Advances in Applied Mathematics 应用数学进展, 2021, 10(12), 4559-4572
Published Online December 2021 in Hans. http://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2021.1012486

Hans汉斯

时变网络中最短路径的故障修复问题

薛小红，张淑蓉

太原理工大学，数学学院，山西 太原

收稿日期：2021年11月27日；录用日期：2021年12月17日；发布日期：2021年12月31日

摘　要

在交通和通信网络中，最短路径及其相关优化问题具有广阔的应用前景。而基于大规模网络的时变特性，时变最短路问题被逐渐关注并且进行了广泛研究。另外，为保证传输性能，当最短路径中存在链路或节点故障时，就需要对该路径进行快速恢复和重建，即需要根据故障影响范围制定切实可行的恢复机制使网络正常运行。进一步，为了提高网络服务质量，需要尽量降低路径恢复成本。因此，本文对时变网络中含有一条故障边的最短路径快速修复问题进行了研究，并提出了有效算法。该算法不仅保证了时变网络中沿路径经过时边赋权函数的时间连续性，而且充分利用了时变最短路子图，在保证传输连续性的基础上减少计算复杂度。最后得出，对于具有n个顶点和m条边的时变网络，算法的时间复杂度为$O\left(\alpha(T)\cdot\left[\|\delta\|_t+|\delta|_t\cdot\log|\delta|_t\right]\right)$，其中$|\delta|_t$表示路径所有出发时间区间内受影响区间点数目的最大值，$\|\delta\|_t$表示所有出发时间区间内受影响区间点和受影响区间边总数的最大值。

关键词

故障恢复，时变网络，旅行时间，最短路径

Fault Repair of the Shortest Path in the Time-Varying Networks

Xiaohong Xue, Shurong Zhang

College of Mathematics, Taiyuan University of Technology, Taiyuan Shanxi

Received: Nov. 27th, 2021; accepted: Dec. 17th, 2021; published: Dec. 31st, 2021

Abstract

In transportation and communication networks, the shortest path and its related optimization problems have broad application prospects. Based on the time-varying feature of large-scale networks, the time-varying shortest path problem has been gradually concerned and widely studied.

文章引用: 薛小红, 张淑蓉. 时变网络中最短路径的故障修复问题[J]. 应用数学进展, 2021, 10(12): 4559-4572.
DOI: 10.12677/aam.2021.1012486

In addition, in order to ensure the transmission performance, when there is a link or node failure in the shortest path, it is necessary to quickly recover and reconstruct the path, that is, it is necessary to formulate a feasible recovery mechanism according to the occurrence of failure to make the network operate normally. Further, in order to improve the quality of network service, it is necessary to minimize the cost of path recovery. Therefore, this paper studies the shortest path fast repair problem with a fault edge in time-varying networks, and proposes an effective algorithm. The algorithm not only ensures the time continuity between edges along the path in the time-varying network, but also makes full use of the time-varying shortest path subgraph to reduce the computational complexity. Finally, for the time-varying network with *n* vertices and *m* edges, the time complexity of the algorithm is $O\left(\alpha(T)\cdot\left[\|\delta\|_t+|\delta|_t\cdot\log|\delta|_t\right]\right)$, in which $|\delta|_t$ represents the maximum number of affected interval nodes in all departure time intervals of the path, and $\|\delta\|_t$ represents the maximum number of affected interval nodes and affected interval edges in all departure time intervals.

Keywords

Fault Recovery, Time Varying Network, Travel Time, Shortest Path

1. 引言

图论是建立网络拓扑模型的基本数学工具，在加权图中寻找最短路径是图论中最基本的问题之一，该问题在通信网络、工厂设施布局以及超大规模集成电路设计中得到了广泛的应用[1] [2] [3] [4]。该问题一般用来求解两顶点之间或从单一源点到其他所有点的最短路径。根据边权值是否为负，可以通过Dijkstra 算法或 Bellman-Ford 算法求解[5]。

然而当故障发生时，如何提高网络生存性并保持服务连续性是网络界面临的一个重大挑战。通常情况下，为了降低路径恢复成本(时间或能耗)，设计路径恢复方案成为亟待解决的问题。如果在两个指定节点之间存在多条路径，则可将其中一条路径视为主动路径，而另外的一条路径可当作备份路径。然而额外构造起点到终点之间的备份路径通常会产生较高的时间复杂度。因此，局部修复的重要性尤为突出。即如果在某个时间点或者某段时间内最短路径上的一个或者多个链路发生故障时，那么从链路故障处开始寻找局部备份路径可以有效降低路径恢复时间。

事实上，当故障发生时可视为将故障边的权值调整为无穷大，此过程通常被称为图更改，而该类问题的更一般化版本包含权值增加、权值减少、边插入和边删除。后两种变化在概念上可以考虑为权值变化的特殊情况。因此，如果变化过程只包含权值的增加和边的删除，被称为增量问题；如果它只包含权值减小和边插入，被称为递减问题，这两类问题都属于半动态问题。当变化过程同时包含权值增加和权值减小，则被称为完全动态问题。如果每次只有一条边发生改变，称为单边问题，而如果多条边同时改变，则称为批处理问题。关于图中的这种动态最短路径问题已有大量的研究[6]-[13]。[1] [8] [12] [14]提出了几种算法来解决单变问题。在[1] [11] [15]中考虑了批处理问题。其中，文献[12]中，Ramalingam 和 Reps 根据自适应参数 $\|\delta\|$ 来分析增量算法，该值可以衡量图改变规模的大小，并提出运行时间为 $O\left(\|\delta\|+|\delta|\log|\delta|\right)$ 的有效算法。同样，文献[14]给出了复杂性为 $O\left(\sqrt{m}\log n\right)$ 的有效算法，其中 n 为顶点数，

m 为边数。

在时变网络中，静态网络的最短路算法和路径恢复方案均不再适用，因此在时变网络中的路径优化和恢复有很重要的研究意义。Li 等人[16]在 2019 年提出了一种在线近似技术 AT-Dijkstra 和自下而上压缩方法，将跳标记方法(hop-labeling)扩展到时变环境中，并给出了相应的查询算法。Wang 等人[17]设计了一个颇为有效的时变最短路径查询算法(TDSP)，用于查找任意出发时刻的最优路径，该算法的时间复杂度是 $O\left(\log^2 k_f \cdot |V| \cdot \log^2 \alpha(T)\right)$，其中 $\alpha(T)$ 表示总时间范围 $[0,T]$ 被分为的子时间段个数。之后，基于 TDSP 查询算法，他们给出了一个时间间隔最短路径查询(TIP)算法，用于查找出发时间在一个固定时间段内的最优路径并输出最优出发时刻，该算法的时间复杂度是 $O\left(\log^2 k_f \cdot |V| \cdot \alpha(I)^2\right)$。基于此，本文将进一步研究时变最短路径的快速恢复方案并给出算法框架，该算法时间复杂度为 $O\left(\alpha(T) \cdot \left[\|\delta\|_t + |\delta|_t \cdot \log|\delta|_t\right]\right)$。

本文主要使用了时变自适应参数 $\|\delta\|_t$ 度量在 t 时刻出发时网络规模的变化量并在局部范围内对时变路径进行修复，提出了在时变网络中修复最短路径的有效方法，通过选择受影响区间点和受影响区间边减少了计算复杂度，进一步寻找到故障修复后的时变最短路。

在给出主要算法的过程中，重点研究了时变赋权函数的应用，路径修复的时间连续性以及有效的时间段划分，同时充分利用了自适应参数降低算法复杂度，并进一步说明了局部修复算法的有效性和应用价值。

2. 模型概述

一个时变的网络通常都被模型化为一个时变的有向图 $G(V,E,W,T)$。

- V 是 G 的顶点集，$|V| = n$；
- E 是边集，$|E| = m$；
- $W = \{w_e(t) : e \in E\}$，其中 $w_e(t) : [0,T] \to R^+$ 表示边 e 关于出发时间 t 的时变旅行函数；
- T 是给定的出发时间上界。

本文考虑的时变是边权值随时间连续变化的情况，且边的权值表示边的经过时间。因此，在本文中，边权函数被假定为是一个非负的连续线性分段函数。

在该时变网络中，v_1v_n-路径被表示为 $P_{v_1v_n}(t) = \langle v_1, v_2, \cdots, v_n \rangle$，表示在 t 时刻出发时从 v_1 到 v_n 的一条路径。其中 v_1 是路 $P_{v_1v_n}(t)$ 的源点，v_n 是路 $P_{v_1v_n}(t)$ 的汇点。

$P_{v_1v_{n-1}}(t)$ 表示从 v_1v_n-路 $P_{v_1v_n}(t)$ 中删除点 v_n 时形成的一条 v_1v_{n-1} 子路径。

现在我们定义路的旅行时间函数和到达时间函数。给定路 $P_{v_1v_{n-1}}(t)$，该路的旅行时间函数和到达时间函数分别表示为 $w_{P_{v_1v_n}}(t)$ 和 $l_{P_{v_1v_n}}(t)$。

- $w_{P_{v_1v_n}}(t)$ 被递归地定义为：

$$w_{P_{v_1v_n}}(t) = w_{P_{v_1v_{n-1}}}(t) + w_{(v_{n-1},v_n)}\left(l_{P_{v_1v_{n-1}}}(t)\right),$$

其中 $w_{P_{v_1v_n}}(t)$ 表示在 t 时刻出发时沿路径 $P_{v_1v_{n-1}}(t)$ 的旅行时间加上边 (v_{n-1},v_n) 的旅行时间。

- $l_{P_{v_1v_n}}(t)$ 被递归地定义为：

$$l_{P_{v_1v_n}}(t) = l_{P_{v_1v_{n-1}}}(t) + w_{(v_{n-1},v_n)}\left(l_{P_{v_1v_{n-1}}}(t)\right),$$

其中 $l_{P_{v_1v_n}}(t)$ 表示在 t 时刻出发时沿路径 $P_{v_1v_{n-1}}(t)$ 到达 v_{n-1} 的到达时间加上边 (v_{n-1},v_n) 的旅行时间。同时，点 v_n 沿路径 $P_{v_1v_n}(t)$ 的到达时间函数被表示为：$l_{P_{v_1v_n}}(t) = t + w_{P_{v_1v_n}}(t)$。特别地，$l_{P_{v_1v_1}}(t) = t$。

本文中，令 $\mathcal{P}_{v_1v_n}(t)$ 表示当出发时间为 t 时从 v_1 到 v_n 的所有路径集合。$w^*_{v_1v_n}(t)$ 表示当出发时间为 t 时从 v_1 到 v_n 的最短旅行时间函数，$l^*_{v_n}(t)$ 表示当出发时间为 t 时 v_n 的最早到达时间函数。则

$$w^*_{v_1v_n}(t)=\min\left\{w_P(t)\mid P(t)\in\mathcal{P}_{v_1v_n}(t)\right\}$$

$$l^*_{v_n}(t)=\min\left\{l_P(t)\mid P(t)\in\mathcal{P}_{v_1v_n}(t)\right\}$$

假设 $P^*_{v_1v_n}(t)=\arg\min\left\{w_P(t)\mid P(t)\in\mathcal{P}_{v_1v_n}(t)\right\}$。显然，$l^*_{v_n}(t)=w_{P^*_{v_1v_n}}(t)+t=w^*_{v_1v_n}(t)+t$。因此 $P^*_{v_1v_n}(t)$ 称为对应于 $w^*_{v_1v_n}(t)$ 和 $l^*_{v_n}(t)$ 的路径。

本文考虑的图均为无重边且无负环的简单有向图，且本文只考虑满足先进先出(FIFO)性质的网络，即对于任意出发时刻 t,t'，如果 $t<t'$，则 $t+w^*_{v_1v_n}(t)<t'+w^*_{v_1v_n}(t')$。

接下来，我们先给出时变网络中求单条最短路的两个问题定义。

定义 2.1 (时变最短路径查询(TDSP) [17])。给定一个时变网络图 $G(V,E,W,T)$，源点为 v_s，汇点为 v_d，t 为出发时刻。TDSP 查询就是找出发时间 t 对应的最短旅行时间函数 $w^*_{v_sv_d}(t)$ 和相应的最短路径 $P^*_{v_sv_d}(t)$。

定义 2.2 (时间间隔最短路径查询(TIP) [17])。给定一个时变网络图 $G(V,E,W,T)$，源点为 v_s，汇点为 v_d，$I=[t,t']$ 为出发时间区间。TIP 查询就是找区间 I 内的最优出发时间 t^*，最短旅行时间函数 $w^*_{v_sv_d}(t^*)$ 和对应的最短路径 $P^*_{v_sv_d}(t^*)$。

由于本文要解决的是时变单故障边最短路径修复问题，设故障为边 (v_x,v_y) 在时间段 L 内发生故障，则将该故障表示为 $\mathcal{F}=[(v_x,v_y),L]$。下面将给出本文要解决的问题定义。

定义 2.3 (时变单故障边最短路径修复问题)。给定一个时变网络图 $G(V,E,W,T)$，源点为 v_s，汇点为 v_d，设故障 $\mathcal{F}=[(v_x,v_y),L]$，则时变单故障边最短路径修复问题是在给定任意 $\mathcal{F}$ 后找出从 v_s 到 v_d 的最短修复路径。

3. 算法方案：单链路故障情形下找时变的故障修复最短路径

3.1. 算法引入

网络图 $G(V,E,W,T)$ 中一般包含四种类型的图更改：权重增加、权重减少、边插入和边删除。本文主要针对删边这种边的变化情况给出时变情形下的修复算法。主要符号及说明见表 1。

Table 1. Main symbol and description
表 1. 主要符号及说明

符号	说明
T	总出发时间区间
$L=[t',t'']$	故障时间段
$w_{(v_i,v_j)}(t)$	当点 v_i 的出发时间为 t 时边 (v_i,v_j) 的旅行时间
$v_i(t:I)$	区间节点，当出发时间 $t\in I$ 时考虑点 v_i
$(v_i,v_j)(t:I)$	区间边，当出发时间 $t\in I$ 时考虑边 (v_i,v_j)
$SP(t:I)$	当 $t\in I$ 时从节点 v_s 到其它所有点最短路构成的图 G 的最短路子图
$\delta^-(v_i:I)$	v_i 在最短路子图 $SP(t:I)$ 中的入度
$p_{v_i}(t:I)$	当 $t\in I$ 时 v_i 在最短路图 $SP(t:I)$ 中的父节点
$l^*_{v_i}(t)$	当 $t\in I$ 时从 v_s 到 v_i 的初始最早到达时间函数(故障发生前)
$l'_{v_i}(t)$	当 $t\in I$ 时从 v_s 到 v_i 更新后的最早到达时间函数(故障发生后)

Continued

$l_{v_i}(t)$	当 $t \in I$ 时从 v_s 到 v_i 更新过程中的最早到达时间函数
$E_\delta(t:I)$	当 $t \in I$ 时删除故障边 (v_x,v_y) 造成的图 G 的受影响区间边集
$V_\delta(t:I)$	当 $t \in I$ 时删除故障边 (v_x,v_y) 造成的图 G 的受影响区间点集
$prec(v_i)$	点 v_i 在图 G 中的前继点
$succ(v_i)$	点 v_i 在图 G 中的后继点

在本文中，对于任意点 $v_i \in V(G)$，当出发时刻 $t \in I$ 时，将 $v_i(t:I)$ 称为区间点，表示当出发时刻 $t \in I$ 时考虑点 v_i。而对于任意边 $(v_i,v_j) \in E(G)$，当出发时刻 $t \in I$ 时，将 $(v_i,v_j)(t:I)$ 称为区间边，表示当出发时刻 $t \in I$ 时考虑边 (v_i,v_j)。$SP(t:I)$ 表示当出发时刻 $t \in I$ 时从节点 v_s 到其它所有点的最短路构成的图 G 的子图，称为当 $t \in I$ 时的最短路子图。由于本文的算法主要是对故障进行路径修复，因此，当某条边发生故障时，不可避免地会有一些边或点受到影响。这就说明故障的发生会导致 $SP(t:I)$ 子图产生相应的改变，比如删除某些受到影响的区间边或更新某些区间点的最早到达时间函数。进一步地，在本文中，对任意点 $v_j \in V(G)$，$l^*_{v_j}(t:I)$ 表示为当出发时间 $t \in I$ 时从 v_s 到 v_j 的更新前的最早到达时间，$l_{v_j}(t:I)$ 表示为当出发时间 $t \in I$ 时从 v_s 到 v_j 的更新过程中的最早到达时间，$l'_{v_j}(t:I)$ 表示为当出发时间 $t \in I$ 时从 v_s 到 v_j 的更新后的最早到达时间。据此，我们给出本文中受影响区间点集和受影响区间边集的定义。

定义 2.1 (受影响区间边集 $E_\delta(t:I)$)。在 $SP(t:I)$ 中，如果在时间段 L 内删除故障边 $(v_x,v_y)(t:I)$ 之后，新的最短路子图中不存在经过 $(v_i,v_j)(t:I)$ 的时变 v_sv_j-路或经过边 $(v_i,v_j)(t:I)$ 但 $l'_{v_j}(t:I) \neq l^*_{v_j}(t:I)$，则称边 $(v_i,v_j)(t:I)$ 为受影响区间边，$E_\delta(t:I)$ 为当出发时间 $t \in I$ 时所有受影响区间边的集合。

定义 2.2 (受影响区间点集 $V_\delta(t:I)$)。在 $SP(t:I)$ 中，如果在时间段 L 内删除故障边 $(v_x,v_y)(t:I)$ 之后，以点 v_j 为头节点的边是受影响区间边，则称点 v_j 为受影响区间点，$V_\delta(t:I)$ 为当出发时间 $t \in I$ 时所有受影响区间点的集合。特别地，当删除故障边 $(v_x,v_y)(t:I)$ 后，若 $SP(t:I)$ 中不存在以 $v_y(t:I)$ 为头节点的边，则点 $v_y(t:I)$ 本身也是一个受影响区间点。

由于路径的重建首要考虑范围为在相应时间段内的受影响区域，所以用 $|\delta|_t$ 表示路径所有出发时间区间内受影响区间点数目的最大值，用 $\|\delta\|_t$ 表示所有出发时间区间内受影响区间点和受影响区间边总数的最大值。

3.2. 主要算法和实现

3.2.1. 算法描述

由于本文的主要工作是在时变情形下修复最短路径，因此本文将故障发生前的最短路 $P^*(t)$ 及每个点对应的 $l^*_{v_i}(t)$ 作为一个输入，其中 $v_i \in V(G)$，该输入可由文献[17]中 TIP 算法直接得到，且该算法输出时间段 I 的一个完全划分 $I_1,I_2,\cdots,I_h$，对每一个 $I_i(i=1,\cdots,h)$，当出发时刻 $t \in I_i$ 时得到了最短 v_sv_d-路 $P^*_i(t:I_i)$ 和最短旅行时间函数 $w^*_{v_sv_d}(t:I_i)$，由此可得到最早到达时间函数 $l^*_{v_d}(t:I_i)$。

在算法 1 中，首先对所有点 $v_i \in V(G)$，初始化 $l_{v_i}(t)=l^*_{v_i}(t)$ 以便更新到达时间函数(第 1 行)，由于算法中给定的故障为 $\mathcal{F}=[(v_x,v_y),L]$，其中 $L=[t',t'']$，因此首先根据故障时间段 $[t',t'']$ 以及故障边 (v_x,v_y) 确定故障出发时间区间子集 $\{I_k\}$ (第 2 行)。这里，根据故障确定出发时间区间子集时，$l^*_{v_y}(t)$ 需要同时满足以下两个条件：

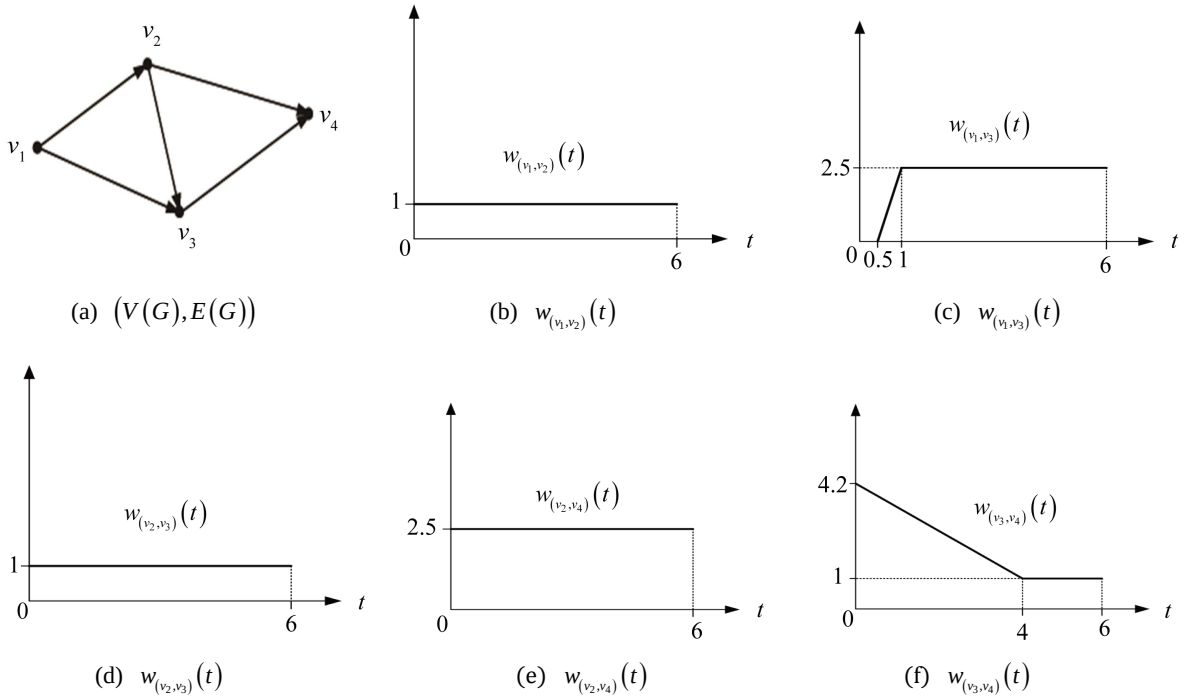
1) $l^*_{v_y}(t) \in \left(t'+w_{(v_x,v_y)}(l^*_{v_x}(t)),t''+w_{(v_x,v_y)}(l^*_{v_x}(t))\right)$

2) $l_{v_y}^*(t)\in(t',t'')$

综合以上两式，得 $l_{v_y}^*(t)\in\left(t',t''+w_{(v_x,v_y)}\left(l_{v_x}^*(t)\right)\right)$，若将该函数值范围对应的自变量区间子集记为 $\left\{\overline{I_k}\right\}$，则 $\left\{\overline{I_k}\right\}$ 为故障对应的出发时间区间子集(第 2 行)。

而对于任意点 $v_i\in V(G)$，由于 $l_{v_i}^*(t)$ 和 $P_{v_1v_i}^*(t)$ 都是已知的，因此只要将所有点的最早到达时间函数根据出发时间区间段再次统一分段就可得到不同时间段的最短路子图(第 3 行) (参考例 1)，并且每一个最短路子图都记录了 $l_{v_i}^*(t)$ 和 $P_{v_1v_i}^*(t)$-路。这样，当某条边在某个时间段发生故障时，只需要选择受故障影响的部分最短路子图进行相应修复即可，即只需要更新受影响区域，而无需对整个最短路子图都进行修复，简化了算法过程。基于此，本文给出解决方案，见算法 1。

例 1：假设给定时变网络图 $G(V,E,W,T)$，见下图 1，其中图(a)是图结构 $(V(G),E(G))$，共包含 4 个顶点 5 条边，其中 $V(G)=\{v_1,v_2,v_3,v_4\}$，v_1 为源点，v_4 为汇点，令 $\overline{I_k}=T=[0,6]$。边 $(v_1,v_2),(v_1,v_3),(v_2,v_3),(v_2,v_4),(v_3,v_4)$ 的旅行时间函数 $w_{(v_1,v_2)}(t),w_{(v_1,v_3)}(t),w_{(v_2,v_3)}(t),w_{(v_2,v_4)}(t),w_{(v_3,v_4)}(t)$ 分别见图(b)~(f)，下面给出当 $t\in\overline{I_k}$ 时求取最短路子图 $SP\left(t:\overline{I_k}\right)$ 的步骤。



(a) $(V(G),E(G))$　(b) $w_{(v_1,v_2)}(t)$　(c) $w_{(v_1,v_3)}(t)$

(d) $w_{(v_2,v_3)}(t)$　(e) $w_{(v_2,v_4)}(t)$　(f) $w_{(v_3,v_4)}(t)$

Figure 1. Time varying network graph $G(V,E,W,T)$

图 1. 时变网络图 $G(V,E,W,T)$

首先利用*TIP*算法可以得到各个点的最早到达时间函数如下：

$$l_{v_1}^*(t)=t,t\in[0,6].$$

$$l_{v_2}^*(t)=t+1,t\in[0,6].$$

$$l_{v_3}^*(t)=\begin{cases}t+0.5, & t\in[0,0.5);\\ 5t-1.5, & t\in[0.5,0.875);\\ t+2, & t\in[0.875,6].\end{cases}$$

$$l_{v_4}^*(t)=\begin{cases}t+3.5, & t\in[0,1.375);\\ 0.2t+4.6, & t\in[1.375,4);\\ t+3, & t\in[4,6].\end{cases}$$

$$P_{v_1v_1}^*(t)=\langle v_1\rangle, t\in[0,6].$$

$$P_{v_1v_2}^*(t)=\langle v_1,v_2\rangle, t\in[0,6].$$

$$P_{v_1v_3}^*(t)=\begin{cases}\langle v_1,v_3\rangle, & t\in[0,0.5);\\ \langle v_1,v_3\rangle, & t\in[0.5,0.875);\\ \langle v_1,v_2,v_3\rangle, & t\in[0.875,6].\end{cases}=\begin{cases}\langle v_1,v_3\rangle, & t\in[0,0.875);\\ \langle v_1,v_2,v_3\rangle, & t\in[0.875,6].\end{cases}$$

$$P_{v_1v_4}^*(t)=\begin{cases}\langle v_1,v_2,v_4\rangle, & t\in[0,1.375);\\ \langle v_1,v_2,v_3,v_4\rangle, & t\in[1.375,4);\\ \langle v_1,v_2,v_3,v_4\rangle, & t\in[4,6].\end{cases}=\begin{cases}\langle v_1,v_2,v_4\rangle, & t\in[0,1.375);\\ \langle v_1,v_2,v_3,v_4\rangle, & t\in[1.375,6].\end{cases}$$

则当$t\in\overline{I_k}=T=[0,6]$时，最短路子图$SP(t:\overline{I_k})$为：

$$SP(t:[0,6])=\begin{cases}(v_1,v_2),(v_1,v_3),(v_2,v_4), & t\in[0,0.5);\\ (v_1,v_2),(v_1,v_3),(v_2,v_4), & t\in[0.5,0.875);\\ (v_1,v_2),(v_2,v_3),(v_2,v_4), & t\in[0.875,1.375);\\ (v_1,v_2),(v_2,v_3),(v_3,v_4), & t\in[1.375,4);\\ (v_1,v_2),(v_2,v_3),(v_3,v_4), & t\in[4,6].\end{cases}$$

$$=\begin{cases}(v_1,v_2),(v_1,v_3),(v_2,v_4), & t\in[0,0.875);\\ (v_1,v_2),(v_2,v_3),(v_2,v_4), & t\in[0.875,1.375);\\ (v_1,v_2),(v_2,v_3),(v_3,v_4), & t\in[1.375,6].\end{cases}$$

令$\mathcal{J}=\{[0,0.875),[0.875,1.375),[1.375,6]\}$，则$\mathcal{J}$是时间区间$T=[0,6]$的最大子集集合，且满足所有点$v_i\in V(G)$在每一个$\mathcal{J}$的子集中都在同一个最短路子图中。

在上述实例中，最短路子图有 3 个，而我们的算法中，由于时变网络的不确定性，将最短路子图的个数记为$\alpha(T)$。

选出故障出发时间区间后确定对应的最短路子图$\{SP(t:\overline{I_k})\}$后(第 3 行)，这就意味着算法只需要对选出的$\{\overline{I_k}\}$区间依次遍历。由于故障边是$(v_x,v_y)(t:\overline{I_k})$，因此，对于每一个区间$\overline{I_k}$，应首先将边$(v_x,v_y)$从$SP(t:\overline{I_k})$中删除(第 5 行)并减小$v_y(t:\overline{I_k})$的入度值$\delta^-(v_y(t:\overline{I_k}))$(第 6 行)，再判断$v_y(t:I)$在当前最短路子图中的入度值。若删边之后入度值为 0，说明$v_y(t:\overline{I_k})$的最早到达时间函数需要更新，即$v_y(t:\overline{I_k})$此时是一个受影响区间点(第 7 行)。

因此，当给定$l_{v_i}^*(t:\overline{I_k})$、$SP(t:\overline{I_k})$时，假设故障$\mathcal{F}=((v_x,v_y),L)$，则会出现以下两种情况：

Case 1. $(v_x,v_y)\in SP(t:\overline{I_k})$且$\delta^-(v_y(t:\overline{I_k}))=1$

说明故障边的出现导致$l_{v_i}^*(t)$和$SP(t)$都需要进行更新，即算法的主要部分；

Case 2. $(v_x,v_y)\notin SP(t:\overline{I_k})$或$\delta^-(v_y(t:\overline{I_k}))>1$

说明故障时间段对初始最优路没有影响，无需对最短路子图更新，直接删除故障边。

接下来，针对 *Case 1* 给出更新最早到达时间函数的算法，主要分为两个子算法：

算法 2：通过故障识别受影响区间点和受影响区间边

当出发时刻$t\in\overline{I_k}$时，由于故障边为$\left(v_x,v_y\right)$，因此$\left(v_x,v_y\right)\left(t:\overline{I_k}\right)$是受影响区间边。当从$SP\left(t:\overline{I_k}\right)$中删除边$\left(v_x,v_y\right)$后点$v_y$的入度大于 1，说明不需要更新解，但当点$v_y$的入度为 0，则说明点$v_y\left(t:\overline{I_k}\right)$是受影响区间点，因此初始化工作点集$J=\left\{v_y\left(t:\overline{I_k}\right)\right\}$，受影响点集$V_\delta\left(t:\overline{I_k}\right)=\varnothing$ (第 1~2 行)。这里，$v_i\left(t:\overline{I_k}\right)$被称作区间点，表示当出发时刻$t\in\overline{I_k}$时考虑点$v_i$。其中，工作点集 J 中存储一些已经识别为受影响区间点但并未进行处理的点。初始 J 中只有$v_y\left(t:\overline{I_k}\right)$，因此在后续的算法过程中，从$v_y\left(t:\overline{I_k}\right)$开始依次处理 J 中的点(第 3~14 行)。而每次当点$v_i\left(t:\overline{I_k}\right)\in J$被处理完之后，相应地将$v_i\left(t:\overline{I_k}\right)$从 J 中移除，并将以v_i为头节点的$SP\left(t:\overline{I_k}\right)$边都删除(受影响区间边) (第 4~5 行)。依次进行这样的迭代直到得到区间$\overline{I_k}$内的完整受影响区间点集$V_\delta\left(t:\overline{I_k}\right)$。

算法 3：更新受影响区间点的最早到达函数和最短路子图

算法 3 主要利用了 Dijkstra 优先搜索思想来更新受影响区间点的最早到达时间函数。由于本文考虑的最短路子图是从源点到其它所有点的最短路，因此当已知任意区间点$v_i\left(t:\overline{I_k}\right)$更新前的最早到达时间函数$l_{v_i}^*\left(t:\overline{I_k}\right)$时，我们可以运用已知的最早到达时间函数和最短路子图的结构来更新受影响区间点的最早到达时间函数$l'_{v_i}\left(t:\overline{I_k}\right)$。

算法 3 用到的两个主要步骤：

1) 根据不受影响区间点的最早到达时间函数更新受影响区间点的最早到达时间函数(考虑前向边) (第 2~10 行)

2) 考虑受影响区间点发出的边导致的最早到达时间函数(考虑后向边) (第 11~23 行)

步骤 1 的主要思想参考下图 2 故障边删除模型，A、B 分别代表在区间$\overline{I_k}$内的不受影响区间点集$\overline{V_\delta}\left(t:\overline{I_k}\right)$和受影响区间点集$V_\delta\left(t:\overline{I_k}\right)$，$\left(v_x,v_y\right)\left(t:\overline{I_k}\right)$是受影响区间边，$v_y\left(t:\overline{I_k}\right)\in V_\delta\left(t:\overline{I_k}\right)$，$v_x\left(t:\overline{I_k}\right)\in\overline{V_\delta}\left(t:\overline{I_k}\right)$，$v_a\left(t:\overline{I_k}\right)\in\overline{V_\delta}\left(t:\overline{I_k}\right)$。当$t\in\overline{I_k}$时，由于对于任意点$v_x\left(t:\overline{I_k}\right)\in\overline{V_\delta}\left(t:\overline{I_k}\right)$，$l_{v_x}^*\left(t:I_k\right)$是已知的且不需要更新，因此当$v_y\left(t:\overline{I_k}\right)\in V_\delta\left(t:\overline{I_k}\right)$时，假设当$t\in\overline{I_k}$时初始的最优$v_sv_y$-路经过边$\left(v_x,v_y\right)$，且点$v_y\left(t:\overline{I_k}\right)$的最早到达时间函数$l_{v_y}^*\left(t:I_k\right)=l_{v_x}^*\left(t:I_k\right)+w_{\left(v_x,v_y\right)}\left(l_{v_x}^*\left(t:I_k\right)\right)$。假设对于点$v_y\left(t:\overline{I_k}\right)$，当$I_k\subseteq\overline{I_k}$时，此时找到的当前最优不受影响的前继点为$v_a\left(t:I_k\right)$，则更新新的$v_sv_y$-路，此时该条路经过边$\left(v_a,v_y\right)$，且新的$l_{v_y}\left(t:I_k\right)=l_{v_a}^*\left(t:I_k\right)+w_{\left(v_a,v_y\right)}\left(l_{v_a}^*\left(t:I_k\right)\right)$。这样，通过这种思想就可以初步更新受影响区间点的最早到达时间函数。因此，对于每个受影响区间点$v_b\left(t:\overline{I_k}\right)$，依次选取最优的不受影响区间点$\left\{v_a\left(t:I_k\right)\right\}$，其中$I_k\subseteq\overline{I_k}$，且$\left\{I_k\right\}$是$\overline{I_k}$的完全划分(第 3 行)，并将边$\left\{\left(v_a,v_b\right)\left(t:I_k\right)\right\}$加入到$\left\{SP\left(t:I_k\right)\right\}$中(第 4 行)。最后，将更新过的区间点$v_b\left(t:I_k\right)$放入 Q 中并更新$\left\{v_b\left(t:I_k\right)\right\}$的最早到达时间函数(第 5~9 行)。

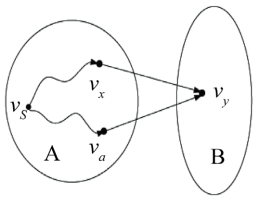


Figure 2. Fault edge deletion model
图 2. 故障边删除模型

在步骤 2 中(第 11~23 行)，对于每一个受影响区间点$v_i\left(t:I_k\right)\in Q$，在第 2~10 行已经根据不受影响区间点集中的最早到达时间函数进行了初步更新。因此，在此步骤中，首先根据这些点的最早到达时间函数选取每个区间I'_k中值最小的点，将选出的这些区间点记为$\left\{v_b\left(t:I'_k\right)\right\}$，其中$I'_k\subseteq I_k$，且$\left\{I'_k\right\}$是$I_k$的

完全划分。这就说明$\left\{v_b\left(t:I_k'\right)\right\}$已是最优区间点，之后无需再进行更新，因此从 Q 中取出(第 13 行)。而对于选出的区间点$\left\{v_b\left(t:I_k'\right)\right\}$，由于本文修复的是最短路子图，因此首先根据前继点判断入度是否为非 0，即是否在$I_k''\subseteq I_k'$中存在不同的v_sv_b-路但具有相同的到达时间(第 15~17 行)。最后，根据已经选出的最优区间点$\left\{v_b\left(t:I_k'\right)\right\}$更新后继点的最早到达时间函数(第 18~21 行)，继续迭代直到 Q 中的元素为空。步骤 2 的主要思想是 Dijkstra 优先搜索，每次选取当前最早到达的点进行向后探搜索，直到所有的受影响区间点都被遍历完。

在算法 1 中，第 2 行$\arg\min\left\{l_{v_y}^*\left(t\right)\middle|l_{v_y}^*\left(t\right)\in\left(t',t''+w_{\left(v_x,v_y\right)}\left(l_{v_x}^*\left(t\right)\right)\right)\right\}$表示返回值函数$\left\{l_{v_y}^*\left(t\right)\middle|l_{v_y}^*\left(t\right)\in\left(t',t''+w_{\left(v_x,v_y\right)}\left(l_{v_x}^*\left(t\right)\right)\right)\right\}$对应的自变量区间，即出发时间区间。第 12 行$\left\{v_b\left(t:I_k'\right)\right\}=\arg\min\left\{l_{v_i}\left(t:I_k\right)\middle|v_i\left(t:I_k\right)\in Q\right\}$表示返回在 Q 中具有最早到达时间函数的区间点集合$\left\{v_b\left(t:I_k'\right)\right\}$，这里$\left\{I_k'\right\}$是$I_k$的完全划分，复杂度为$O\left(\alpha\left(T\right)\log n\right)$，其中 n 为 Q 中元素的个数，$\alpha\left(T\right)$是每个函数操作所需的时间代价，参考例 2。算法 2 中第 7 行$\text{Insert}\left(Q,v_b\left(t:I_k\right),l_{v_b}\left(t:I_k\right)\right)$表示如果$v_b\left(t:I_k\right)$在 Q 中，则更新$l_{v_b}\left(t:I_k\right)$，如果不在 Q 中，将$v_b\left(t:I_k\right)$以及对应的最早到达时间函数$l_{v_b}\left(t:I_k\right)$放入$Q$中，第 20 行$\text{Update}\left(Q,v_c\left(t:I_k''\right),l_{v_c}\left(t:I_k''\right)\right)$则表示更新 Q 中$v_c\left(t:I_k''\right)$对应的最早到达时间函数$l_{v_c}\left(t:I_k''\right)$。

例 2：argmin 解释：设$T=[0,3]$，且

$$l_{v_1}\left(t:T\right)=2,t\in[0,3].$$

$$l_{v_2}\left(t:T\right)=\begin{cases}t+1, & t\in[0,2);\\ 3, & t\in[2,3].\end{cases}$$

则：

$$\arg\min\left\{l_{v_1}\left(t:[0,3]\right),l_{v_2}\left(t:[0,3]\right)\right\}=\begin{cases}v_2\left(t:[0,1)\right);\\ v_1\left(t:[1,3]\right).\end{cases}$$

令$\mathcal{J}=\left\{[0,1),[1,3]\right\}$，则$\mathcal{J}$是时间区间$T=\overline{I_k}=[0,3]$的最小子集合，且满足在每一个子区间内仅存在一个点$v_i$使得$v_i\left(t:T\right)=\arg\min\left\{l_{v_i}\left(t:T\right)\right\}$。

3.2.2. 算法方案

本节中，算法 1 针对本文问题给出了主要算法方案，算法详细描述参考 3.3.1。

算法 1：单链路故障情形下找时变的故障修复最短路径

输入： $G\left(V,E,W,T\right),v_s,v_d,\left\{l_{v_i}^*\left(t\right)\right\},P^*\left(t\right),SP\left(t\right)$，其中$v_i\in V\left(G\right)$，$t\in T$，$\mathcal{F}=\left[\left(v_x,v_y\right),L\right]$，其中$L=[t',t'']$

输出： $P'\left(t\right)$-路

1. 初始化：$l_{v_i}\left(t\right)=l_{v_i}^*\left(t\right)$
2. $\left\{\overline{I_k}\right\}=\arg\min\left\{l_{v_y}^*\left(t\right)\middle|l_{v_y}^*\left(t\right)\in\left(t',t''+w_{\left(v_x,v_y\right)}\left(l_{v_x}^*\left(t\right)\right)\right)\right\}$
3. 求$\left\{SP\left(t:\overline{I_k}\right)\right\}$
4. **for**{每个区间$\overline{I_k}$} **do**
5. 从$SP\left(t:\overline{I_k}\right)$中删除边$\left(v_x,v_y\right)$

Continued

6. $\delta^-\left(v_y\left(t:\overline{I_k}\right)\right)=\delta^-\left(v_y\left(t:\overline{I_k}\right)\right)-1$
7. **if**{ $\delta^-\left(v_y\left(t:I_k\right)\right)=0$ } **then**
8. 执行**算法 2**
9. 执行**算法 3**
10. $l'_{v_b}\left(t:I_k''\right)=l_{v_b}\left(t:I_k''\right)$
11. **endif**
12. **endfor**

接下来，算法 2 根据最短路子图给出识别受影响区间点以及删除受影响区间边的算法方案。

算法 2：识别受影响区间点，删除受影响区间边

1. $J=\left\{v_y\left(t:\overline{I_k}\right)\right\}$
2. $V_\delta\left(t:\overline{I_k}\right)=\varnothing$
3. **while** { $J\neq\varnothing$ } **do**
4. 从 J 中删除 $\left\{v_i\left(t:\overline{I_k}\right)\right\}$
5. 从 $SP\left(t:\overline{I_k}\right)$ 中删除边 $\left(v_z,v_i\right)$
6. $V_\delta\left(t:\overline{I_k}\right)\leftarrow\left(v_i,\overline{I_k}\right)$
7. $l_{v_y}\left(t:\overline{I_k}\right)=\infty$
8. **for**{每个满足 $\left(v_i,v_j\right)\in SP\left(t:\overline{I_k}\right)$ 的区间点 $v_j\left(t:\overline{I_k}\right)$ } **do**
9. 从 $SP\left(t:\overline{I_k}\right)$ 中删除边 $\left(v_i,v_j\right)$，$\delta^-\left(v_j\left(t:\overline{I_k}\right)\right)=\delta^-\left(v_j\left(t:\overline{I_k}\right)\right)-1$
10. **if**{ $\delta^-\left(v_j\left(t:\overline{I_k}\right)\right)=0$ } **then**
11. $J\leftarrow v_j\left(t:\overline{I_k}\right)$
12. **endif**
13. **endfor**
14. **endwhlie**

最后，算法 3 给出了更新最早到达时间函数的算法实现方案。

算法 3：更新从 v_s 到 v_b 的最早到达时间函数，更新 $SP\left(t:\overline{I_k}\right)$，其中 $v_b\left(t:\overline{I_k}\right)\in V_\delta\left(t:\overline{I_k}\right)$

1. $Q=\varnothing$
2. **for**{每个区间点 $v_b\left(t:\overline{I_k}\right)\in V_\delta\left(t:\overline{I_k}\right)$ } **do**
3. $\left\{l_{v_b}\left(t:I_k\right)\right\}=\min\left\{l_{v_a}^*\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}^*\left(t:I_k\right)\right)\middle|\left(v_a,v_b\right)\in E(G),v_a\notin V_\delta\left(t:I_k\right)\right\}$，其中 $\{I_k\}$ 是 $\overline{I_k}$ 的完全划分
4. 将 $\left\{\left(v_a,v_b\right)\right\}$ 边插入 $SP\left(t:I_k\right)$
5. **for**{每个 $l_{v_b}\left(t:I_k\right)$ } **do**
6. **if**{ $l_{v_b}\left(t:I_k\right)\neq\infty$ } **then**
7. Insert $\left(Q,v_b\left(t:I_k\right),l_{v_b}\left(t:I_k\right)\right)$
8. **endif**

Continued

9. **endfor**
10. **endfor**
11. **whlie**{ $Q \neq \varnothing$ } **do**
12. $\left\{v_b\left(t:I_k'\right)\right\} = \arg\min\left\{l_{v_i}\left(t:I_k\right) \mid v_i\left(t:I_k\right) \in Q\right\}$，其中 $\left\{I_k'\right\}$ 是 I_k 的完全划分
13. 从 Q 中删除 $\left\{v_b\left(t:I_k'\right)\right\}$
14. **for**{每个 $v_b\left(t:I_k'\right)$ } **do**
15. **for**{每个满足 $l_{v_a}\left(t:I_k''\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k''\right)\right)=l_{v_b}\left(t:I_k''\right)$ 的点 $v_a \in prec\left(v_b\right)$，$I_k'' \subseteq I_k'$ } **do**
16. 将边 $\left(v_a,v_b\right)$ 插入到 $SP\left(t:I_k''\right)$，$\delta^-\left(v_b\left(t:I_k''\right)\right)=\delta^-\left(v_b\left(t:I_k''\right)\right)+1$
17. **endfor**
18. **for**{每个满足 $l_{v_b}\left(t:I_k''\right)+w_{(v_b,v_c)}\left(l_{v_b}\left(t:I_k''\right)\right)<l_{v_c}\left(t:I_k''\right)$ 的点 $v_c \in succ\left(v_b\right)$，$I_k'' \subseteq I_k'$ } **do**
19. $l_{v_c}\left(t:I_k''\right)=l_{v_b}\left(t:I_k''\right)+w_{(v_b,v_c)}\left(l_{v_b}\left(t:I_k''\right)\right)$
20. Update($Q,v_c\left(t:I_k''\right),l_{v_c}\left(t:I_k''\right)$)
21. **endfor**
22. **endfor**
23. **endwhlie**

4. 算法正确性分析

为方便描述，后续的证明均以出发时刻 $t \in I_k$ 的情况进行说明，其中这里的 I_k 表示任意出发时间区间。另外，再次需要强调的是，当 $t \in I_k$ 时，对任意点 $v_i \in V(G)$，$l_{v_i}^*\left(t:I_k\right)$、$l_{v_i}\left(t:I_k\right)$ 和 $l_{v_i}'\left(t:I_k\right)$ 分别表示更新前、更新过程中和更新后的最早到达时间函数。

观察：在更新最早到达时间函数操作之前，以下的两个性质是成立的：

1) 对于任意点 $v_i \in V(G)$ 及 $t \in I_k$，从 v_s 到 v_i 的最早到达时间函数被正确存储，即 $l_{v_i}\left(t:I_k\right)=l_{v_i}^*\left(t:I_k\right)$。

2) $SP\left(t:I_k\right)$ 是当 $t \in I_k$ 时以源点 v_s 为源节点的最短路子图，即对于任意点 $v_i \in V(G)$ 及 $t \in I_k$，在 $SP\left(t:I_k\right)$ 中从 v_s 到 v_i 的路就是最短路。

为了证明算法的正确性，我们将说明在算法执行完后上面两个性质仍然成立。

引理 4.1 $G(V,E,W,T)$ 为给定时变网络图，如果删边操作在边 $\left(v_x,v_y\right)$ 上执行，且性质 1、2 成立，则对任意出发时间 $t \in I_k$，算法 2 正确处理了所有点。

下面我们将证明更新算法(即算法 3)的正确性。

引理 4.2 给定时变网络图 $G(V,E,W,T)$，如果删边操作在边 $\left(v_x,v_y\right)$ 上执行，当出发时间 $t \in I_k$ 时，则该算法对任意区间点 $v_i\left(t:I_k\right) \in Q$ 正确地以非递减顺序计算了 $l_{v_i}\left(t:I_k\right)$。

证明：对于任意点 $v_i\left(t:I_k\right) \in Q$，假设点 $v_j\left(t:I_k\right)$ 为从 Q 中取出不满足非递减顺序的第一个点(算法 3 第 12 行)。令当 $t \in I_k$ 时从 Q 中先取出 v_i 再取出 v_j，由假设可知，$l_{v_j}\left(t:I_k\right)<l_{v_i}\left(t:I_k\right)$。当 v_j 从 Q 中取出时，$l_{v_j}\left(t:I_k\right)$ 已经被计算过(算法 2 第 3 行)，即 $l_{v_j}\left(t:I_k\right)=l_{v_a}^*\left(t:I_k\right)+w_{(v_a,v_j)}\left(l_{v_a}^*\left(t:I_k\right)\right)$，其中 v_j 满足 $l_{v_j}\left(t:I_k\right)=l_{v_a}\left(t:I_k\right)+w_{(v_a,v_j)}\left(l_{v_a}\left(t:I_k\right)\right)>l_{v_a}\left(t:I_k\right)=l_{v_a}^*\left(t:I_k\right)$。因此，$l_{v_a}\left(t:I_k\right)<l_{v_j}\left(t:I_k\right)<l_{v_i}\left(t:I_k\right)$。由算法 3 第 12 行的取点规则可知，点 v_a 比点 v_j 先从 Q 中取出，且 v_j 的优先性和最早到达时间函数被正确更新(算法 3 第 20 行)，且 v_j 的更新在点 v_i 之前，但这与假设矛盾，因此假设不成立。

引理 4.3 给定时变网络图为 $G(V,E,W,T)$，如果删边操作在边 $\left(v_x,v_y\right)$ 上执行，且性质 1、2 成立，则算法 3 执行完毕之后性质 1、2 仍然成立。

证明：引理 4.1 首先说明了算法 2 正确处理了所有点。接下来，对于任意受影响区间点 $v_b \in V_\delta\left(t:I_k\right)$，

执行第算法 2 第 2~10 行，由于点 v_a 在 I_k 区间内是不受影响区间点，因此该操作保证了 $v_i\left(t:I_k\right)$ 在所有区间的当前最优前继点 v_a 被选出，并将此时的 $l_{v_i}\left(t:I_k\right)$ 记录下来供后续步骤使用。现在说明对于满足算法 3 中第 15 行和第 18 行条件的点，在算法 3 执行完毕之后性质 1、2 仍然成立。

对于满足算法 3 中满足 15 行条件的任意点 v_i，根据引理 4.2 可得 $l_{v_a}^*\left(t:I_k\right)=l_{v_a}'\left(t:I_k\right)$，又由于 $l_{v_b}\left(t:I_k\right)=l_{v_b}'\left(t:I_k\right)=l_{v_b}^*\left(t:I_k\right)$，因此 $v_b\left(t:I_k\right)$ 的到达时间在第 38 行不会被增加，因此，对于满足条件第 38 行的任意点 v_b，性质 1、2 成立。

为了证明满足算法 3 中第 18 行条件的点 v_b 在算法 3 执行完毕之后性质 1、2 仍然成立，即 $l_{v_b}\left(t:I_k\right)=l_{v_b}'\left(t:I_k\right)$，下面我们分别证明 $l_{v_b}\left(t:I_k\right)\geq l_{v_b}'\left(t:I_k\right)$ 和 $l_{v_b}\left(t:I_k\right)\leq l_{v_b}'\left(t:I_k\right)$。

要证明 $l_{v_b}\left(t:I_k\right)\geq l_{v_b}'\left(t:I_k\right)$，首先证明如下断言成立：

断言：在算法 3 的执行过程中，当点 v_b 的到达时间函数不是无穷大时(算法 3 第 6 行)，v_b 的优先性就是在删边之后的图中从源点到点 v_b 的最早到达时间函数。

证明：首先，在算法 2 中，当区间点 $v_b\left(t:I_k\right)$ 被插入到 Q 中时，它的优先性是 $l_{v_a}\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k\right)\right)=l_{v_a}^*\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}^*\left(t:I_k\right)\right)\geq l_{v_b}'\left(t:I_k\right)$，其中 $v_a\left(t:I_k\right)$ 是满足第 3 行条件的 $v_b\left(t:I_k\right)$ 最优的不受影响邻居区间点且 $l_{v_a}'\left(t:I_k\right)=l_{v_a}^*\left(t:I_k\right)$。$v_b\left(t:I_k\right)$ 的优先性对应于在删边之后的 $SP\left(t:I_k\right)$ 子图中从 v_s 到 v_b 的最早到达时间函数。这就说明在算法 2 结束后我们的陈述仍然是正确的。

假设我们给出的断言在算法 3 的执行中不正确。设 $v_b\left(t:I_k\right)$ 是在 Q 中使得陈述不正确的具有最小优先性的点，$l_{v_b}\left(t:I_k\right)=l_{v_a}\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k\right)\right)$。由于我们假设断言不正确，即 $l_{v_b}\left(t:I_k\right)$ 不是在删边之后的 $SP\left(t:I_k\right)$ 子图中从源点 v_s 到 v_b 的最早到达时间函数，即 $l_{v_b}\left(t:I_k\right)<l_{v_b}'\left(t:I_k\right)$。由于该陈述对于区间点 $v_b\left(t:I_k\right)$ 在算法 3 开始的时候是正确的，这就意味着在 Q 中 $v_b\left(t:I_k\right)$ 的优先性已经改变为 $l_{v_a}\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k\right)\right)$(第 3 行)。通过假设，当点 v_a 从 Q 中取出，$l_{v_b}\left(t:I_k\right)$ 就是在删边之后的新图中从源点从 v_s 到 v_b 的最早到达时间函数。这与假设矛盾，所以断言成立。下面只需证明 $l_{v_b}\left(t:I_k\right)\leq l_{v_b}'\left(t:I_k\right)$。

假设 $v_b\left(t:I_k\right)$ 是第一个从 Q 中取出来但 $l_{v_b}\left(t:I_k\right)$ 不是最优的点，也就是 $l_{v_b}\left(t:I_k\right)>l_{v_b}'\left(t:I_k\right)$。令 $v_a\left(t:I_k\right)$ 是 $v_b\left(t:I_k\right)$ 在 $SP\left(t:I_k\right)$ 中的父节点，其中点 $v_a\left(t:I_k\right)$ 被正确处理，即 $l_{v_b}'\left(t:I_k\right)=l_{v_a}'\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}'\left(t:I_k\right)\right)$。通过引理 4.2 及 $l_{v_a}'\left(t:I_k\right)<l_{v_b}'\left(t:I_k\right)$，我们推得：当 v_a 满足第 18 行时，由于 v_b 是第一个满足 $l_{v_a}^*\left(t:I_k\right)=l_{v_a}'\left(t:I_k\right)$，$l_{v_b}\left(t:I_k\right)>l_{v_a}\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k\right)\right)=l_{v_b}'\left(t:I_k\right)$ 的点，这说明 $v_a\left(t:I_k\right)$ 优先 $v_b\left(t:I_k\right)$ 从 Q 中取出。另外，当 $l_{v_a}\left(t:I_k\right)$ 被正确计算之后，第 18 行就考虑边 $\left(v_a,v_b\right)$，且更新 v_b 的最早到达时间函数 $l_{v_b}\left(t:I_k\right)$ 为 $l_{v_a}\left(t:I_k\right)+w_{(v_a,v_b)}\left(l_{v_a}\left(t:I_k\right)\right)$(第 18 行)。这与 $l_{v_b}\left(t:I_k\right)$ 不是最优矛盾。

因此，该引理得证。

5. 算法复杂性分析

边权增加的修复最短路(即增量算法)最坏情况下运行时间为 $O\left(\left\|\delta\right\|+\left|\delta\right|\log\left|\delta\right|\right)$。其中 $\left|\delta\right|$ 表示受影响区间点的数目，$\left\|\delta\right\|$ 表示受影响区间点和受影响区间边的总数，该复杂度分析见[12]。基于这种自适应参数的思想，下面给出本文中算法的复杂度分析。

在算法 1 中，第 1~3 行的运行时间为 $O(1)$，由于在本文中，故障的出发时间区间集为 $\left\{\overline{I_k}\right\}$，对于每个区间 $\overline{I_k}$，我们令 $\max\left|V_\delta\left(t:\overline{I_k}\right)\right|=\left|\delta\right|_t$，表示所有出发时间区间内受影响区间点数目的最大值，相应地，$\left\|\delta\right\|_t$ 表示所有出发时间区间内受影响区间点和受影响区间边总数的最大值。第 3 行的 **for** 循环最多执行 $\alpha(T)$ 次，下面分析算法 2 和算法 3 的复杂度。

在算法 2 中，第 3 行的 **while** 循环最多执行 $|\delta|_t$ 次，而第 8 行的 **for** 循环最多执行 $|succ(v_b)|$ 次，其余操作的复杂度均为 $O(1)$，因此算法 2 的总复杂度为 $O\left(\sum_{v_b \in V_\delta(t:\overline{I_k})} |succ(v_b)|\right) = O\left(\|\delta\|_t\right)$，其中 $\|\delta\|_t$ 表示所有出发时间区间内受影响区间点和受影响区间边总数的最大值。在算法 3 中，第 2 行最多执行 $|\delta|_t$ 次，第 3 行最多执行 $\log(prec(v_b))$ 次，由于 $prec(v_b)$ 为常数，故时间复杂度为 $O(1)$。第 11 行最多执行 $|\delta|_t$ 次，第 12 行的操作复杂度为 $O\left(\log|\delta|_t\right)$，因此算法 3 的总复杂度为：$|\delta|_t \cdot \log|\delta|_t$。

故该算法的总复杂度为：$O\left(\alpha(T)\cdot\left[\|\delta\|_t + |\delta|_t \cdot \log|\delta|_t\right]\right)$。

由此可见，在大规模的时变网络中，当故障影响范围较小或者故障持续时间较短时，本文提出的算法具有较优的复杂性。

6. 结语

本文研究了单链路故障情形下找时变的故障修复最短路径问题，当给定任意链路故障$\mathcal{F}$之后，本文基于原始的时变最短路径进行修复进而找到新的时变最短路径，以保证网络的生存性和服务质量要求。针对该问题，本文给出一个灵活且高效的算法解决该问题，该算法不仅利用了最短路子图的思想和自适应参数降低了计算复杂度，而且在路径修复过程中有效利用了时间段划分并保证了节点之间传输的连续性，最后证明了该算法的正确性并计算得到该算法的时间复杂度为 $O\left(\alpha(T)\cdot\left[\|\delta\|_t + |\delta|_t \cdot \log|\delta|_t\right]\right)$，其中 $|\delta|_t = \max\left|V_\delta\left(t:\overline{I_k}\right)\right|$，表示所有出发时间区间内受影响区间点数目的最大值，$\|\delta\|_t$ 表示所有出发时间区间内受影响区间点和受影响区间边总数的最大值，$\alpha(T)$ 是每个函数操作所需的时间代价。

参考文献

[1] Narváez, P., Siu, K.-Y. and Tzeng, H.-Y. (2000) New Dynamic Algorithms for Shortest Path Tree Computation. *IEEE/ACM Transactions on Networking*, **8**, 734-746. https://doi.org/10.1109/90.893870

[2] Bruera, F., Cicerone, S., D'Angelo, G., Di Stefano, G. and Frigioni, D. (2008) Dynamic Multi-Level Overlay Graphs for Shortest Paths. *Mathematics in Computer Science*, **1**, 709-736. https://doi.org/10.1007/s11786-007-0023-5

[3] Delling, D. and Wagner, D. (2007) Landmark-Based Routing in Dynamic Graphs. *WEA'07 Proceedings of the 6th International Conference on Experimental Algorithms*, Rome, 6-8 June 2007, 52-65. https://doi.org/10.1007/978-3-540-72845-0_5

[4] Wagner, D. and Wattenhofer, R. (2008) Algorithms for Sensor and Ad Hoc Networks. Lecture Notes in Computer Science, Vol. 4621, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-74991-2

[5] Kleinberg, J. and Tardos, E. (2005) Algorithm Design. Addison Wesley, Boston, MA. https://www.researchgate.net/publication/232203501_Algorithm_Design

[6] Ausiello, G., Italiano, G.F., Spaccamela, A.M. and Nanni, U. (1991) Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms*, **12**, 615-638. https://doi.org/10.1016/0196-6774(91)90036-X

[7] Feuerstein, E. and Marchetti-Spaccamela, A. (1991) Dynamic Algorithms for Shortest Paths in Planar Graphs. *WG'91 Proceedings of the 17th International Workshop*, Fischbachau, 17-19 June 1991, 187-197. https://doi.org/10.1007/3-540-55121-2_18

[8] Frigioni, D., Marchetti-Spaccamela, A. and Nanni, U. (1998) Semidynamic Algorithms for Maintaining Single-Source Shortest Path Trees. *Algorithmica*, **22**, 250-274. https://doi.org/10.1007/PL00009224

[9] Henzinger, M.R., Klein, P., Rao, S., *et al.* (1994) Faster Shortest-Path Algorithms for Planar Graphs. *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, Montreal, 23-25 May 1994, 27-37. https://doi.org/10.1145/195058.195092

[10] Klein, P.N. and Subramanian, S. (1998) A Fully Dynamic Approximation Scheme for Shortest Paths in Planar Graphs. *Algorithmica*, **22**, 235-249. https://doi.org/10.1007/PL00009223

[11] Ramalingam, G. (1996) Thomas Reps. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms*, **21**, 267-305. https://doi.org/10.1006/jagm.1996.0046

[12] Ramalingam, G. and Reps, T. (1996) On the Computational Complexity of Dynamic Graph Problems. *Theoretical Computer Science*, **158**, 233-277. https://doi.org/10.1016/0304-3975(95)00079-8

[13] Rohnert, H. (1985) A Dynamization of the All Pairs Least Cost Path Problem. *Proceedings on STACS* 85 2*nd Annual Symposium on Theoretical Aspects of Computer Science*, Saarbrücken, 3-5 January 1985, 279-286. https://doi.org/10.1007/BFb0024016

[14] Frigioni, D., Marchetti-Spaccamela, A. and Nanni, U. (2000) Fully Dynamic Algorithms for Maintaining Shortest Paths Trees. *Journal of Algorithms*, **34**, 251-281. https://doi.org/10.1006/jagm.1999.1048

[15] Bauer, R. and Wagner, D. (2009) Batch Dynamic Single-Source Shortest-Path Algorithms: An Experimental Study. *SEA'*09: *Proceedings of the* 8*th International Symposium on Experimental Algorithms*, Dortmund, 4-6 June, 51-62. https://doi.org/10.1007/978-3-642-02011-7_7

[16] Li, L., Wang, S. and Zhou, X. (2019) Time-Dependent Hop Labeling on Road Network. 2019 *IEEE* 35*th International Conference on Data Engineering* (*ICDE*), Macao (China), 8-11 April 2019, 902-913. https://doi.org/10.1109/ICDE.2019.00085

[17] Wang, Y., Li, G. and Tang, N. (2019) Querying Shortest Paths on Time Dependent Road Networks. *Proceedings of the VLDB Endowment*, **12**, 1249-1261. https://doi.org/10.14778/3342263.3342265