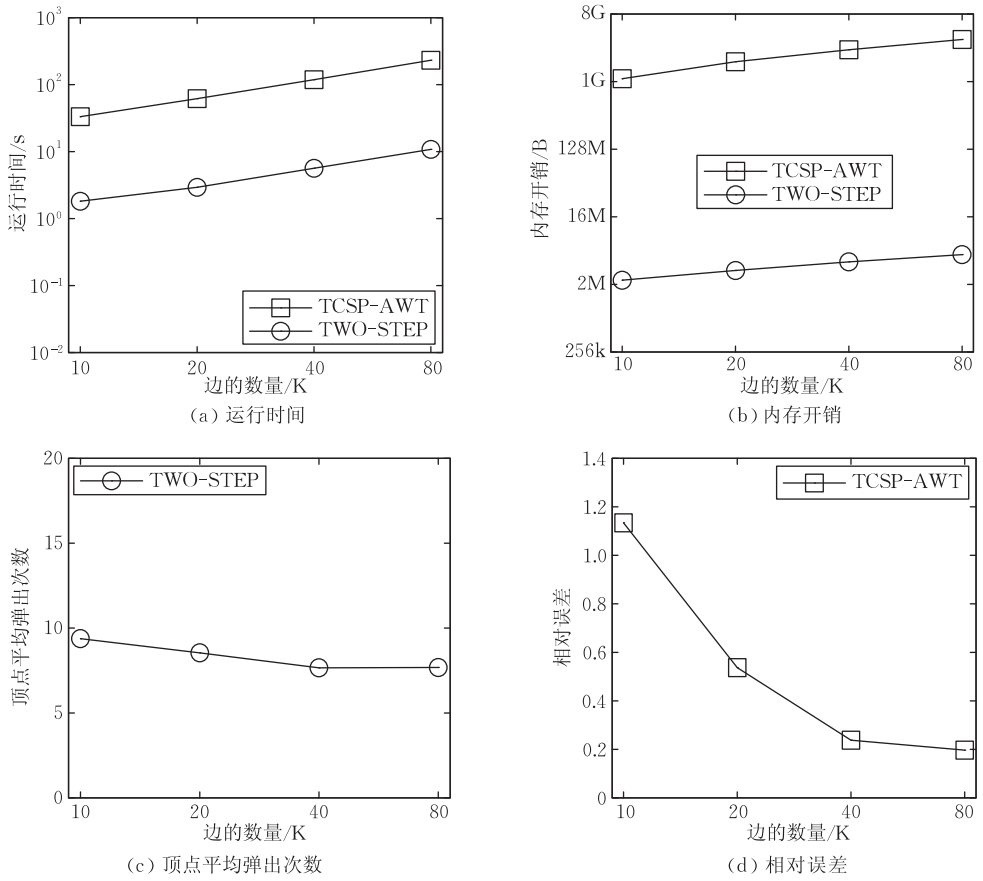


(a) 运行时间　(b) 内存开销　(c) 顶点平均弹出次数　(d) 相对误差

图 5　顶点数量对算法性能的影响

实验 2. 边的数量对算法性能的影响. 图 6 中，我们考察图中边的数量对算法性能的影响. 本组实验中，我们固定图中顶点数量为 10 K，边的数量从 10 K 变化到 80 K，费用代价函数的分段数量 $k=10$，最早出发时刻和最晚到达时刻区间 $T=[0,1000]$，TCSP-AWP 算法的时间点个数 $l=500$. 通过图 6(a)和图 6(b)，我们同样发现，我们的算法在时间和空间上都要优于 TCSP-AWT 算法. 图 6(c)中，我们发现，随着边的数量的增加，每个顶点从队列中弹出的平均次数有所减少. 这是因为，边的数量增加导致图的密度增大，进而图中任意两点之间的距离减少. 因此，算法可以更快地计算出到达终点的最小费用，

从而减少了每个顶点的弹出次数.然而,边的数量增加,导致了顶点的邻居数增加,当顶点从队列弹出时,更新其所有邻居到达时间-最小代价函数的时间开销也就增大.因此,在图 6(a)中,算法运行时间会随着边的数量的增加而增加.图 6(d)中,我们发现 TCSP-AWT 算法的相对误差随着边的增加而减小.这是因为,边的增加使得起点和终点之间的距离减小,所以,路径上累积的误差也会减小.



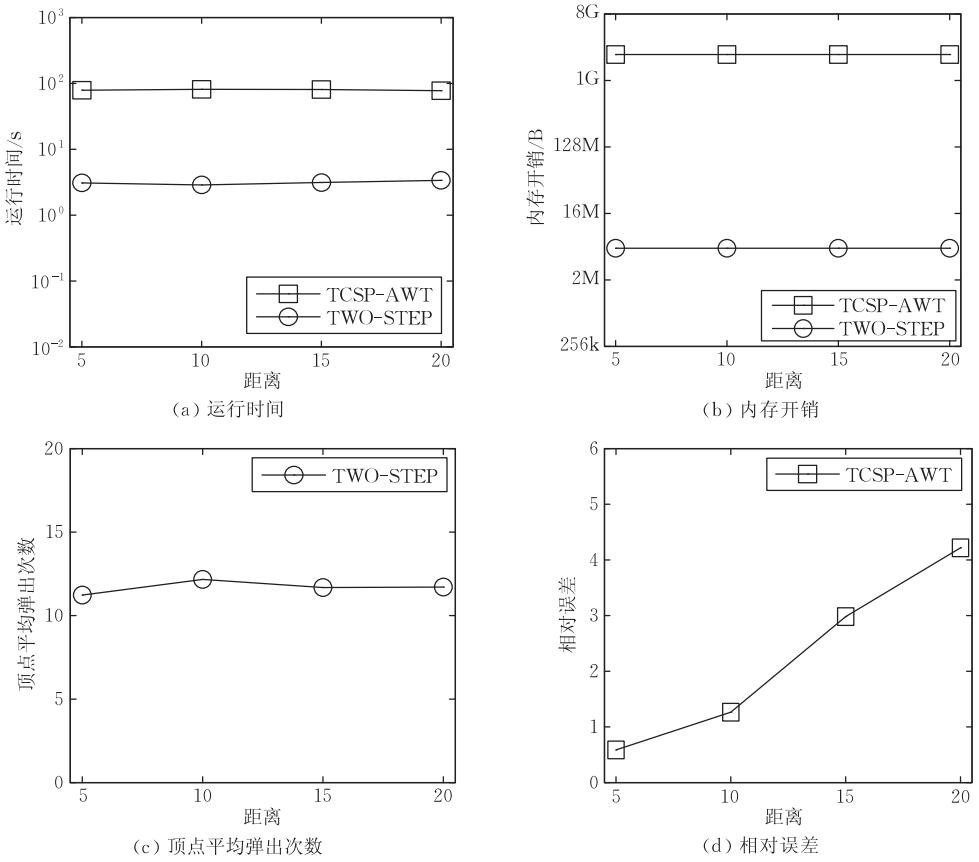
(a) 运行时间 (b) 内存开销 (c) 顶点平均弹出次数 (d) 相对误差

图 6 边的数量对算法性能的影响

实验 3. 起点和终点之间距离对算法性能的影响.图 7 中,我们研究了起点与终点之间距离变化对算法性能的影响.本组实验中,图中顶点数量为 20K,起点与终点之间的距离从 5 变化到 10.费用代价函数的分段数量 $k=10$,最早出发时刻和最晚到达时刻区间 $T=[0,1000]$,TCSP-AWP 算法的时间点个数 $l=500$.通过图 7(a)和图 7(b),我们发现,算法的运行时间和内存开销对起点和终点之间距离的变化并不敏感.这是因为,起点和终点之间距离最短的路径在很多情况下并不是费用代价函数下最优的路径.与边的数量变化不同,本组实验中,图的密度没有增加,仅仅是起点和终点之间的距离发生变化.因此,在图 7(c)中,顶点从队列中弹出的平均次数也没有显著变化.在图 7(d)中,我们发现 TCSP-AWT 算法的相对误差随着起点和终点之间距离的增大而增大,其原因是,随着起点与终点之间距离增大,路径上的累积误差也会增大.

实验 4. 查询时间区间的距离对算法性能的影响.图 8 中,我们考察了查询时间区间对算法性能的影响.本组实验中,图中顶点数量为 10 K,最早出发时刻固定为 0,最晚到达时刻从 600 变化到 1200,因此,TCSP-AWP 算法的时间点个数从 300 变化到 600.费用代价函数的分段数量 $k=10$.图 8(a)和图 8(b)中,我们的算法时间和空间开销受查询时间区间变化的影响很小.这是因为,虽然查询时间区间变大,但费用代价函数分段数量没有改变.相反地,因为查询时间区间变大,导致 TCSP-AWT 算法的离散时间点增多,所以,TCSP-AWT 算法的时间和空间开销都增加.图 8(c)中,每个顶点的平均弹出次数没有受到明显影响.图 8(d)中,我们发现,TCSP-AWT 算法的相对误差也没有明显变化.

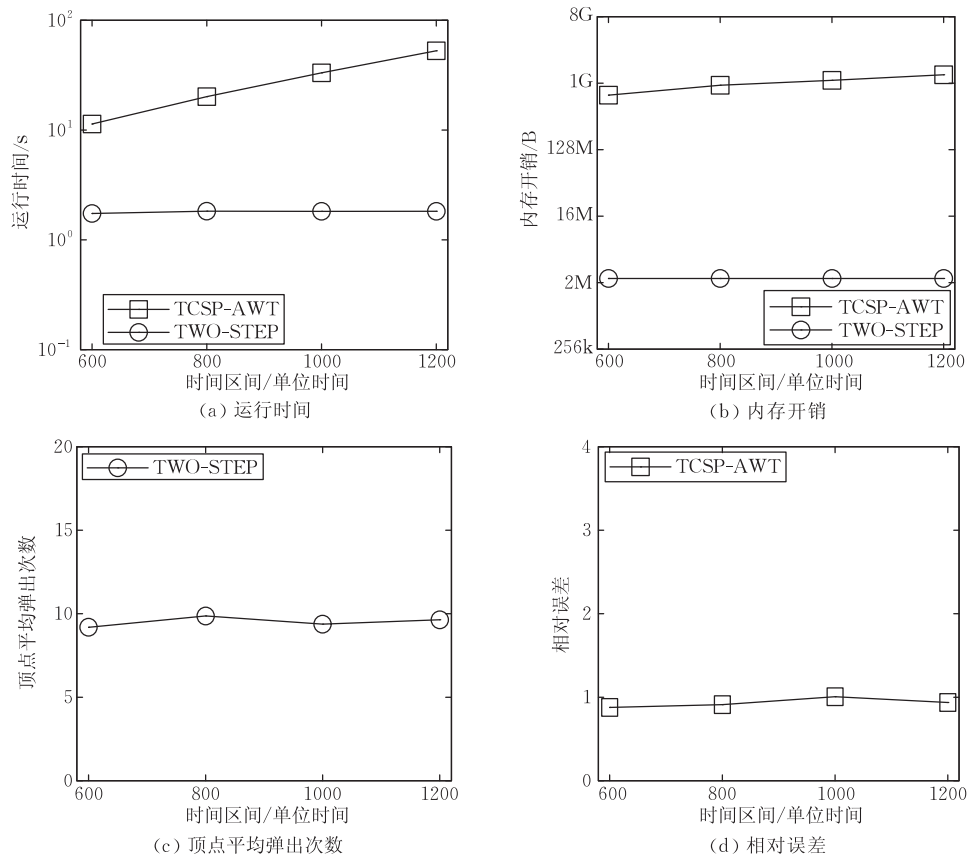
实验 5. 费用代价函数的分段区间数量对算法性能的影响.在图 9 中,我们研究了费用代价函数分段区间数量对算法性能的影响.本组实验中,图中顶



(a) 运行时间　(b) 内存开销

(c) 顶点平均弹出次数　(d) 相对误差

图 7　起点和终点间距离对算法性能的影响

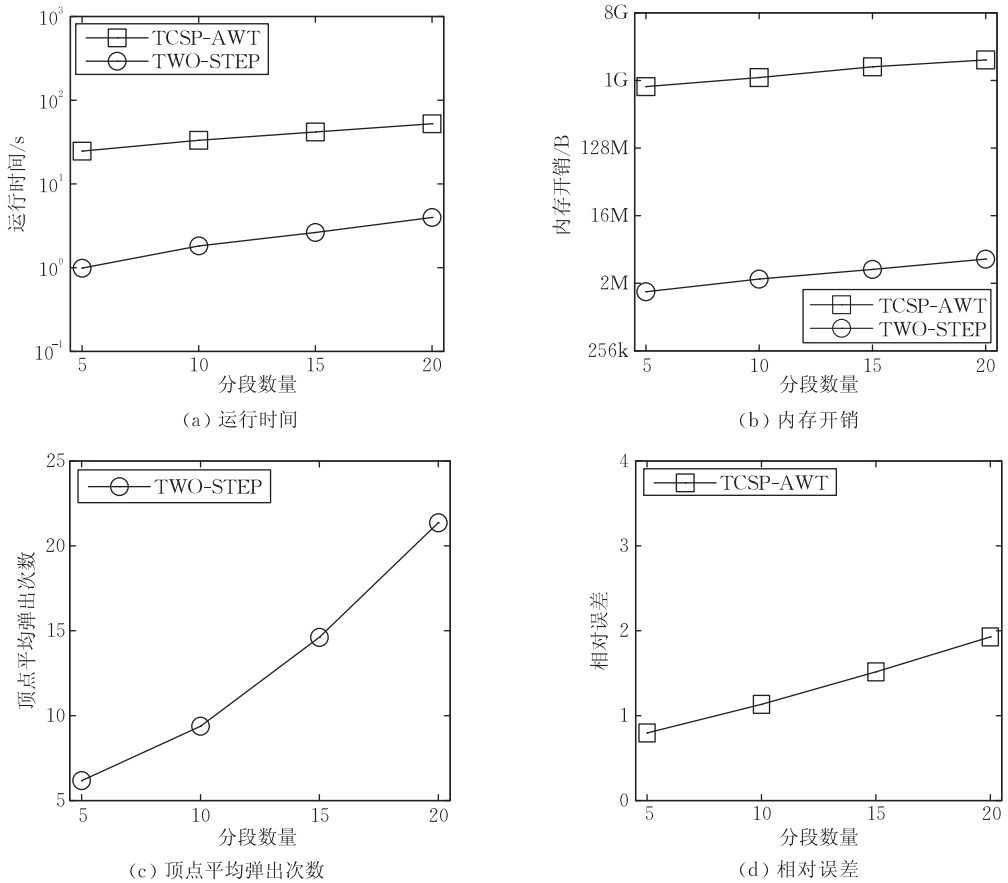


(a) 运行时间　(b) 内存开销

(c) 顶点平均弹出次数　(d) 相对误差

图 8　查询时间区间对算法性能的影响

点数量为 10K，边上的费用代价函数的分段区间数量从 5 变化到 20，最早出发时刻和最晚到达时刻区间 $T=[0,1000]$，TCSP-AWP 算法的离散时间点的个数 $l=500$. 通过图 9(a)和图 9(b)，我们发现，算法的运行时间和内存开销均随着函数分段区间数量的增加而增加. 这是因为，当函数分段区间数量增加时，更新图中顶点到达时间-最小代价函数的开销增大. 我们的算法在时间和空间上都要优于 TCSP-AWP 算法. 通过图 9(c)，我们发现，每个顶点从队列中的弹出次数随着函数分段数量的增加而增加. 这是因为，当费用函数分段区间数量增加时，顶点上的到达时间-最小代价函数的分段数量也会增加. 所以，顶点从队列中弹出次数也就会增加. 在图 9(d)中，我们发现 TCSP-AWT 算法的相对误差也随着函数分段数量的增加而增加. 这是因为，当分段数量增加时，费用代价函数的取值数量也会增加，所以，费用代价函数取值发生变化时刻位于两个离散时间点之间的机会也就增加. 当到达某个顶点时，TCSP-AWT 算法无法找到通行下一条边的最优代价的机会也就增加. 因此，TCSP-AWT 算法的相对误差会随着函数分段区间数量的增加而增加.



(a) 运行时间 (b) 内存开销

(c) 顶点平均弹出次数 (d) 相对误差

图 9 费用代价函数分段数量对算法性能的影响

5 相关工作

目前，关于时间依赖图上的最短路径查询问题(TDSP)已经有了很多的研究工作[1-6,13-15]. 然而，这些工作主要解决的是最短旅行时间问题，即在时间依赖图模型上找到一条起点到终点的路径，使得该路径的时间代价总和最小. 这些工作主要分为两类，一类基于离散的时间模型，另外一类基于连续的时间模型. 文献[13-15]研究了离散时间模型下的时间依赖最短路径查询. 文献[14]中，图上每一条边被赋予一组聚集属性，该聚集属性表示了这条边在不同时刻所呈现的状态. 文献[13]将出发时间区间平滑地离散为 k 个时间点，并依据这 k 个时间点将图中顶点和边复制 k 次，因此，时间依赖图被转化了一个规模增大 k 倍的静态图. 文献[13]进而给出了如何在这个静态图上完成最短路径查询的方法. 这些基于离散时间模型的方法并不能回答连续时间模型下的最短路径查询问题. 文献[1-6]研究了连续时间模型下的最短路径查询问题. 文献[1]提出了一种基于

Bellman-Ford 算法思想的方法. 对图中任意顶点 v_i，该方法迭代的计算到达顶点 v_i 的最早时刻，并利用这些中间结果，最终计算出到达终点 v_e 的最早时刻. 最后，根据该时刻找到旅行时间最短的路径. 文献[2]提出了道路网中的速度流模型，该模型下，图中每条边的通过速度可以用一个分段线性的函数表示. 该文作者提出一个基于 Dijkstra 算法思想的方法计算速度流模型下的最快路径查询问题. 文献[3]提出了一种基于 A* 算法的扩展算法. 该算法的主要思想是用一个优先队列维护所有待扩展的路径. 对任意一点 v_i，起点 v_s 到 v_i 的最早到达时间函数被维护在优先队列中，该算法通过顶点 v_i 到终点 v_e 的欧氏距离和网络中的最大速度，预估顶点 v_i 到终点 v_e 的时间. 算法根据预估时间弹出队列元素，并找到起点到终点的最短旅行时间路径. 文献[4]介绍了如何应用数据挖掘的方法找到各种条件下描述道路通过速度的模型. 文献[5]是目前最有效的解决 TDSP 问题的方法. 该方法采用两阶段搜索策略. 第 1 阶段，用优先队列更新图中所有顶点最早到达时间的函数，并最终计算出终点 v_e 的最早到达时间函数. 第 2 阶段，根据 v_e 的最早到达时间函数找到旅行时间最小的路径. 然而，这些解决 TDSP 问题的研究工作都利用了以下性质：到达顶点 v_i 的最早时刻可以根据到达其入边邻居的最早时刻计算得出. 然而，在计算具有时间限制的费用代价最优路径时，该性质并不成立. 因此，目前解决 TDSP 问题的方法均不能解决本文面对的问题.

运筹学领域中的一些工作研究了时间依赖图上具有时间限制的费用代价最优路径的查询问题[7-11]. 然而，这些工作假定的时间模型是离散的. 在离散时间模型中，时间区间被离散化为时间点的集合$[t_1, t_2, \cdots, t_T]$. 对图上任意一条边(v_i, v_j)，用户只能选择在某个给定时刻 t_i 离开顶点 v_i. 这些工作的主要问题有：(1)离散时间模型下不能找到精确的最优解；(2)这些方法只是从理论上给出计算最优路径的方法，并没有从提高算法实际运行效率的角度来设计算法和优化算法，也没有应用数据处理中的优化技术. 所以，这些方法需要承受较高的时间和空间开销. 因此，运筹学领域中的这些工作也不能很好地解决本文所要面对的问题.

6　结　论

本文中，研究了时间依赖图上具有时间限制的费用代价最优路径的查询问题. 我们首先给出了时间依赖图模型的定义以及时间依赖图上具有时间限制的费用代价最优路径查询的定义. 然后，给出了一个有效的算法计算时间限制下的费用代价最优路径. 同时，证明算法的时间复杂度和空间复杂度分别为 $O(kn\log n + mk^2\log k)$ 和 $O((n+m)k)$. 最后，通过真实数据集上的实验验证了算法的有效性.

在后续的工作中，我们将考虑时间代价函数和费用代价函数共同作用下的最优路径查询问题，以及在时间依赖图模型上多维代价的最优路径查询问题.

参　考　文　献

[1] Orda A, Rom R. Shortest-path and minimum delay algorithms in networks with time-dependent edge-length. Journal of ACM, 1990, 37(3): 607-625

[2] Sung K, Bell M G, Seong M, Park S. Shortest paths in a network with time-dependent flow speeds. European Journal of Operational Research, 2000, 123(12): 32-39

[3] Kanoulas E, Du Y, Xia T, Zhang D. Finding fastest paths on a road network with speed patterns//Proceedings of the 22th IEEE International Conference on Data Engineering. Atlanta, Georgia, USA, 2006: 10-19

[4] Gonzalez H, Han J, Li X, Myslinska M, Sondag J P. Adaptive fastest path computation on a road network: A traffic mining approach//Proceedings of the 25th International Conference on Very Large Database. Vienna, Austria, 2007: 794-805

[5] Ding B, Yu J X, Qin L. Finding time-dependent shortest paths over large graphs//Proceedings of the 12th International Conference on Extending Database Technology. Nantes, France, 2008: 205-216

[6] Dehne F, Omren M, Jorgudiger J. Shortest paths in time-dependent FIFO networks using edge load forecasts//Proceedings of the 2nd International Workshop on Computational Transportation Science. Seattle, Washington, USA, 2009: 1-6

[7] Cai X, Kloks T, Wong C K. Shortest path problems with time constraints. Networks, 1997, 29(3): 141-150

[8] Dean B C. Algorithms for minimum-cost paths in time-dependent networks with waiting policies. Networks, 2004, 44(1): 41-46

[9] Nasrabadi E, Hashemi S M. On solving dynamic shortest path problems//Proceedings of the 20th Euro Continuous Optimization and Knowledge-Based Technologies. Neringa, Lithuania, 2008: 48-53

[10] Hashemi S M, Mokarami S, Nasrabadi E. Dynamic shortest path problems with time-varying costs. Optimization Letters, 2010, 4(1): 147-156

[11] Kluge S, Brokate M, Reif K. New complexity results for time-constrained dynamical optimal path problems. Journal of Graph Algorithms and Applications, 2010, 14(2): 123-147

[12] Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. 2nd Edition. Boston: The MIT Press,

[13] Chabini I. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. Transportation Research Record, 1998, 1645: 170-175

[14] George B, Shekha S. Time-aggregated graphs for modeling spatio-temporal networks. Journal on Data Semantics XI, 2008, 4231: 191-212

[15] Nachtigall K. Time depending shortest-path problems with applications to railway networks. European Journal of Operational Research, 1995, 83(1): 154-166

YANG Ya-Jun, born in 1983, Ph. D. candidate. His main research interests include graph mining, graph database and massive data management.

GAO Hong, born in 1966, professor, Ph. D. supervisor. Her research interests include wireless sensor networks, cyber-physical systems, massive data management and data mining etc.

LI Jian-Zhong, born in 1950, professor, Ph. D. supervisor. His research interests include database, massive data processing, wireless sensor networks, cyber-physical systems etc.

Background

Graphs have been widely used to model complex relationships among various entities in many real-life applications. Shortest path query is an important problem in graphs and has been well-studied on static graphs. However, in real applications, the costs of edges in graphs always change over time. We call such graphs as time-dependent graphs. In this paper, we study how to find the optimal path with the minimum cost under time constraint on large time-dependent graphs. Most existing works about time-dependent shortest path problem (TDSP) focus on finding the shortest path with the minimum travel time. All these works utilize the following property: the earliest arriving time of vertex v can be computed by the earliest arriving time of v's neighbors. Unfortunately, this property does not hold in our problem. In this paper, we propose a novel algorithm to compute the optimal path with the minimum cost under time constraint. We show the time and space complexity are $O(kn\log n+mk^2\log k)$ and $O((n+m)k)$ respectively.

This work is supported in part by the State Key Development Program of Basic Research of China, the Key Problem of National Natural Science Foundation of China, the NSF of China and the NSFC-RGC of China. These foundations focus on the research of various areas of data intensive computing. Our group has been working on the research of database for many years, and many high quality papers have been published in worldwide conferences and transactions, such as SIGMOD, VLDB, ICDE, KDD, INFOCOM, TKDE, VLDB Journal et al.