最短路径算法—标号更新法之深入浅出 Label Correcting Algorithm

崔赞扬，北京交通大学

Email: dr.zanyang@bjtu.edu.cn

李崇楠，北京交通大学

Email: chongnanli1997@hotmail.com

Faculty member:

Xuesong (Simon) Zhou, Arizona State University

Email: xzhou74@asu.edu

本文重在从算法基本原理、复杂度分析、优缺点、代码实现、算法扩展等方面科普 Label Correcting Algorithm（最短路算法重要分支），同时给出了下一步学习内容建议。本文在周学松教授的指导下完成，特别感谢周学松教授的全程指导，感谢姚宇、牛志强、张宇丰在文章结构、内容安排、内容校对等方面提出的宝贵建议。

1. 前言

最短路问题是图论理论的一个经典问题，其目的在于寻找网络中任意两个节点间的最短路径，这里的最短可以衍生为距离最短、费用最小、时间最短等一系列度量。交通领域最短路径问题有着广泛应用场景，例如交通流分配问题可以看作一对多（one to all）的单源最短路问题，智能导航系统可以看作实时路况条件下一对一（one to one）的多源最短路问题。通常在求解问题时我们不仅要关注结果，更要关注求解过程，即算法的效率，因为它关系到解决问题的成本。F. Benjamin Zhan 和 Charles E. Noon[1]以实际路网数据做了大量数据实验，对 15 种最短路径算法的效率做出了客观评估，这里直接引用其研究成果（如表 1-1，1-2）。

表 1-1　Relative Performance Summary for Data Set2 with a Scaling Factor of 1000

Algorithm	Relative Performance by Network										Overall Performance		Average Max-to-Mean Ratio
	NE1	AL1	MN1	IA1	MS1	SC1	FL1	MO1	LA1	GA1	Total time	Ratio	
PAPE	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	15.12	1.00	1.10
TWO_Q	1.08	1.08	1.08	1.08	1.09	1.08	1.07	1.07	1.08	1.06	16.22	1.07	1.10
THRESH	1.87	1.58	1.52	1.63	1.49	1.44	1.51	1.46	1.46	1.33	22.26	1.47	1.13
BFP	1.43	1.57	1.59	1.75	2.36	1.95	1.99	2.04	2.46	2.51	31.96	2.11	1.67
DIKBA	4.33	2.86	2.91	2.82	2.34	2.15	2.31	2.22	2.15	1.95	35.41	2.34	1.08
DIKB	4.33	2.87	2.92	2.85	2.35	2.16	2.33	2.23	2.16	1.97	35.61	2.36	1.09
GOR	2.47	2.47	2.49	2.46	2.59	2.50	2.42	2.52	2.50	2.47	37.61	2.49	1.10
DIKBM	4.63	3.20	3.14	3.29	2.66	2.42	2.53	2.61	2.33	2.31	39.98	2.64	1.28
DIKBD	3.75	3.48	3.29	3.36	2.95	2.90	2.96	3.06	2.88	2.67	45.25	2.99	1.04
BF	1.74	2.09	2.21	2.25	3.10	2.65	3.34	3.15	3.89	4.05	48.35	3.20	1.79
DIKQ	3.24	4.39	4.07	4.15	4.17	5.12	3.93	6.06	5.16	5.88	75.09	4.97	1.31
DIKH	4.61	5.37	4.71	5.12	4.87	5.26	4.68	5.37	5.28	5.31	77.47	5.12	1.48
DIKR	6.37	6.08	5.82	5.98	5.20	5.20	5.24	5.49	5.25	4.78	80.69	5.34	1.02
DIKF	7.59	8.25	7.82	8.19	7.57	8.42	7.73	8.94	8.57	8.23	124.63	8.24	1.07
GOR1	6.90	7.70	7.26	7.67	7.59	8.51	8.83	9.03	9.57	8.86	129.70	8.58	1.29
CPU TIME Of minimum	0.46	0.73	0.90	0.86	1.09	1.63	2.08	2.24	2.25	2.87	15.12	--	--

表 1-2　Relative Performance Summary for Data Set2 with a Scaling Factor of 1000

Algorithm	Relative Performance by Network										Overall Performance		Average Max-to-Mean Ratio
	LA2	MS2	NE2	FL2	SC2	IA2	MN2	AL2	MO2	US2	GA2	Total time	Ratio	
TWO_Q	1.05	1.02	1.00	1.00	1.01	1.02	1.00	1.00	1.01	1.00	1.00	2.95	1.00	1.16
PAPE	1.19	1.00	1.12	1.08	1.00	1.00	1.08	1.03	1.00	1.05	1.00	3.05	1.03	1.33

THRESH	1.00	1.33	1.39	1.06	1.42	1.72	1.47	1.28	1.59	1.36	1.43	4.11	1.39	1.13
DIKBA	1.17	1.60	1.56	1.14	1.58	1.95	1.69	1.44	1.75	1.30	1.60	4.53	1.53	1.11
DIKB	1.18	1.60	1.56	1.14	1.60	1.95	1.69	1.44	1.76	1.30	1.60	4.55	1.54	1.11
DIKBM	1.18	1.62	1.57	1.14	1.60	1.98	1.71	1.44	1.79	1.33	1.62	4.60	1.56	1.12
GOR	1.69	1.65	1.51	1.65	1.65	1.53	1.57	1.72	1.63	1.66	1.69	4.79	1.62	1.14
DIKBD	1.34	1.74	1.73	1.24	1.68	2.11	1.83	1.53	1.87	1.49	1.69	4.92	1.67	1.10
DIKR	1.78	2.18	2.26	1.58	2.07	2.78	2.40	1.93	2.39	2.02	2.13	6.35	2.15	1.12
DIKH	2.31	2.72	2.72	1.95	2.62	3.52	3.03	2.46	3.04	2.41	2.76	7.97	2.70	1.18
DIKF	4.04	4.66	4.23	3.32	4.21	5.23	4.80	4.07	4.79	3.80	4.44	12.73	4.31	1.14
GOR1	9.59	11.41	8.24	10.38	10.18	9.12	9.46	210.58	11.04	11.24	10.82	30.15	10.21	1.40
BFP	9.24	10.86	9.61	17.80	12.21	10.08	10.48	14.92	13.74	21.99	16.02	41.54	14.06	1.94
DIKQ	5.98	13.45	20.69	4.82	18.88	39.89	33.76	21.48	30.88	19.67	32.35	71.21	24.11	1.73
BF	19.35	24.54	22.75	37.69	26.23	23.83	23.95	31.57	32.59	44.66	34.28	90.44	30.62	1.99
CPU TIME Of Minimum	0.12	0.15	0.22	0.21	0.24	0.28	0.30	0.30	0.30	0.39	0.43	2.95	--	--

注释：①NE1,AL1,…GA1,LA2,MS2,…GA2 为实际路网编号；②CPU TIME Of Minimum 为最佳最短路径算法的最小平均 CPU 运算时间（单位：毫秒），与算法对应的每一行给出了相应算法在求解不同路网最短路径时平均 CPU 运算时间与最小平均 CPU 运算时间的比值。例如表 1-1 中 PAPE 算法是求解 NE1 网络的最佳算法，其最小平均 CPU 运算时间为 0.46 毫秒，DIKF 是求解 NE1 路网的最差算法，其平均 CPU 运算时间为 0.46*7.59=3.49 毫秒；③Total Time 列为算法求解所有路网单源最短路径的 CPU 运算时间之和，Ratio 为每个算法的 Total Time 与最佳算法的 Total Time 比值，代表算法总体速度；④Average Max-to-Mean Ratio 为在重复实验中最大 CPU 运算时间与平均运算时间的比值（该实验结果是由 100 次独立重复试验得到的）。

表 1-1 和表 1-2 是在不同实际路网上的统计结果，其中 TWO_Q 又称 Deque Label Correcting Algorithm，而 DIXX 则是不同版本的 Dijkstra algorithm 实现。根据实验结果我们可以得知在两种不同的数据集上 TWO_Q 都表现出了较好的性能，而 Dijkstra algorithm 的性能不太令人满意。因此，我们有必要学习诸如 TWO_Q 一样高效的算法来更好的解决实际问题。此外，含有负环的网络在实际应用中也是极为常见（例如，鼓励拼车和使用不同类型的(负)拉格朗日乘数

来确保每个乘客在车辆路径问题中只被服务一次[2]），因此在选择算法时也必须考虑算法对含有负环网络的适用性。

本篇文档将围绕 Label Correcting Algorithm（标号更新法或标号更正法）展开，主要介绍一些基本的 Label Correcting Algorithms，为读者后续深入学习网络最短路问题以及其他网络流问题提供支撑。本文后续章节安排如下：第二章节主要介绍最短路问题及其数学模型、最短路径求解算法及其分类；第三章节主要介绍单源及多源 Label Correcting Algorithms 的核心内容与相应代码实现；第四章节主要介绍如何利用本文介绍的算法求解多目标最短路径问题以及如何处理大规模网络；在附录 1 部分补充了 Label Correcting Algorithm 如何处理含有负环的网络最短路径问题，附录 2 给出了本文所研究的简单有向图，附录 3 提供了由周学松老师开发的 NeXTA 软件，辅助最短路问题学习。为了更好的理解本文所介绍内容建议读者具备一定的图论知识以及计算机编程能力（Python or Matlab）。

Data and source code folder:

https://github.com/marcolee19970823/label_correcting

https://github.com/PariseC/Shortest_Path_Algorithm/tree/master/label_correcting_algorithm

Our github sites:

https://github.com/marcolee19970823

https://github.com/PariseC

Nexta network editor:

https://github.com/xzhou99/NeXTA-GMNS

完成本文内容学习以后，读者可以学习以下深层次最短路径的相关问题：

1. 算法性能分析（Algorithm analysis）；
2. 最小费用流问题（Minimum cost flow）；
3. 拉格朗日松弛问题（Lagrangian relaxation）；
4. 多商品流问题（Multicommodity flows）。

这些进阶内容的学习资源可以在如下网站获取：

https://ocw.mit.edu/courses/sloan-school-of-management/15-082j-network-optimization-fall-2010/assignments/

本文档内容主要参考《NETWORK FLOWS》[3]完成，另外上述进阶内容也可参考该文献进行学习。另外，关于本文档介绍的基础算法也可参考《Development and testing of dynamic traffic assignment and simulation procedures for ATIS/ATMS applications》[4]进行学习。

2. 最短路径问题

2.1 问题描述

在开始介绍最短路问题之前我们先来简单讨论网络流（network flow problems）问题。在我们日常生活中，网络无处不在：为我们提供电力能源的电力网络，为我们提供方便通讯的电话网络，满足我们各种出行需求的交通网络。在所有这些问题领域，我们都希望某些实体(电力、消费品、一个人或一辆车,一个消息)从一个点到另一个点尽可能需要少的费用以及获取最大的效益。这就是网络流问题的实质。根据不同的研究目的网络流问题可分为最短路径问题（shortest path problem）、最大流问题（maximum flow problem）、最小费用流问题（minimum cost flow problem）、最小费用最大流问题（minimum cost maximum flow problem）等等。

作为网络流问题的研究内容之一，最短路问题主要解决在网络中从一个节点到另一个节点成本最低的路径是什么。一种最通用的最短路问题可以如此描述：希望在网络中找到一条从源节点（source node）s到接收节点（target node）t的最小成本路径，这里的最小成本可定义为路径长度、旅行时间、旅行费用等。

2.2 应用领域

二十世纪六十年代，在最短路问题的研究上已经颇有成效，该问题在计算机科学、运筹学等学科的研究中一直是一个热点问题。最短路问题在现实应用中也相应的代表了最低成本、最短时间问题等。该问题作为网络流学科中的经典问题，以其丰富的适用性，具有广泛的应用领域。单就交通运输而言，最短路问题就已经有如下重要应用。

表 2-1 最短路问题在交通领域应用

应用领域	相关文献
车辆路径规划	毕明华. 动态物流中多点多源最佳路径算法研究与实现[D]. 浙江理工大学,2019.

公共交通换乘方案及线路规划	张妍. 随机环境下的地铁换乘问题两阶段优化模型[D].北京交通大学,2016. 牛学勤,王炜.基于最短路搜索的多路径公交客流分配模型研究[J].东南大学学报(自然科学版),2002. 马良河,刘信斌,廖大庆.城市公交线路网络图的最短路与乘车路线问题[J].数学的实践与认识,2004.
航空调度	田倩南. 面向航空调度中机场任务指派与受扰航班恢复问题的研究[D].华中科技大学,2018.
供应链管理	郑金忠,陈宏纪,李兴涛,李友虎.基于供应链的航材配送最短路算法[J].物流技术,2004.
铁路运输调度指挥	苗义烽. 突发事件下的列车运行调度模型与算法研究[D].中国铁道科学研究院,2015. Meng, L., & Zhou, X. Simultaneous train rerouting and rescheduling on an N-track network: A model reformulation with network-based cumulative flow variables[J].Transportation Research Part B: Methodological, 2014, 67: 208-234.
铁路运输计划编制	陈枭楠. 基于均衡性的高速铁路乘务计划一体化优化方法研究[D].北京交通大学,2019.
交通流量分配	Zhou X , Taylor J , Pratico F . DTALite: A queue-based mesoscopic traffic simulator for fast model evaluation and calibration[J]. Cogent Engineering, 2014. 颜佑启,欧阳建湘.最短路—最大流交通分配法[J].中国公路学报,2005. 柳伍生,贺剑,李甜甜,谌兰兰.出行策略与行程时间不确定下的公交客流分配方法[J].交通运输系统工程与信息,2018.
行人出行	Shatu F, Yigitcanlar T. Development and validity of a virtual street walkability audit tool for pedestrian route choice analysis — SWATCH[J]. Journal of transport geography, 2018.
物流运输	Mahmoudi M , Zhou X . Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state-space-time network representations[J]. 2015. 张运河,林柏梁,梁栋,高红艳.优化多式联运问题的一种广义最短路方法研究[J].铁道学报,2006.
随机条件下路径规划	Yang, L., & Zhou, X. Constraint reformulation and a Lagrangian relaxation-based solution algorithm for a least expected time path problem[J]. Transportation Research Part B: Methodological. 2014.. Xing T , Zhou X . Finding the most reliable path with and without link travel time correlation: A Lagrangian substitution based approach[J]. Transportation Research Part B: Methodological, 2011.

2.3 数学模型

这里给出通用单源最短路径数学模型描述：$G=(N,A)$，G是由节点集合N（元素个数为n）和弧集合A（元素个数为m）组成的网络，弧(i,j)长度为c_{ij}。定义节点$s \in N$为源节点（source），其他节点$i \in N$为非源节点（non-source），路径长度为该路径所包含弧的长度之和。求解单源最短路径问题就是找出源节点s到每一个非源节点i的有向最短路径。最短路径问题的数学模型如下：

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1, & \text{if } i=s \\ -1, & \text{if } i=t \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

x_{ij}：0-1 决策变量，$x_{ij}=1$表示经过弧(i,j)，$x_{ij}=0$表示不经过弧(i,j).

我们可以采用 GAMS 软件实现上述最短路径模型，并得到准确最优解。这里给出一个 GAMS 求解 Chicago network 的简单案例（https://github.com/xzhou99/learning-transportation/tree/master/GAMS_code%20-space-time-network/1%20shortest_path ）

表 2-2　GAMS 建模过程

```
variable z;
    binary variables
    x(a,i,j ) selection of agent a between i and j;

    equations
    so_obj
    comm_flow_on_node_origin(a,i) origin node flow of agent a on node i at time t
    comm_flow_on_node_intermediate(a,i) intermediate node flow of agent a on node i at time t
    comm_flow_on_node_destination(a,i) destination node flow of agent a on node i at time t;

    so_obj.. z =e= sum (a,sum((i,j)$(arcs(a,i,j)>0.1), x(a,i,j)*travel_cost(a,i,j)));

    Model SP /ALL/ ;
solve SP using MIP minimizing z;
```

此外，本文所研究的最短路径问题无特殊说明外，均具有以下假设：

- 所有弧长均为整数值
- 网络包含从节点s到网络中所有其他节点的有向路径
- 网络不包含负循环
- 网络为有向图

2.4 最短路算法

面对最短路径问题我们可以通过求解整数或线性（详见：https://en.wikipedia.org/wiki/Unimodular_matrix）规划模型的方法求解，然而这种做法并不高效，当网络含有负环或者网络规模较大时现有计算能力很难对其求解。因此需要更高效的算法来求解最短路径问题。由于最短路径问题的特殊性，基于图论开发出了许多有效的迭代算法，例如：Dijkstra 算法、Floyd-Warshall 算法、Bellman-Ford 算法等等。表 2-3 罗列了常见的最短路算法，这些算法被分为两类：Label Setting Algorithm 和 Label Correcting Algorithm。

表 2-3 常见最短路算法分类

算法序号	算法名称	算法归类
1	Dijkstra	Label Setting Algorithm
2	Dial	
3	Heap	
4	Generic (Bellman-Ford)	Label Correcting Algorithm
5	FIFO	
6	Deque	

这两类算法基本出发点是相同的：在每次迭代时为每个非源节点i分配一个临时距离标签，作为源节点s到节点i最短路径的估计值；不同的是它们如何更新临时距离标签：Label Setting Algorithm，在每次迭代时将当前临时距离标签最小的更新为永久距离标签，直到所有的临时距离标签都更新为永久距离标签；而 Label Correcting Algorithm 在每次迭代时都有可能更新临时距离标签的值，直到最后一次迭代时所有的临时距离标签才成为永久距离标签。正因为其更新机制不同，他们所适用的最短路径问题也是有所区别的（如下表）。

表 2-4 Label Setting Algorithms 与 Label Correcting Algorithms 适用条件

算法	适用条件

（标准）Label Setting Algorithm	无环网络 （Dijkstra Algorithm 无法求解含负权的网络最短路径问题）
Label Correcting Algorithm	所有类型（含有负弧长、负环）

接下来我们以 Dijkstra algorithm 为例，证明标准 Label Setting Algorithm 对于含有负环网络的不适用。令S为永久距离标签对应的节点集合，$\bar{S}$为非永久距离标签对应的节点集合，N为网络节点集合，n为网络节点个数，$d(i)$表示源节点s到非源节点i的临时距离标签，$pred(i)$表示非源节点的前向节点，$A(i)$表示从节点i发出的所有弧的集合（适用于本文所有符号表示）。Dijkstra algorithm 伪代码如下：

表 2-5　Dijkstra Algorithm

Algorithm 1 Dijkstra

```
begin
    S := ∅;  S̄ := N;
    d(i) := ∞  for each node  i ∈ N;
    d(s) := 0  and  pred(s) := 0;
    while  |S|  < n  do
    begin
        let  i  ∈ S̄  be a node for which  d(i) = min{d(j):  j  ∈ S̄};
        S := S  ∪  {i};
        S̄ := S̄ –  {i};
          for each arc  (i, j) ∈ A(i)  do
            if  d(j)  > d(i) + c_ij  then  d(j) := d(i) + c_ij  and  pred(j) := i;
    end;
end;
```

根据表 2-5，求解图 2-1 中从节点 1 到其他节点最短路径过程为：①令$S=\emptyset,\bar{S}=N,d(1)=0,d(i)=\ \infty,i=2,\ldots,6,\mathrm{pred}(1)=0$；②从$\bar{S}$中选择节点 1（距离标签最小）作为当前节点，并将其从$\bar{S}$移到S中（此时节点 1 标记为永久节点），之后根据判别条件将节点 2 和 3 的临时距离标签更新为$d(2)=6,d(3)=4$，前向节点为$pred(2)=1,\ pred(3)=1$；③继续从$\bar{S}$中选择节点 3（距离标签最小）作为当前节点，并将其从$\bar{S}$移到S中（此时节点 3 标记为永久节点），之后根据判别条件将节点 5 的临时距离标签更新为$d(5)=6$,前向节点为$pred(5)=3$；④继续从$\bar{S}$中选择节点 5（距离标签最小）作为当前节点，并将其从$\bar{S}$移到S中（此时节点 5 标记为永久节点），之后根据判别条件将节点 4 和 6 的临时距离标签更新为$d(4)=4,d(6)=9$，前向节点为$pred(4)=5$，$pred(6)=5$；⑤继续从$\bar{S}$中选择节点 4 作为当前节点，并将其从$\bar{S}$移到S中（此时节点 4 标记为永久节点），此时没有可更新的距离标签；⑥继续从$\bar{S}$中选择节点 2 作为当前节点，并将其从$\bar{S}$移到S中（此时节点 2 标记为永久节点），此时没有可更新的距离标签；⑦继

续从$\bar{S}$中选择节点 6 作为当前节点，并将其从$\bar{S}$移到S中（此时节点 6 标记为永久节点），此时没有可更新的距离标签，且$\bar{S}$为空，算法结束。由此得到节点 1 到其他各节点的最短路径及其长度：1-2（6），1-3（4）,1-3-5-4（4），1-3-5（6），1-3-5-6（9）。但应注意到节点 3、4、5 构成一个负环（负环长度为-1），只要经过一次路径长度就减少 1，因此可以无限减少节点 1 到节点 3,4,5,6 的距离。由此可见，Dijkstra 无法正确求解含有负环的网络最短路径问题，。其他 Label Setting Algorithm 的情况类似，可自行举出反例进行验证。

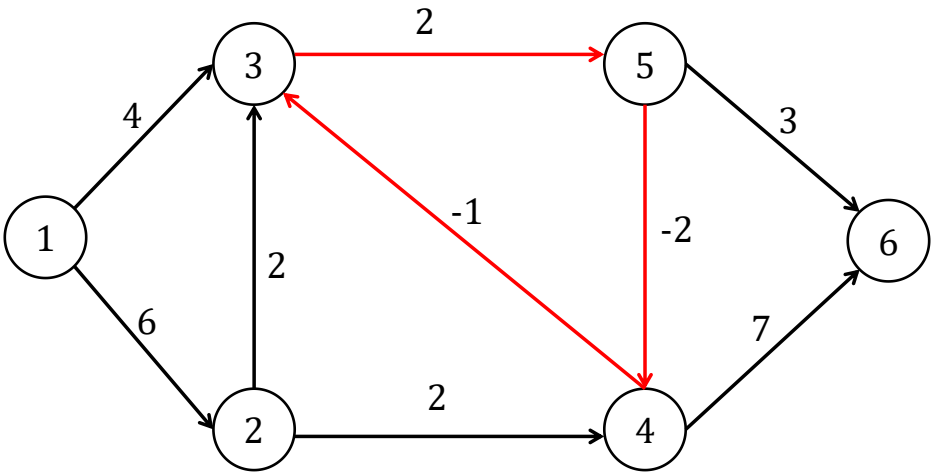


图 2-1　含有负环的有向图（负环用红色标识）

以上我们通过反例验证 Label Setting Algorithm 不适合处理含负环网络的最短路径问题，Label Correcting Algorithm 能否处理这种情况我们将在后续章节进行详细探讨。

3. Label Correcting Algorithms

本章将围绕 Label Correcting Algorithms 展开。首先，3.1 小节介绍了最短路径最优性条件，这些条件允许我们评估一组距离标签是否达到最优，以及什么时候我们应该结束算法。基于这一最优性条件，3.2-3.5 小节介绍了基本的 Label Correcting Algorithms 用于求解不含有负环的单源最短路径问题。对于多源最短路径问题将在 3.6 小节进行讨论，3.7 小节将对本章内容进行总结。

在正式介绍内容之前我们做一下约定：①本文以有向图作为研究对象；②网络中不含有负环；③网络弧长均为整数；④在实现相应算法时以表格 3-1 为输入文件。

表 3-1　算法输入文件格式

文件	内容

node.csv

A	B	C	D	E
node_id	x_coord	y_coord	node_type	zone_id
1	0	2	1	1
2	1	0	0	2
3	1	3	0	3
4	3	0	0	4
5	3	3	0	5
6	4	2	0	6

road_link.csv

A	B	C	D
road_link_id	from_node_id	to_node_id	length
1	1	2	6
2	1	3	4
3	2	3	2
4	2	4	2
5	3	4	1
6	3	5	2
7	4	6	7
8	5	4	1
9	5	6	3

node_type：1 为源节点，0 位非源节点；

zone_id：网络节点所属类别（为了辅助工具 NeXTA 需要，默认与 node_id 一致。详见附录 3）。

3.1 最优性判别条件

最优性定理 1：对于任意节点$j \in N$，设$d(j)$表示从源节点s到节点j的某条有向路径的长度，则$d(j)$当且仅当满足以下最短路径最优性条件时为源节点s到节点j最短路径距离:

$$\mathrm{d(j)} \le \mathrm{d(i)} + c_{ij}, \forall (i,j) \in A \qquad (3)$$

式（3）对于网络中任意弧(i,j)，源点s到节点j的最短路径长度$d(j)$始终小于等于源点s到节点i的最短路径长度与弧(i,j)的长度之和。反之，如果存在某些弧(i,j)满足$d(j) > d(i) + c_{ij}$，那么我们就可以把节点i加到源节点s到节点j的路径中去，从而降低源节点s到节点j的最短路径长度，这与$d(j)$是最短路径长度的命题相矛盾。

接下来我们从数学的角度再次证明上述定理的正确性。假设源点s到任意节点j的某条有向路径为P：$s = i_1 - i_2 - i_3 \ldots - i_k = j$,由式（3）可得：

$$\begin{gathered} \mathrm{d(j)} = \mathrm{d}(i_k) \le \mathrm{d}(i_{k-1}) + c_{i_{k-1}i_k} \\ \mathrm{d}(i_{k-1}) \le \mathrm{d}(i_{k-2}) + c_{i_{k-2}i_{k-1}} \\ \cdots \\ \mathrm{d}(i_2) \le \mathrm{d}(i_1) + c_{i_1 i_2} = c_{i_1 i_2} \\ \text{注：} \mathrm{d}(i_1) = d(s) = 0 \end{gathered} \qquad (4)$$

把上述不等式相加可得到：

$$\mathrm{d(j)} = \mathrm{d}(i_k) \le c_{i_{k-1}i_k} + c_{i_{k-2}i_{k-1}} + \cdots + c_{i_1 i_2} = \sum_{(i,j)\in P} c_{ij} \quad (5)$$

式（5）说明$d(j)$是从源节点s到节点$j(j \neq s)$的任意有向路径长度的下界，又因为$d(j)$是源节点s到节点j的临时有向路径长度，因此它又是最短路径的上界，所以$d(j)$即为源节点s到节点j的最短路径长度。

在此，我们对定理 1 做进一步拓展：定义c_{ij}^{d}表示弧(i,j)关于距离标签$d(*)$的缩短距离，其计算公式为：$c_{ij}^{d} = c_{ij} + d(i) - d(j)$,关于$c_{ij}^{d}$有以下三条性质：

1. 在任意有向环 W 中，$\sum_{(i,j)\in W} c_{ij}^{d} = \sum_{(i,j)\in W} c_{ij}$;
2. 对于从节点k到节点 1 的任意有向路径 P ，$\sum_{(i,j)\in P} c_{ij}^{d} = \sum_{(i,j)\in P} c_{ij} + d(k) - d(l)$;
3. 如果$d(*)$是网络中的一条最短路径，则$c_{ij}^{d} \ge 0, \forall (i,j) \in A$;

接下来思考：如果网络中存在负环，上述三条性质是否还成立？假设 W 是网络 G 中的一个有向环，由上述性质 3 可推出性质 1 中$\sum_{(i,j)\in W} c_{ij}^{d} = \sum_{(i,j)\in W} c_{ij} \ge 0$，因此 W 不可能是负环。由此得出：**含有负环的网络不满足定理 1**。

此外，本文所介绍的最优性判别条件与动态规划中的 Bellman Optimality Condition 是一致的。

3.2 Generic Label Correcting Algorithm

3.2.1 算法介绍

如上文所介绍的最优性判断法则，本文所介绍的 Generic Label Correcting Algorithm 可以认为是 Bellman Optimality Condition 的具体实现。

在 Generic Label Correcting Algorithm 中以距离标签$d(j)$来表示源节点s到任意节点j的最短路径长度，但与 Label Setting Algorithms 不同的是：这里不区分永久距离标签和临时距离标签，在算法迭代中距离标签是连续变化的，当所有距离标签都满足最优性条件时算法结束，此时距离标签即为源节点s到任意节点j的最短路径长度。Generic Label Correcting Algorithm 步骤如下：

表 3-2　Generic Label Correcting Algorithm

Algorithm 2 Generic Label Correcting Algorithm
1:　begin
2:　　$d(s) := 0$, and $\mathbf{pred}(s) := 0$;
3:　　$d(j) := \infty$ for each node $j \in N - \{s\}$;
4:　　while some arc (i,j) satisfies $d(j) > d(i) + c_{ij}$ do
5:　　begin
6:　　　$d(j) := d(i) + c_{ij}$;
7:　　　$\mathbf{pred}(j) := i$;
8:　　end;
9:　end;

我们可以看出 Generic Label Correcting Algorithm 的伪代码很简洁，核心就是逐个检查不满最优性条件的距离标签，并根据$d(j) = d(i) + c_{ij}$更新距离标签，同时前向节点也随之更新，直到所有的距离标签都满足最优性条件。这里以附录 2 为例，求解节点 1 到其他节点的最短路径：①令节点 1 的距离标签 *d(1)*=0，前向节点 *pred(1)*=0，其他节点的距离标签设为无穷大，如 3-1（a）；②检查弧（1,3），（1,2）是否满足最优性条件，并更新相应距离标签及前向节点，如图 3-1（b）；③检查弧（3,4），（3,5）是否满足最优性条件，并更新相应距离标签及前向节点，如图 3-1（c）；④检查弧（5,6），(5,4）是否满足最优性条件，并更新相应距离标签及前向节点，如图 3-1（d）。至此图 3-1（d）中的所有弧都满足最优性条件，我们可以通过前向节点集合来生成节点 1 到其他节点的最短路径，例如，节点 5 的前向节点为 3，节点 3 的前向节点为 1，因此节点 1 到节点 5 的最短路径为 1-3-5。通过这种方法我们可以得到一颗以节点 1 为根的前向节点树（如图 3-2），此前向节树记录了根节点到其他子节点的最短路径。

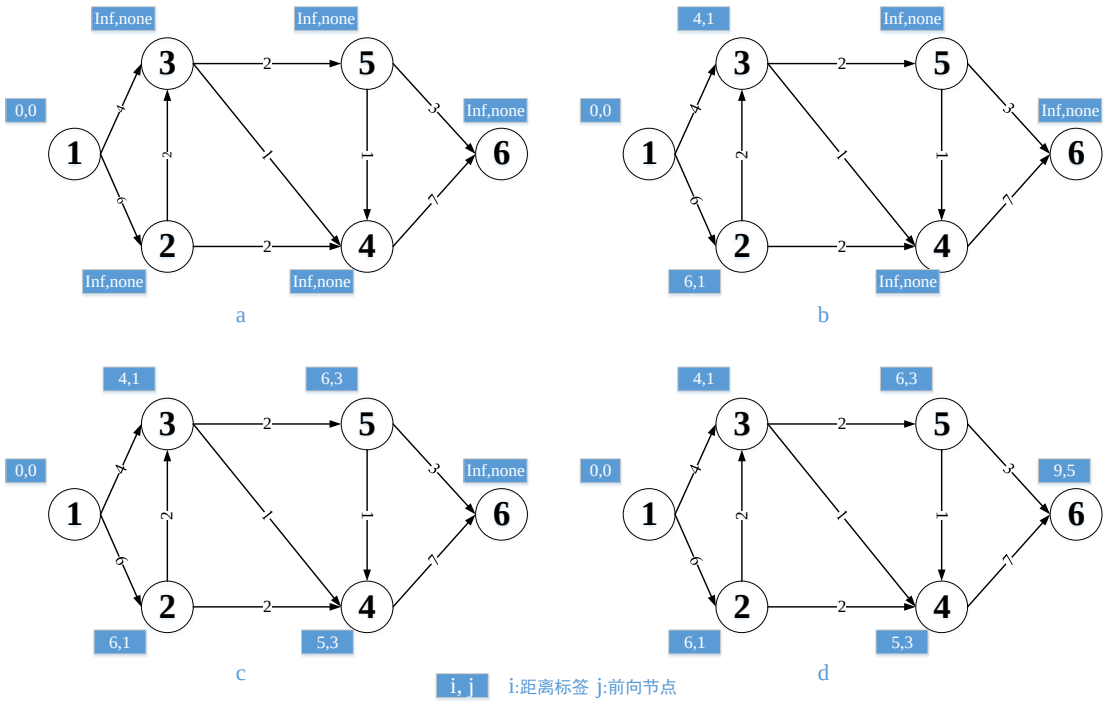


图 3-1　Generic Label-Correcting Algorithm 计算流程

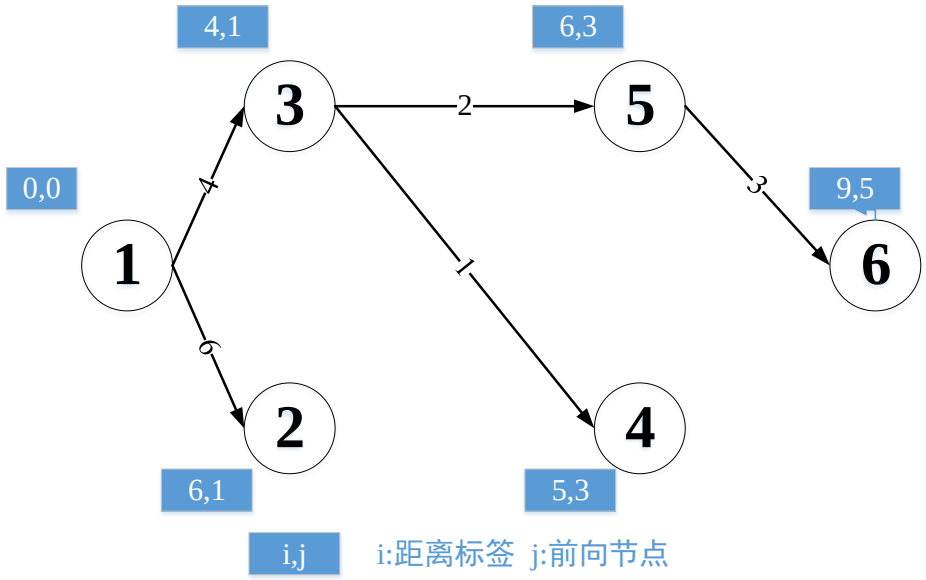


图 3-2　前向树

这里需要说明的是前向节点集合并不一定会形成一棵以源节点为根的树。如图 3-3（a）所示，假设弧（4,1）满足 $d(1)>d(4)+c_{41}$,我们将节点 1 的前向节点由 0 更改为 4，得到图 3-3（b），此时前向节点不再构成一棵树。之所以会发生这种情况是因为网络中含有**负环**，在 3.1 小节我们已经讨论过，含有负环的网络不符合最优性定理。

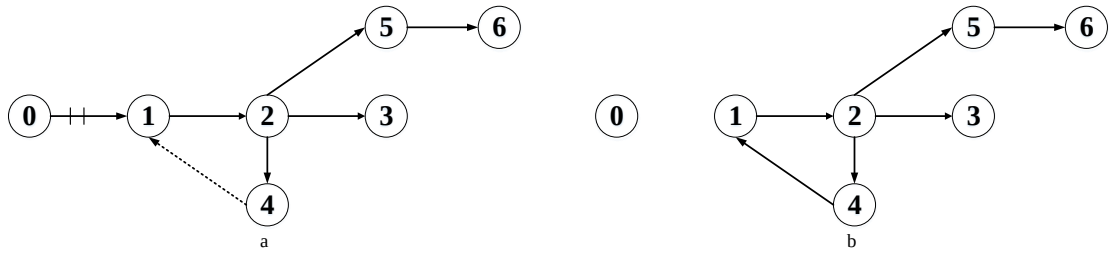


图 3-3　含有环的前向图

算法复杂度分析

接下来我们对 Generic Label Correcting Algorithm 的收敛性进行分析。通过伪代码我们得知算法只有一个 while 循环，但这个循环并没有明确指出迭代次数的值。我们假设C为最大的弧长值，那么源节点到其他节点的路径长度上界为nC（该路径含有 n-1 条弧，每条弧的长度为C），路径长度下界为$-nC$，所以对于任意距离标签$d(j)$的最大更新次数为$2nC$（假设每次更新距离标签只减少 1 单位），网络中节点数目为 n ，因此距离标签总的更新次数为$2n^2C$。因为每次迭代只更新一个距离标签，因此总的迭代次数为$\boldsymbol{O(n^2C)}$。

3.2.2 算法实现

首先给出 Python 版本的 Generic Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-3　Python 实现 Generic Label Correcting Algorithm

```
"""导入相关基础包，定义全局变量"""
import pandas as pd
import numpy as np
import copy
g_node_list=[] #网络节点集合
g_link_list=[] #网络节点类别集合
g_node_zone={} #网络弧集合
g_shortest_path=[]#最短路径集合
g_origin=None   #网络源节点
g_number_of_nodes=0#网络节点个数
node_predecessor=[]#前向节点集合
node_label_cost=[]#距离标签集合
Max_label_cost=99999#初始距离标签
"""导入网络数据文件，构建基础网络并初始化相关变量"""
#读取网络节点数据
df_node=pd.read_csv('node.csv')
df_node=df_node.iloc[:,:].values
for i in range(len(df_node)):
    g_node_list.append(df_node[i,0])
    g_node_zone[df_node[i,0]]=df_node[i,-1]
    g_number_of_nodes+=1
    if df_node[i,3]==1:
        g_origin=df_node[i,0]
Distance=np.ones((g_number_of_nodes,g_number_of_nodes))*\
```

```
Max_label_cost #距离矩阵
node_predecessor=[-1]*g_number_of_nodes
node_label_cost=[Max_label_cost]*g_number_of_nodes
node_predecessor[g_origin-1]=0
node_label_cost[g_origin-1] = 0
#读取网络弧数据
df_link=pd.read_csv('road_link.csv')
df_link=df_link.iloc[:,:].values
for i in range(len(df_link)):
    g_link_list.append((df_link[i,1],df_link[i,2]))
    Distance[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,3]
"""最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签"""
while True:
    v=0# 未满足最优性条件的节点个数
    for head,tail in g_link_list:
        if node_label_cost[tail-1]>\
node_label_cost[head-1]+Distance[head-1,tail-1]:
            node_label_cost[tail-1]=\
node_label_cost[head-1]+Distance[head-1,tail-1]
            node_predecessor[tail-1]=head
            v=v+1
    if v==0:
        break
"""依据前向节点生成最短路径"""
agent_id=1
o_zone_id=g_node_zone[g_origin]
for destination in g_node_list:
    if g_origin!=destination:
        d_zone_id=g_node_zone[destination]
        if node_label_cost[destination-1]==Max_label_cost:
            path=" "
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,
path,node_label_cost[destination-1]])
        else:
            to_node=copy.copy(destination)
            path= "%s"%to_node
            while node_predecessor[to_node-1]!=g_origin:
                path="%s;"%node_predecessor[to_node-1]+path
                g=node_predecessor[to_node-1]
                to_node=g
            path="%s;"%g_origin+path
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        agent_id+=1
"""将求解结果导出到 csv 文件"""
#将数据转换为 DataFrame 格式方便导出 csv 文件
g_shortest_path=np.array(g_shortest_path)
col=['agent_id','o_zone_id','d_zone_id','node_sequence','distance']
file_data = pd.DataFrame(g_shortest_path,
index=range(len(g_shortest_path)),columns=col)
file_data.to_csv('agent.csv', index=False)
```

接下来给出 MATLAB 版本的 Generic Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-4　MATLAB 实现 Generic Label Correcting Algorithm

```
%% clear
clc
clear

%% 设置变量
g_node_list = []; %网络节点集合
g_link_list = []; %网络弧集合
g_origin = [];   %网络源节点
g_number_of_nodes = 0;%网络节点个数
node_predecessor = [];%前向节点集合
node_label_cost = [];%距离标签集合
Max_label_cost = inf; %初始距离标签

%% 导入网络数据文件，构建基础网络并初始化相关变量
%读取网络节点数据
df_node = csvread('node.csv',1,0);
g_number_of_nodes = size(df_node,1);
g_node_list = df_node(:,1);
for i = 1 : g_number_of_nodes
    if df_node(i,4)==1
        g_origin=df_node(i,1);
        o_zone_id = df_node(i,5);
    end
end
Distance = ones(g_number_of_nodes,g_number_of_nodes)*Max_label_cost;
%距离矩阵
node_predecessor = -1*ones(1,g_number_of_nodes);
node_label_cost = Max_label_cost*ones(1,g_number_of_nodes);
node_predecessor(1,g_origin) = 0;
node_label_cost(1,g_origin) = 0;
%读取网络弧数据
df_link = csvread('road_link.csv',1,0);
g_link_list = df_link(:,2:3);
for i = 1 : size(df_link, 1)
    Distance(df_link(i,2),df_link(i,3)) = df_link(i,4);
end

%% 最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签
while 1
    v=0; %未满足最优性条件的节点个数
    for i = 1 : size(df_link, 1)
        head = g_link_list(i,1);
        tail = g_link_list(i,2);
        if node_label_cost(tail)>…
node_label_cost(head)+Distance(head,tail)
            node_label_cost(tail)=node_label_cost(head)+…
Distance(head,tail);
            node_predecessor(tail)=head;
            v=v+1;
        end
    end
    if v==0
        break;
    end
end

```

```
%% 依据前向节点生成最短路径
distance_column = [];
path_column = {};
o_zone_id_column = o_zone_id * ones(g_number_of_nodes-1, 1);
d_zone_id_column = [];
agent_id_column = [1:(g_number_of_nodes-1)]';
for i = 1: size(g_node_list, 1)
    destination = g_node_list(i);
    if g_origin ~= destination
        d_zone_id_column = [d_zone_id_column; df_node(i,5)];
        if node_label_cost(destination)==Max_label_cost
            path="";
            distance = 99999;
            distance_column = [distance_column; 99999];
        else
            to_node=destination;
            path=num2str(to_node);
            while node_predecessor(to_node)~=g_origin
                path=strcat(';',path);
                path=strcat(num2str(node_predecessor(to_node)),path);
                g=node_predecessor(to_node);
                to_node=g;
            end
            path=strcat(';',path);
            path=strcat(num2str(g_origin),path);
            distance_column = [distance_column; node_label_cost(i)];
        end
        path_column=[path_column;path];
    end
end

title = {'agent_id','o_zone_id','d_zone_id','node_sequence','distance'};
result_table=table(agent_id_column,o_zone_id_column,…
d_zone_id_column,path_column,distance_column,'VariableNames',title);
writetable(result_table, 'agent.csv',…
'Delimiter',',','QuoteStrings',true)

```

3.3 Modified Label Correcting Algorithm

3.3.1 算法介绍

这里先回顾一下 Generic Label Correcting Algorithm 的核心步骤：对违反最优性条件的弧，更新其对应节点的距离标签及前向节点，即表 3-2 第 4 行。我们可以看出在以最优性条件检查距离标签时该算法并没有给出具体的规则，可以认为是一种遍历的方式。因此我们在实现代码时，即表 3-3 第 39 行，表 3-4 第 39 行，默认采取对所有弧进行遍历。显然这是一个简单但低效的方法。本小节我们将介绍一种改进思路，称其为 Modified Label Correcting Algorithm，可有助于提高算法效率。

这里引入一个**可扫描列表** SE_LIST（Scan Eligible LIST），用来记录可能违反最优性条件的所有弧，如果 SE_LIST 为空，则表明所有弧都满足最优性条件，已找到最短路径，否则就从 SE_LIST 中选择一条弧(i,j)，判定其是否违反最优性条件，并将其从 SE_LIST 中移除。如果弧(i,j)违反了最优性条件，就用它更新节点j的距离标签，同时更新节点j的前向节点。此时请注意，节点j的距离标签的任何减少都会影响从节点j发出的所有弧的缩减长度c_{ij}^d，从而导致其中一些弧就可能违反了最优性条件，换句话说，**当节点j的距离标签更新时，它可能会导致从节点j发出的弧不满足最优性条件**。还要注意的是，节点j的距离标签更新对于所有流入节点j的弧都保持最优条件，也就是说**节点j的距离标签更新不会影响节点j的前向节点的最优性**。因此，如果节点j的距离标签发生改变，则应该将所有从节点j发出的弧,即$A(j)$，加入到 SE_LIST 中。又因为节点j唯一对应一个$A(j)$，所以我们可以只记录距离标签发生改变的节点编号，在进行扫描时取出与其对应的$A(j)$，这将有助于减少算法工作量，提高算法效率。以上就是 Modified Label Correcting Algorithm 的基本思路，其算法步骤如下。

表 3-5　Modified Label Correcting Algorithm

Algorithm 3 Modified Label Correcting Algorithm

```
1:    begin
2:        d(s) := 0 and pred(s) := 0;
3:        d(j) := ∞ for each node j ∈ N − {s};
4:        SE_LIST := {s};
5:        while SE_LIST ≠ ∅ do
6:        begin
7:            remove an element i from SE_LIST;
8:            for each arc (i, j) ∈ A(i) do
9:            if d(j) > d(i) + c_ij then
10:           begin
11:               d(j) := d(i) + c_ij ;
12:               pred(j) := i;
13:               if j ∉ SE_LIST then add node j to SE_LIST;
14:           end;
15:       end;
16:   end;
```

复杂度分析

在介绍 Generic Label Correcting Algorithm 的时候我们提到对于任意距离标签$d(j)$最多更新次数为$2nC$，因此 Modified Label Correcting Algorithm 总的迭代次数为$\sum_{i\in N}(2nC)|A(i)| = O(nmC)$。当$C$值特别大时，算法总的迭代次数为$O(2^n)$。

3.3.2 算法实现

首先给出 Python 版本的 Modified Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-6　Python 实现 Modified Label Correcting Algorithm

```
"""导入相关基础包，定义全局变量"""
import pandas as pd
import numpy as np
import copy
import random
#全局变量定义
g_node_list=[] #网络节点集合
g_node_zone={} #网络节点类别集合
g_link_list=[] #网络弧集合
g_adjacent_arc_list={}#节点邻接弧集合（从节点 i 发出的弧集合）
g_shortest_path=[]#OD 最短路径集合
g_node_status=[]#网络节点状态
g_origin=None  #网络源节点
g_number_of_nodes=0
node_predecessor=[]#前向节点集合
node_label_cost=[]#距离标签集合
SE_LIST=[]#可扫描节点集合
Max_label_cost=99999#初始距离标签
"""导入网络数据文件，构建基础网络并初始化相关变量"""
#读取网络节点数据
df_node=pd.read_csv('node.csv')
df_node=df_node.iloc[:,:].values
for i in range(len(df_node)):
    g_node_list.append(df_node[i,0])
    g_node_zone[df_node[i, 0]] = df_node[i, -1]
    g_number_of_nodes+=1
    g_adjacent_arc_list[df_node[i,0]]=[]
    if df_node[i, 3] == 1:
        g_origin = df_node[i, 0]
g_node_status=[0 for i in range(g_number_of_nodes)] #初始化网络节点状态
Distance=np.ones((g_number_of_nodes,g_number_of_nodes))*\
Max_label_cost #距离矩阵
node_predecessor=[-1]*g_number_of_nodes
node_label_cost=[Max_label_cost]*g_number_of_nodes
node_predecessor[g_origin-1]=0
node_label_cost[g_origin-1] = 0
#读取网络弧数据
df_link=pd.read_csv('road_link.csv')
df_link=df_link.iloc[:,:].values
for i in range(len(df_link)):
    g_link_list.append((df_link[i,1],df_link[i,2]))
    Distance[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,3]
    g_adjacent_arc_list[df_link[i,1]].append(df_link[i,2])
"""最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签"""
SE_LIST=[g_origin]
g_node_status[g_origin-1]=1
while len(SE_LIST):
    head=random.sample(SE_LIST,1)[0]#从待检查集合中随机抽取节点
    g_node_status[head-1]=0
    SE_LIST.remove(head)#将被抽取的节点从集合中移除
    adjacent_arc_list=g_adjacent_arc_list[head]#获取当前节点的邻接弧
    for tail in adjacent_arc_list:
```

```
        if node_label_cost[tail-1]>\
node_label_cost[head-1]+Distance[head-1,tail-1]:
            node_label_cost[tail-1]=\
node_label_cost[head-1]+Distance[head-1,tail-1]
            node_predecessor[tail-1]=head
            if g_node_status[tail-1]==0:
                SE_LIST.append(tail)
                g_node_status[tail-1]=1
"""依据前向节点生成最短路径"""
agent_id=1
o_zone_id=g_node_zone[g_origin]
for destination in g_node_list:
    if g_origin!=destination:
        d_zone_id=g_node_zone[destination]
        if node_label_cost[destination-1]==Max_label_cost:
            path=" "
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        else:
            to_node=copy.copy(destination)
            path="%s"%to_node
            while node_predecessor[to_node-1]!=g_origin:
                path="%s;"%node_predecessor[to_node-1]+path
                g=node_predecessor[to_node-1]
                to_node=g
            path="%s;"%g_origin+path
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        agent_id+=1
"""将求解结果导出到 csv 文件"""
#将数据转换为 DataFrame 格式方便导出 csv 文件
g_shortest_path=np.array(g_shortest_path)
col=['agent_id','o_zone_id','d_zone_id','node_sequence','distance']
file_data = pd.DataFrame(g_shortest_path,
index=range(len(g_shortest_path)),columns=col)
file_data.to_csv('agent.csv', index=False)
```

接下来给出 MATLAB 版本的 Modified Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-7　MATLAB 实现 Modified Label Correcting Algorithm

```
%% clear
clc
clear

%% 设置变量
g_node_list = []; %网络节点集合
g_link_list = []; %网络弧集合
g_origin = [];   %网络源节点
g_number_of_nodes = 0;%网络节点个数
node_predecessor = [];%前向节点集合
node_label_cost = [];%距离标签集合
Max_label_cost = inf; %初始距离标签
g_adjacent_arc_list={}; %节点邻接弧集合（从节点 i 发出的弧集合）
g_node_status=[]; %网络节点状态
SE_LIST=[]; %可扫描节点集合

```

```
%% 导入网络数据文件，构建基础网络并初始化相关变量
%读取网络节点数据
df_node = csvread('node.csv',1,0);
g_number_of_nodes = size(df_node,1);
g_node_list = df_node(:,1);
g_node_status=zeros(1,g_number_of_nodes);
for i = 1 : g_number_of_nodes
    if df_node(i,4)==1
        g_origin=df_node(i,1);
        o_zone_id = df_node(i,5);
    end
end
Distance = ones(g_number_of_nodes,g_number_of_nodes)*Max_label_cost;
%距离矩阵
node_predecessor = -1*ones(1,g_number_of_nodes);
node_label_cost = Max_label_cost*ones(1,g_number_of_nodes);
node_predecessor(1,g_origin) = 0;
node_label_cost(1,g_origin) = 0;
g_adjacent_arc_list = cell(1, g_number_of_nodes);
%读取网络弧数据
df_link = csvread('road_link.csv',1,0);
g_link_list = df_link(:,2:3);
for i = 1 : size(df_link, 1)
    Distance(df_link(i,2),df_link(i,3)) = df_link(i,4);
    g_adjacent_arc_list{df_link(i,2)} = [g_adjacent_arc_list{df_link(i,2)},
df_link(i,3)];
end

%% 最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签
SE_LIST=[g_origin];
g_node_status(g_origin)=1;
while ~isempty(SE_LIST)
    head=SE_LIST(randperm(numel(SE_LIST),1));
    %从待检查集合中随机抽取节点
    SE_LIST(SE_LIST==head)=[]; %将被抽取的节点从集合中移除
    g_node_status(head)=0;
    adjacent_arc_list = g_adjacent_arc_list(head); %获取当前节点的邻接弧
    adjacent_arc_list = cell2mat(adjacent_arc_list);
    for i = 1 : length(adjacent_arc_list)
        tail = adjacent_arc_list(i);
        if node_label_cost(tail)>…
node_label_cost(head)+Distance(head,tail)
            node_label_cost(tail)=…
node_label_cost(head)+Distance(head,tail);
            node_predecessor(tail)=head;
            if g_node_status(tail)==0
                SE_LIST = [SE_LIST,tail];
                g_node_status(tail) = 1;
            end
        end
    end
end

%% 依据前向节点生成最短路径
distance_column = [];
path_column = {};
o_zone_id_column = o_zone_id * ones(g_number_of_nodes-1, 1);
```

```
74  d_zone_id_column = [];
75  agent_id_column = [1:(g_number_of_nodes-1)]';
76  for i = 1: size(g_node_list, 1)
77      destination = g_node_list(i);
78      if g_origin ~= destination
79          d_zone_id_column = [d_zone_id_column; df_node(i,5)];
80          if node_label_cost(destination)==Max_label_cost
81              path="";
82              distance = 99999;
83              distance_column = [distance_column; 99999];
84          else
85              to_node=destination;
86              path=num2str(to_node);
87              while node_predecessor(to_node)~=g_origin
88                  path=strcat(';',path);
89                  path=strcat(num2str(node_predecessor(to_node)),path);
90                  g=node_predecessor(to_node);
91                  to_node=g;
92              end
93              path=strcat(';',path);
94              path=strcat(num2str(g_origin),path);
95              distance_column = [distance_column; node_label_cost(i)];
96          end
97          path_column=[path_column;path];
98      end
99  end
100
101 title = {'agent_id','o_zone_id','d_zone_id','node_sequence','distance'};
102 result_table=table(agent_id_column,o_zone_id_column,…
103 d_zone_id_column,path_column,distance_column,'VariableNames',title);
104 writetable(result_table, 'agent.csv',…
105 'Delimiter',',','QuoteStrings',true)
106
107
```

3.4 FIFO Label Correcting Algorithm

3.4.1 算法介绍

同样我们再次回顾一下 Modified Label Correcting Algorithm 的关键步骤：引入 SE_LIST 记录距离标签更新的节点编号，并在下一次迭代时检查 SE_LIST 内某一节点j发出的所有弧，即表 3-5 第 7 行与第 13 行。我们可以看出算法并没有给出从 SE_LIST 中选择节点以及向 SE_LIST 添加节点的具体规则，因此我们在相应代码实现时以随机的方式选取节点，并将新的节点添加到 SE_LIST 的尾部，即表 3-6 第 48-50 行与 58-60 行，表 3-7 第 49-52 行与 63-65 行。这种随机方式是否会对算法效率产生影响呢？前边我们已经知道，当最大网络弧长C非常大时 Modified Label Correcting Algorithm 的迭代次数为$\boldsymbol{O(2^n)}$。现在假设我们其将应用到一个病态的

数据集上（这类数据集往往含有非常大的C值），且每次迭代时从 SE_LIST 中选取节点或向 SE_LIST 中添加节点的顺序不合适时，算法总的迭代次数会随着网络节点数n成指数式增长。显然，为了保证算法效率的可靠性我们必须设计出较为合理的规则对 SE_LIST 中节点进行操作。这里介绍一种基于 FIFO（FirstInFirstOut，先进先出）的节点处理规则，我们称其为 FIFO Label Correcting Algorithm。

这里我们尝试对 A 中的弧进行排序以解决 Generic Label Correcting Algorithm 中弧扫描规则不明确问题（或者说 SE_LIST 中节点操作顺序问题）：将弧集合 A 中的所有弧以**某种特定的方式排序**，然后在每次迭代中逐个检查 A 中的弧，如果某条弧满足条件：$d(j) > d(i) + c_{ij}$，则更新相应的距离标签：$d(j) := d(i) + c_{ij}$，及节点的前向节点。当某步迭代后，A中所有弧都满足最优性条件时，结束算法。

接下来我们回顾一下 3.3.1 小节的内容，在引入 SE_LIST 时我们提到只有当节点j的距离标签更新时才需要在后续迭代时检查从节点j发出的所有弧是否满足最优性条件。所以上述尝试还需进一步改进。

我们将弧集合A中的弧按照它们的尾节点升排序，以便所有具有相同尾节点的弧都连续出现在集合A中。这样在扫描弧时，我们可以一次考虑一个节点发出的所有弧，比如节点i，扫描$A(i)$中的弧，并判断其是否满足最优性条件。假设在某次迭代遍历过程中，算法没有更新节点i的距离标签，那么在下一步迭代中，始终存在$d(j) \le d(i) + c_{ij}, \forall j \in A(i)$，因此没有必要再次检查$A(i)$中的弧。

根据以上分析，我们同样引入可扫描列表 SE_LIST，记录在一次迭代过程中距离标签发生更新的所有节点，并在下一次迭代中只考虑该列表中节点发出的所有弧。具体细节为：从 SE_LIST 一端（这里以左端为例）取出一个节点i，检查$A(i)$中的所有弧是否满足最优性条件；从 SE_LIST 另一端（右端）添加新的节点以便后续迭代检查判断。我们称为 FIFO 规则，即先进先出。采用这类规则的算法称为 FIFO Label Correcting Algorithm，其算法步骤如下。

表 3-8　FIFO Label Correcting Algorithm

Algorithm 4 FIFO Label Correcting Algorithm
1:　begin
2:　　$d(s) := 0$ and $\mathrm{pred}(s) := 0$;
3:　　$d(j) := \infty$ for each node $j \in N - \{s\}$;
4:　　SE_LIST: $= \{s\}$;
5:　　while SE_LIST $\neq \emptyset$ do
6:　　begin

```
7:          remove the first element on the left from SE_LIST;
8:          for each arc (i,j) ∈ A(i) do
9:          if d(j) > d(i) + c_ij then
10:         begin
11:             d(j) := d(i) + c_ij;
12:             pred(j) := i;
13:             if j ∉ SE_LIST then add node j to the right end of SE_LIST;
14:         end;
15:     end;
16: end;
```

学习到这里你可能会发现，Modified Label Correcting Algorithm 和 FIFO Label Correcting Algorithm 的步骤几乎一样。事实也是如此，如果 Modified Label Correcting Algorithm 采取“从 SE_LIST 的头部选择节点，并将新的节点添加到 SE_LIST 的尾部”的策略，这和我们本节提出的 FIFO Label Correcting Algorithm 是相同的，你也可以认为 FIFO Label Correcting Algorithm 是 Modified Label Correcting Algorithm 的一种特例。

复杂度分析

FIFO Label Correcting Algorithm 以 FIFO 方式处理对 SE_LIST 进行操作，有效避免了最大弧长C值对算法效率产生的影响。我们已经知道 Modified Label Correcting Algorithm 总的迭代次数为$O(nmC)$，所以 FIFO Label Correcting Algorithm 总的迭代次数为$O(nm)$。

3.4.2 算法实现

首先给出 Python 版本的 FIFO Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-9　Python 实现 FIFO Label Correcting Algorithm

```python
"""导入相关基础包，定义全局变量"""
import pandas as pd
import numpy as np
import copy
g_node_list=[] #网络节点集合
g_node_zone={} #网络节点类别集合
g_link_list=[] #网络弧集合
g_adjacent_arc_list={}#节点邻接弧集合（从节点 i 发出的弧集合）
g_shortest_path=[]#最短路径集合
g_node_status=[]#网络节点状态
g_number_of_nodes=0#网络节点个数
g_origin=None   #网络源节点
node_predecessor=[]#前向节点集合
node_label_cost=[]#距离标签集合
SE_LIST=[]#可扫描节点集合
Max_label_cost=99999#初始距离标签
"""导入网络数据文件，构建基础网络并初始化相关变量"""
#读取网络节点数据
```

```
df_node=pd.read_csv('node.csv')
df_node=df_node.iloc[:,:].values
for i in range(len(df_node)):
    g_node_list.append(df_node[i,0])
    g_node_zone[df_node[i, 0]] = df_node[i, -1]
    g_number_of_nodes+=1
    g_adjacent_arc_list[df_node[i,0]]=[]
    if df_node[i, 3] == 1:
        g_origin = df_node[i, 0]
g_node_status=[0 for i in range(g_number_of_nodes)] #初始化网络节点状态
Distance=np.ones((g_number_of_nodes,g_number_of_nodes))\
*Max_label_cost #距离矩阵
node_predecessor=[-1]*g_number_of_nodes
node_label_cost=[Max_label_cost]*g_number_of_nodes
node_predecessor[g_origin-1]=0
node_label_cost[g_origin-1] = 0
#读取网络弧数据
df_link=pd.read_csv('road_link.csv')
df_link=df_link.iloc[:,:].values
for i in range(len(df_link)):
    g_link_list.append((df_link[i,1],df_link[i,2]))
    Distance[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,3]
    g_adjacent_arc_list[df_link[i,1]].append(df_link[i,2])
SE_LIST=[g_origin]
g_node_status[g_origin-1]=1
"""最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签"""
while len(SE_LIST):
    head=SE_LIST[0]#从可扫描列表中取出第一个元素
    SE_LIST.pop(0)#从可扫描列表中删除第一个元素
    g_node_status[head-1]=0
    adjacent_arc_list=g_adjacent_arc_list[head]#获取当前节点的邻接弧集合
    for tail in adjacent_arc_list:
        if node_label_cost[tail-1]>\
node_label_cost[head-1]+Distance[head-1,tail-1]:
            node_label_cost[tail-1]=\
node_label_cost[head-1]+Distance[head-1,tail-1]
            node_predecessor[tail-1]=head
            if g_node_status[tail-1]==0:
                SE_LIST.append(tail)#将新节点添加到可扫描列表尾部,append 方法实现从列表
尾部添加元素
                g_node_status[tail-1]=1
"""依据前向节点生成最短路径"""
agent_id=1
o_zone_id=g_node_zone[g_origin]
for destination in g_node_list:
    if g_origin!=destination:
        d_zone_id=g_node_zone[destination]
        if node_label_cost[destination-1]==Max_label_cost:
            path=" "
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        else:
            to_node=copy.copy(destination)
            path= "%s" % to_node
            while node_predecessor[to_node-1]!=g_origin:
                path="%s;"% node_predecessor[to_node - 1] + path
                g=node_predecessor[to_node-1]
```

```
                to_node=g
            path="%s;"%g_origin+path
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        agent_id+=1
"""将求解结果导出到 csv 文件"""
#将数据转换为 DataFrame 格式方便导出 csv 文件
g_shortest_path=np.array(g_shortest_path)
col=['agent_id','o_zone_id','d_zone_id','node_sequence','distance']
file_data = pd.DataFrame(g_shortest_path,
index=range(len(g_shortest_path)),columns=col)
file_data.to_csv('agent.csv',index=False)
```

接下来给出 MATLAB 版本的 FIFO Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）。

表 3-10　MATLAB 实现 FIFO Label Correcting Algorithm

```
%% clear
clc
clear

%% 设置变量
g_node_list = []; %网络节点集合
g_link_list = []; %网络弧集合
g_origin = [];   %网络源节点
g_number_of_nodes = 0;%网络节点个数
node_predecessor = [];%前向节点集合
node_label_cost = [];%距离标签集合
Max_label_cost = inf; %初始距离标签
g_adjacent_arc_list={}; %节点邻接弧集合（从节点 i 发出的弧集合）
g_node_status=[]; %网络节点状态
SE_LIST=[]; %可扫描节点集合

%% 导入网络数据文件，构建基础网络并初始化相关变量
%读取网络节点数据
df_node = csvread('node.csv',1,0);
g_number_of_nodes = size(df_node,1);
g_node_list = df_node(:,1);
g_node_status = zeros(1,g_number_of_nodes);
for i = 1 : g_number_of_nodes
    if df_node(i,4)==1
        g_origin=df_node(i,1);
        o_zone_id = df_node(i,5);
    end
end
Distance = ones(g_number_of_nodes,g_number_of_nodes)*Max_label_cost;
%距离矩阵
node_predecessor = -1*ones(1,g_number_of_nodes);
node_label_cost = Max_label_cost*ones(1,g_number_of_nodes);
node_predecessor(1,g_origin) = 0;
node_label_cost(1,g_origin) = 0;
g_adjacent_arc_list = cell(1, g_number_of_nodes);
%读取网络弧数据
df_link = csvread('road_link.csv',1,0);
g_link_list = df_link(:,2:3);
for i = 1 : size(df_link, 1)
    Distance(df_link(i,2),df_link(i,3)) = df_link(i,4);
```

```
41      g_adjacent_arc_list{df_link(i,2)} = [g_adjacent_arc_list{df_link(i,2)},
42  df_link(i,3)];
43  end
44
45  %% 最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签
46  SE_LIST=[g_origin];
47  g_node_status(g_origin)=1;
48  while ~isempty(SE_LIST)
49      head=SE_LIST(1); %从可扫描列表中取出第一个元素
50      SE_LIST(1)=[]; %从可扫描列表中删除第一个元素
51      g_node_status(head)=0;
52      adjacent_arc_list = g_adjacent_arc_list(head);
53      %获取当前节点的邻接弧
54      adjacent_arc_list = cell2mat(adjacent_arc_list);
55      for i = 1 : length(adjacent_arc_list)
56          tail = adjacent_arc_list(i);
57          if node_label_cost(tail)>…
58  node_label_cost(head)+Distance(head,tail)
59              node_label_cost(tail)=…
60  node_label_cost(head)+Distance(head,tail);
61              node_predecessor(tail)=head;
62              if g_node_status(tail)==0
63                  SE_LIST = [SE_LIST,tail];
64                  g_node_status(tail) = 1;
65              end
66          end
67      end
68  end
69
70  %% 依据前向节点生成最短路径
71  distance_column = [];
72  path_column = {};
73  o_zone_id_column = o_zone_id * ones(g_number_of_nodes-1, 1);
74  d_zone_id_column = [];
75  agent_id_column = [1:(g_number_of_nodes-1)]';
76  for i = 1: size(g_node_list, 1)
77      destination = g_node_list(i);
78      if g_origin ~= destination
79          d_zone_id_column = [d_zone_id_column; df_node(i,5)];
80          if node_label_cost(destination)==Max_label_cost
81              path="";
82              distance = 99999;
83              distance_column = [distance_column; 99999];
84          else
85              to_node=destination;
86              path=num2str(to_node);
87              while node_predecessor(to_node)~=g_origin
88                  path=strcat(';',path);
89                  path=strcat(num2str(node_predecessor(to_node)),path);
90                  g=node_predecessor(to_node);
91                  to_node=g;
92              end
93              path=strcat(';',path);
94              path=strcat(num2str(g_origin),path);
95              distance_column = [distance_column; node_label_cost(i)];
96          end
97          path_column=[path_column;path];
```

```
98        end
99    end
100
101   title = {'agent_id','o_zone_id','d_zone_id','node_sequence','distance'};
102   result_table=table(agent_id_column,o_zone_id_column,…
103   d_zone_id_column,path_column,distance_column,'VariableNames',title);
104   writetable(result_table, 'agent.csv',…
105   'Delimiter',',','QuoteStrings',true)
```

3.5 Deque Label Correcting Algorithm

3.5.1 算法介绍

同样，我们先回顾一下已学的三个 Label Correcting Algorithms 的特点：Generic Label Correcting Algorithm 提出了一种新的算法框架，即 Label Correcting，它允许我们以更有效、灵活的方式求解最短路问题，但它的灵活性也正是它缺点，由于规则的缺失导致算法效率不能保证；因此 Modified Label Correcting Algorithm 引入了可扫描列表 SE_LIST 来明确应该扫描哪些弧，算法虽有改进但却带来了新的问题，以何种规则处理 SE_LIST 中节点？最后 FIFO Label Correcting Algorithm 提出了 FIFO 规则来明确 SE_LIST 中节点的操作规则，即表 3-8 第 7 行和第 13 行，在相应代码实现中体现在表 3-9 第 46-48 行和第 56-59 行，表 3-10 第 49-51 行和第 62-64 行。这里请思考下除了 FIFO 是最好的么？还有其他规则可以选择么？

接下来我们介绍另一种规则：将 SE_LIST 作为 Deque 处理。如图 3-4（a）所示，Deuqe 以 Left-front 标记队列的左端头部，Left-rear 标记队列的左端尾部，Right-front 标记队列的右端头部，Right-rear 标记队列的右端尾部，当需要进行删除或者插入操作时可以选择在两端进行。

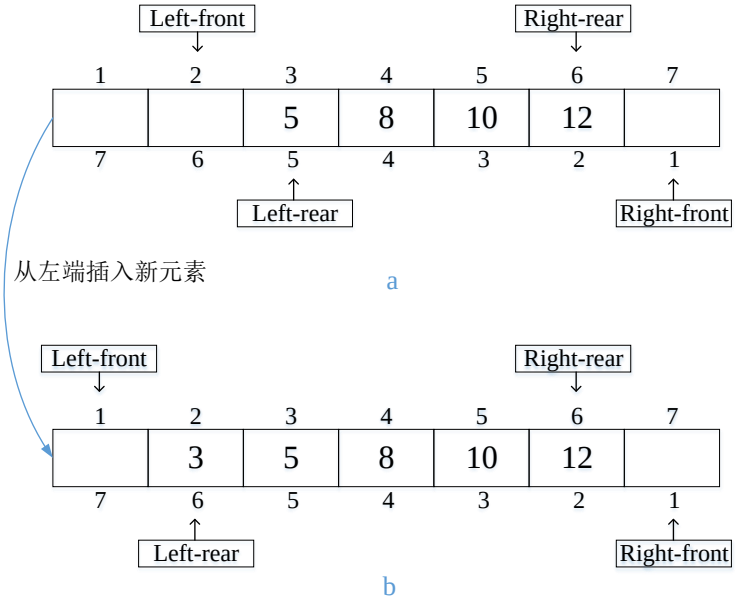


图 3-4　Deque 数据结构

Deque Label Correcting Algorithm 将 SE_LIST 看作一个 Deque 数据结构，并且规定遍历节点时从 SE_LIST 的一端头部开始（这里我们以左端头部为例），添加节点时，如果该节点已在 SE_LIST 出现过中，则将其添加到 SE_LIST 的左端头部，否则将其添加到 SE_LIST 右端尾部尾端。接下来进一步分析采用 Deque 数据结构的原因。

假设在前k次迭代中，节点j已在 SE_LIST 中出现过，且某些节点$i_1, i_2, \dots i_k$可能将其作为前向节点。第k次迭代时，$i_1, i_2, \dots i_k$已在 SE_LIST 中，如果节点j的距离标签再次更新后，节点j将被加入 SE_LIST 中：如果将节点j添加到 SE_LIST 的右端尾部，则随后迭代时则会依次检查$i_1, i_2, \dots i_k$等节点，并更新其他节点的距离标签，当算法检查到节点j时，又会更新节点$i_1, i_2, \dots i_k$的距离标签，因此建立在$i_1, i_2, \dots i_k$旧标签基础之上时其他距离标签将会失效，因此降低了算法效率。而对在k次迭代中从未在 SE_LIST 中出现的节点j来说，节点j将 SE_LIST 已有的节点作为其可能的前向节点，因此应加到 SE_LIST 右端尾部。因此这种处理规则是合适的。Deque Label Correcting Algorithm 步骤如下。

表 3-11　Deque Label Correcting Algorithm

Algorithm 5 Deque Label Correcting Algorithm

```
begin
    d(s) := 0 and pred(s) := 0;
    d(j) := ∞ for each node j ∈ N − {s};
    SE_LIST := {s};
    while SE_LIST ≠ ∅ do
    begin
        remove the first element on the left from SE_LIST;
        for each arc (i, j) ∈ A(i) do
        if d(j) > d(i) + c_ij then
        begin
            d(j) := d(i) + c_ij;
            pred(j) := i;
            if node j has not been in SE_LIST earlier then add node j to the right end of SE_LIST;
            else add node j to the left end of SE_LIST;
        end;
    end;
end;
```

关于 Deque Label Correcting Algorithm 也可以参考文献[5]进一步学习。

复杂度分析

大量研究表明，Deque 方法检查的节点数量比大多数其他标签校正算法要少。虽然该算法时间复杂度不是多项式，但在实际应用中却很有效。而且，该算法被证明是解决稀疏和交通网络最短路问题最快的算法之一。

3.5.2 算法实现

首先给出 Python 版本的 Deque Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）

表 3-12　Python 实现 Deque Label Correcting Algorithm

```
"""导入相关基础包，定义全局变量"""
import pandas as pd
import numpy as np
import copy
g_node_list=[] #网络节点集合
g_node_zone={} #网络节点类别集合
g_link_list=[] #网络弧集合
g_adjacent_arc_list={}#节点邻接弧集合（从节点 i 发出的弧集合）
g_node_status=[]#网络节点状态
g_shortest_path=[]#最短路径集合
g_origin=None #网络源节点
g_number_of_nodes=0#网络节点个数
node_predecessor=[]#前向节点集合
node_label_cost=[]#距离标签集合
Max_label_cost=99999#初始距离标签
"""导入网络数据文件，构建基础网络并初始化相关变量"""
#读取网络节点数据
df_node=pd.read_csv('node.csv')
df_node=df_node.iloc[:,:].values
for i in range(len(df_node)):
    g_node_list.append(df_node[i,0])
    g_node_zone[df_node[i, 0]] = df_node[i, -1]
    g_number_of_nodes+=1
    g_adjacent_arc_list[df_node[i,0]]=[]
    if df_node[i, 3] == 1:
        g_origin = df_node[i, 0]
g_node_status=[0 for i in range(g_number_of_nodes)] #初始化网络节点状态
Distance=np.ones((g_number_of_nodes,g_number_of_nodes))\
*Max_label_cost #距离矩阵
node_predecessor=[-1]*g_number_of_nodes
node_label_cost=[Max_label_cost]*g_number_of_nodes
node_predecessor[g_origin-1]=0
node_label_cost[g_origin-1] = 0
#读取网络弧数据
df_link=pd.read_csv('road_link.csv')
df_link=df_link.iloc[:,:].values
for i in range(len(df_link)):
    g_link_list.append((df_link[i,1],df_link[i,2]))
    Distance[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,3]
    g_adjacent_arc_list[df_link[i,1]].append(df_link[i,2])
SE_LIST=[g_origin]
```

```
g_node_status[g_origin-1]=1
"""最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签"""
while len(SE_LIST):
    head=SE_LIST[0]#从可扫描列表左端取出第一个元素
    SE_LIST.pop(0)#从可扫描列表中删除左端第一个元素
    g_node_status[head-1]=2
    adjacent_arc_list=g_adjacent_arc_list[head]#获取当前节点的邻接弧集合
    for tail in adjacent_arc_list:
        if node_label_cost[tail-1]>\
node_label_cost[head-1]+Distance[head-1,tail-1]:
            node_label_cost[tail-1]=\
node_label_cost[head-1]+Distance[head-1,tail-1]
            node_predecessor[tail-1]=head
            if g_node_status[tail-1]==0:
                SE_LIST.append(tail) #如果当前节点不曾出现在 SE_LIST 则将其假如右端尾部
                g_node_status[tail-1]=1
            elif g_node_status[tail-1]==2:
                SE_LIST.insert(0,tail) #如果当前节点曾出现在 SE_LIST 则将其假如左端头部
                g_node_status[tail-1]=1
"""依据前向节点生成最短路径"""
agent_id=1
o_zone_id=g_node_zone[g_origin]
for destination in g_node_list:
    if g_origin!=destination:
        d_zone_id=g_node_zone[destination]
        if node_label_cost[destination-1]==Max_label_cost:
            path= " "
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        else:
            to_node=copy.copy(destination)
            path="%s" % to_node
            while node_predecessor[to_node-1]!=g_origin:
                path = "%s;" % node_predecessor[to_node - 1] + path
                g=node_predecessor[to_node-1]
                to_node=g
            path = "%s;" % g_origin + path
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,path,
node_label_cost[destination - 1]])
        agent_id+=1
"""将求解结果导出到 csv 文件"""
#将数据转换为 DataFrame 格式方便导出 csv 文件
g_shortest_path=np.array(g_shortest_path)
col=['agent_id','o_zone_id','d_zone_id','node_sequence','distance']
file_data = pd.DataFrame(g_shortest_path,
index=range(len(g_shortest_path)),columns=col)
file_data.to_csv('agent.csv', index=False)
```

接下来给出 MATLAB 版本的 Deque Label Correcting Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）。

表 3-13　MATLAB 实现 Deque Label Correcting Algorithm

```
%% clear
clc
clear

%% 设置变量
```

```
6    g_node_list = []; %网络节点集合
7    g_link_list = []; %网络弧集合
8    g_origin = [];   %网络源节点
9    g_number_of_nodes = 0;%网络节点个数
10   node_predecessor = [];%前向节点集合
11   node_label_cost = [];%距离标签集合
12   Max_label_cost = inf; %初始距离标签
13   g_adjacent_arc_list={}; %节点邻接弧集合（从节点 i 发出的弧集合）
14   g_node_status=[]; %网络节点状态
15   SE_LIST=[]; %可扫描节点集合
16
17   %% 导入网络数据文件，构建基础网络并初始化相关变量
18   %读取网络节点数据
19   df_node = csvread('node.csv',1,0);
20   g_number_of_nodes = size(df_node,1);
21   g_node_list = df_node(:,1);
22   g_node_status = zeros(1,g_number_of_nodes);
23   for i = 1 : g_number_of_nodes
24       if df_node(i,4)==1
25           g_origin=df_node(i,1);
26           o_zone_id = df_node(i,5);
27       end
28   end
29   Distance = ones(g_number_of_nodes,g_number_of_nodes)*Max_label_cost;
30   %距离矩阵
31   node_predecessor = -1*ones(1,g_number_of_nodes);
32   node_label_cost = Max_label_cost*ones(1,g_number_of_nodes);
33   node_predecessor(1,g_origin) = 0;
34   node_label_cost(1,g_origin) = 0;
35   g_adjacent_arc_list = cell(1, g_number_of_nodes);
36   %读取网络弧数据
37   df_link = csvread('road_link.csv',1,0);
38   g_link_list = df_link(:,2:3);
39   for i = 1 : size(df_link, 1)
40       Distance(df_link(i,2),df_link(i,3)) = df_link(i,4);
41       g_adjacent_arc_list{df_link(i,2)} = [g_adjacent_arc_list{df_link(i,2)},
42   df_link(i,3)];
43   end
44
45   %% 最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签
46   SE_LIST=[g_origin];
47   g_node_status(g_origin)=1;
48   while ~isempty(SE_LIST)
49       head=SE_LIST(1); %从可扫描列表中取出第一个元素
50       SE_LIST(1)=[]; %从可扫描列表中删除第一个元素
51       g_node_status(head) = 2;
52   adjacent_arc_list = g_adjacent_arc_list(head);
53   %获取当前节点的邻接弧
54       adjacent_arc_list = cell2mat(adjacent_arc_list);
55       for i = 1 : length(adjacent_arc_list)
56           tail = adjacent_arc_list(i);
57           if node_label_cost(tail)>…
58   node_label_cost(head)+Distance(head,tail)
59               node_label_cost(tail)=…
60   node_label_cost(head)+Distance(head,tail);
61               node_predecessor(tail)=head;
62               if g_node_status(tail)==0
```

```
                SE_LIST = [SE_LIST, tail];
                g_node_status(tail) = 1;
            elseif g_node_status(tail) == 2
                SE_LIST = [tail, SE_LIST];
                g_node_status(tail) = 1;
            end
        end
    end
end

%% 依据前向节点生成最短路径
destination_column = [];
distance_column = [];
path_column = {};
o_zone_id_column = o_zone_id * ones(g_number_of_nodes-1, 1);
d_zone_id_column = [];
agent_id_column = [1:(g_number_of_nodes-1)]';
for i = 1: size(g_node_list, 1)
    destination = g_node_list(i);
    if g_origin ~= destination
        destination_column = [destination_column; destination];
        d_zone_id_column = [d_zone_id_column; df_node(i,5)];
        if node_label_cost(destination)==Max_label_cost
            path="";
            distance = 99999;
            distance_column = [distance_column; 99999];
        else
            to_node=destination;
            path=num2str(to_node);
            while node_predecessor(to_node)~=g_origin
                path=strcat(';',path);
                path=strcat(num2str(node_predecessor(to_node)),path);
                g=node_predecessor(to_node);
                to_node=g;
            end
            path=strcat(';',path);
            path=strcat(num2str(g_origin),path);
            distance_column = [distance_column; node_label_cost(i)];
        end
        path_column=[path_column;path];
    end
end

title = {'agent_id','o_zone_id','d_zone_id','node_sequence','distance'};
result_table=table(agent_id_column,o_zone_id_column,…
d_zone_id_column,path_column,distance_column,'VariableNames',title);
writetable(result_table, 'agent.csv',…
'Delimiter',',','QuoteStrings',true)

```

3.6 多源最短路径算法

在前边介绍中我们主要考虑简单网络的单源最短路径问题，本小节我们将研究简单网络的多源最短路径问题，也就是说我们要找的是简单网络中任意两个节点之间的最短路径。为了实

现这个目的，需要该网络中至少包含一条从任意节点i到任意节点j的有向路径，即网络具有强连接性(网络中任意两节点都是可达的)。如果网络不满足这个条件，我们可以通过增加一条弧长为无穷大的虚拟弧来解决。

这里我们将介绍两种求解简单网络多源最短路径的 Label Correcting Algorithms，第一种算法是基于之前所介绍的单源 Label Correcting Algorithms 实现的，简单来说就是依次将网络中的每个节点作为源节点，求解它到其他节点的最短路径，我们称其为 Repeated Shortest Path Algorithm；第二种算法是基于多源最短路径最优性条件实现的简，我们称其为 All-pairs Label Correcting Algorithm，这种算法其实是 Label Correcting Algorithms 的普适化，可以将单源 Label Correcting Algorithms 看作 All-pairs Label Correcting Algorithm 的特例。第一种方法较为简单这里不再赘述。

我们将在 3.6.1 小节给出多源最短路径最优性判别条件；以多源最短路径最优性条件为基础，3.6.2 小节介绍了 All-pairs Generic Label Correcting Algorithm 的一般步骤，并对其可行性进行说明；3.6.3 小节对 All-pairs Generic Label Correcting Algorithm 中的经典算法——Floyd-Warshall Algorithm 做进一步介绍。

3.6.1 多源最短路最优性条件

在 All-pairs Label Correcting Algorithm 中我们记$d[i,j]$为任意节点对$[i,j]$的距离标签，代表从节点i到节点j的一条路径的长度的上界。算法不断更新距离标签矩阵，直到所有的距离标签都为最短路径长度。其所依据的“全对最短路径最优性条件”如下：

最优性定理 2：对于任意一对节点$[i,j] \in N \times N$，记$d[i,j]$为节点i到节点j的有向路径长度，当且仅当满足以下条件时，$d[i,j]$为节点i到节点j的最短路径长度：

$$\mathrm{d[i,j]} \leq \mathrm{d[i,k]} + \mathrm{d[k,j]}, \forall i,j,k \in N \quad (6)$$

证明过程可参考定理 1，这里不再赘述。

3.6.2 All-Pairs Generic Label Correcting Algorithm

All-Pairs Generic Label Correcting Algorithm 求解思想与单源 Label Correcting Algorithms 类似：从一些距离标签开始，不断对其更新直到其满足最优性条件。以下给出了 All-pairs Generic Label-Correcting Algorithm 的一般步骤。

表 3-14　All-pairs Generic Label-Correcting Algorithm

Algorithm 6 All-pairs Generic Label-Correcting Algorithm	
1:	**begin**
2:	**set** $d[i,j]:=\infty$, **for all** $(i,j)\in N\times N$;
3:	**set** $d[i,i]:=0$ **for all** $i\in N$;
4:	**for each** $(i,j)\in A$ **do** $d[i,j]:=c_{ij}$;
5:	**while the network contains three nodes** i,j,**and** k **satisfying** $d[i,j]>d[i,k]+d[k,j]$ **do**
6:	$d[i,j]:=d[i,k]+d[k,j]$;
7:	**end**

接下来我们对该算法的收敛性和正确性进行证明。首先来看正确性：算法在每次迭代时，只要节点对$[i,j]$的距离标签为有限值，就代表了一条从节点i到节点j的长度为$d[i,j]$的有向路径。当算法满足最优性条件结束迭代时，节点i到节点j的有向路径可分解为一些有向环和一条路径。由于我们所研究的网络中不含有环，所以路径的长度就等于有向路径的长度，又因为这条路径满足最优性条件，因此$d[i,j]$就是节点对$[i,j]$的最短路径长度。接下来分析收敛性：由于所有的弧长均为整数，且最大的弧长为 C，所以任意节点对的距离标签取值范围是$[-nC,nC]$。在每次迭代中算法不断减小距离标签的值，因此算法在$\boldsymbol{O(n^3C)}$内结束迭代。由此证明该算法的收敛性和正确性。

请注意表 3-14 第 5 行，在检查节点距离标签是否满足最优性条件 2 时并没有给出具体的检查规则。与 generic label-correcting algorithm 类似，不明确的检查规则会导致算法的求解效率无法得到保障。这里不再给出 All-Pairs Generic Label Correcting Algorithm 的相应代码实现。

3.6.3 Floyd-Warshall Algorithm

3.6.3.1 算法介绍

为了解决 All-Pairs Generic Label Correcting Algorithm 节点检查规则不明确带来的算法效率问题，这里给出 Floyd-Warshall 的改进方法。Floyd-Warshall 基于动态规划技术对算法进行了改进，这就是著名的 Floyd-Warshall Algorithm。其核心是在第 k 次迭代时只考虑将节点$1,2,\dots k-1$作为节点对$[i,j]$的中间节点，具体做法如下：令$d[i,j]^k$为第 k 次迭代后节点对$[i,j]$的距离标签，且在第 k 次迭代时只考虑将节点$1,2,\dots,k-1$作为其内部节点。算法开始时，计算任意节点对$[i,j]$的距离标签$d[i,j]^1$，然后依据最优性条件 2 更新任意节点对之间的距离标签$d[i,j]^2$，重复上述过程。最终$d[i,j]^{n+1}$即为节点对$[i,j]$的最短路径长度。第 k 次迭代时$d[i,j]^k$更新规则如下：

$$d[i,j]^k=\min(d[i,j]^{k-1},d[i,k]^{k-1}+d[k,j]^{k-1}) \tag{7}$$

Floyd-Warshall Algorithm 关键步骤伪代码如下

表 3-15　Floyd-Warshall Algorithm

Algorithm 7 Floyd-Warshall Algorithm

```
1:   begin
2:       for all node pairs [i,j] ∈ N × N do
3:           d[i,j]: =  ∞,and pred[i,j]: = 0;
4:       for all node i ∈ N do d[i,i]: = 0;
5:       for each (i,j) ∈ A do d[i,j]: = c_ij, and pred[i,j]: = i;
6:       for each k := 1 to n do
7:           for each [i,j] ∈ N × N do
8:               if d[i,j] > d[i,k] + d[k,j] then
9:               begin
10:                  d[i,j]: = d[i,k] + d[k,j];
11:                  pred[i,j]: = pred[k,j];
12:              end;
13:  end;
```

与单源最短路径算法不同，多源最短路径算法以pred$[i,j]$记录节点对$[i,j]$的前向节点。pred$[i,j]$记录了当前路径中到达终点j前的最后一个节点，当距离标签$d[i,j]$更新时pred$[i,j]$也随之更新。当算法结束时，我们可以通过pred[i,j]来生成从节点i到节点j的最短路径P：假设生成从节点i到节点j的最短路径，首先令g=pred[i,j]，把节点g加入P；然后令h=pred[i,g],g=h，把节点g加入P。重复上述过程直到pred[i,g]=i。

复杂度分析

Floyd-Warshall Algorithm 给出了明确的检查节点的顺序，使得算法迭代次数控制在$O(n^3)$，

3.6.3.2 算法实现

首先给出 Python 版本的 Floyd-Warshall Algorithm 实现（求解附录 2 中任意节点间的最短路径）

表 3-16　Python 实现 Floyd-Warshall Algorithm

```
"""导入相关基础包，定义全局变量"""
import pandas as pd
import numpy as np
g_node_list=[] #网络节点集合
g_node_zone={} #网络节点类别集合
g_link_list=[] #网络弧集合
g_shortest_path=[]#最短路径集合
g_number_of_nodes=0#网络节点个数
node_predecessor=[]#前向节点集合
node_label_cost=[]#距离标签集合
Max_label_cost=99999#初始距离标签
"""导入网络数据文件，构建基础网络并初始化相关变量"""
#读取网络节点数据
df_node=pd.read_csv('node.csv')
df_node=df_node.iloc[:,:].values
for i in range(len(df_node)):
```

```
    g_node_list.append(df_node[i,0])
    g_node_zone[df_node[i, 0]] = df_node[i, -1]
    g_number_of_nodes+=1
node_label_cost=np.ones((g_number_of_nodes,g_number_of_nodes))\
*Max_label_cost
node_predecessor=np.zeros((g_number_of_nodes,g_number_of_nodes))
for i in range(g_number_of_nodes):
    for j in range(g_number_of_nodes):
        if i==j:
            node_label_cost[i,j]=0
#读取网络弧数据
df_link=pd.read_csv('road_link.csv')
df_link=df_link.iloc[:,:].values
for i in range(len(df_link)):
    g_link_list.append((df_link[i,1],df_link[i,2]))
    node_label_cost[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,3]
    node_predecessor[df_link[i,1]-1,df_link[i,2]-1]=df_link[i,1]
"""最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签"""
for k in g_node_list:
    for arc_head in g_node_list:
        for arc_tail in g_node_list:
            if node_label_cost[arc_head-1,arc_tail-1]>\
node_label_cost[arc_head-1,k-1]+node_label_cost[k-1,arc_tail-1]:
                    node_label_cost[arc_head-1,arc_tail-1]=\
node_label_cost[arc_head-1,k-1]+node_label_cost[k-1,arc_tail-1]
                    node_predecessor[arc_head-1,arc_tail-1]=\
node_predecessor[k-1,arc_tail-1]
"""依据前向节点生成最短路径"""
agent_id=1
for from_node in g_node_list:
    o_zone_id=g_node_zone[from_node]
    for to_node in g_node_list:
        if from_node!=to_node:
            d_zone_id=g_node_zone[to_node]
            if node_label_cost[from_node-1,to_node-1]==Max_label_cost:
                path =" "
            else:
                path="%s" % to_node
                prior_point=\
int(node_predecessor[from_node-1,to_node-1])
                while prior_point!=from_node:
                    path= "%s;" %prior_point+path
                    prior_point=\
int(node_predecessor[from_node-1,prior_point-1])
                path="%s;" %from_node + path
            g_shortest_path.append([agent_id, o_zone_id,d_zone_id,
path,node_label_cost[from_node-1,to_node-1]])
"""将求解结果导出到 csv 文件"""
#将数据转换为 DataFrame 格式方便导出 csv 文件
g_shortest_path=np.array(g_shortest_path)
col=['agent_id','o_zone_id','d_zone_id','node_sequence','distance']
file_data = pd.DataFrame(g_shortest_path,
index=range(len(g_shortest_path)),columns=col)
file_data.to_csv('agent.csv', index=False)
```

接下来给出 MATLAB 版本的 Floyd-Warshall Algorithm 实现（求解附录 2 中源节点 1 到其他节点的最短路径）。

表 3-17　MATLAB 实现 Floyd-Warshall Algorithm

```
%% clear
clc
clear

%% 设置变量
g_node_list=[]; %网络节点集合
g_link_list=[]; %网络弧集合
g_number_of_nodes=0; %网络节点个数
node_predecessor=[]; %前向节点集合
node_label_cost=[]; %距离标签集合
Max_label_cost=inf;%初始距离标签

%% 导入网络数据文件，构建基础网络并初始化相关变量
%读取网络节点数据
df_node = csvread('node.csv',1,0);
g_number_of_nodes = size(df_node,1);
g_node_list = df_node(:,1);
node_label_cost=ones(g_number_of_nodes,g_number_of_nodes)*…
Max_label_cost;
node_predecessor=zeros(g_number_of_nodes,g_number_of_nodes);
for i = 1 : g_number_of_nodes
    for j = 1 : g_number_of_nodes
        if i==j
            node_label_cost(i,j)=0;
        end
    end
end

%读取网络弧数据
df_link = csvread('road_link.csv',1,0);
g_link_list = df_link(:,2:3);
for i = 1 : size(df_link,1)
    Distance(df_link(i,2),df_link(i,3)) = df_link(i,4);
    node_label_cost(df_link(i,2),df_link(i,3))=df_link(i,4);
    node_predecessor(df_link(i,2),df_link(i,3))=df_link(i,2);
end

%% 最短路径求解：扫描网络弧，依据检查最优性条件更新距离标签
for k = 1 : g_number_of_nodes
    for arc_head =1:g_number_of_nodes
        for arc_tail =1:g_number_of_nodes
            if node_label_cost(arc_head,arc_tail)>…
node_label_cost(arc_head,k)+node_label_cost(k,arc_tail)
                    node_label_cost(arc_head,arc_tail)=…
node_label_cost(arc_head,k)+node_label_cost(k,arc_tail);
                    node_predecessor(arc_head,arc_tail)=…
node_predecessor(k,arc_tail);
            end
        end
    end
end

%% 依据前向节点生成最短路径
agent_id_column = [1:((g_number_of_nodes-1)*g_number_of_nodes)]';
o_zone_id_column = [];
d_zone_id_column = [];
```

```
path_column = {};
distance_column = [];
for from_node = 1 : g_number_of_nodes
    for to_node = 1 : g_number_of_nodes
        if from_node~=to_node
            if node_label_cost(from_node,to_node)==Max_label_cost
                path = "";
                distance = 99999;
                distance_column = [distance_column; 99999];
            else
                path=num2str(to_node);
                prior_point=node_predecessor(from_node,to_node);
                while prior_point~=from_node
                    path=strcat(';',path);
                    path=strcat(num2str(prior_point),path);
                    prior_point=node_predecessor(from_node,prior_point);
                end
                path=strcat(';',path);
                path=strcat(num2str(from_node),path);
                distance_column = [distance_column;…
 node_label_cost(from_node,to_node)];
            end
            path_column=[path_column;path];
            o_zone_id_column=[o_zone_id_column;df_node(from_node,5)];
            d_zone_id_column=[d_zone_id_column;df_node(to_node,5)];
        end
    end
end

title = {'agent_id','o_zone_id','d_zone_id','node_sequence','distance'};
result_table=table(agent_id_column,o_zone_id_column,…
d_zone_id_column,path_column,distance_column,'VariableNames',title);
writetable(result_table, 'agent.csv',…
'Delimiter',',','QuoteStrings',true)
```

3.7 总结

在本章中，我们首先介绍了求解简单网络单源最短路径问题的 Generic Label Correcting Algorithm 的一般流程，随后基于不同的改进方法依次介绍了：Modified Label Correcting Algorithm、FIFO Label Correcting Algorithm、Deque Label Correcting Algorithm。其中，FIFO Label Correcting Algorithm 和 Deque Label Correcting Algorithm 可以认为是 Modified Label Correcting Algorithm 的特殊实现。其次，对简单网络的多源对短路径问题进行了探讨，介绍了两种不同的求解思路：基于单源最短路径算法实现和基于 All-Pairs Generic Label Correcting Algorithm 实现，最后介绍了后者中的经典算法：Floyd-Warshall Algorithm。

从上面内容我们不难发现 Label Correcting Algorithms 是以路径最优性条件为前提展开的，Generic Label Correcting Algorithm（Modified Label Correcting Algorithm）为我们提供了基本算法框架，允许我们采用不同的规则对弧选择进行处理（对节点选择进行处理），这是 Label

Correcting Algorithms 的优势，即我们可以根据实际需要灵活调整相应规则，使得算法具有很好的拓展性。同时这也是 Label Correcting Algorithms 的缺点，不合适的处理规则会降低算法的效率。在表 3-18 中我们对上述算法的特点进行了总结供读者在选择算法时加以参考。

表 3-18　Label Correcting Algorithms

算法	时间复杂度	特点
Generic Label Correcting Algorithm	$O(\min\{n^2mC, m2^n\})$	1.选择违反最优性条件的弧并更新距离标签； 2.需要$O(m)$的时间检查违反最优性条件的弧； 3.普适性强，任何最短路算法都可以看作它的特例； 4.运行时间为伪多项式，因此效率低
Modified Label Correcting Algorithm	$O(\min\{nmC, m2^n\})$	1.是对前一个算法的改进； 2.以 LIST 储存距离标签发生更新的节点，然后从 LIST 移除一个节点，并检查与它关联的弧，更新相关距离标签； 3.为更新 LIST 中的元素提供了多种方式； 4.运行时间依然是伪多项式；
FIFO Label Correcting Algorithm	$O(nm)$	1.前一个算法的特定实现方法； 2.以先进先出的顺序将 LIST 看作一个队列； 3.实现了求解任意弧长最短路径问题的最佳强多项式运行时间； 4.在实践中更高效； 5. 在$O(nm)$时间内可以辨别出负循环的存在。

Deque Label Correcting Algorithm	$O(\min\{nmC, m2^n\})$	1.前一个算法的另一个特定实现方法； 2.将 LIST 看作双队列，如果节点已经在 LIST 中出现过，则将其添加到 LIST 头部，否则添加到尾部。
Repeated Shortest Path Algorithm	$O(nS(n,m,C))$	1.对网络进行预处理，使所有(减少的)弧算法长度都是非负的。然后以每个节点 i 为源节点，应用单源最短路径算法 n 次； 2.我们可以灵活地使用 Dijkstra 算法的实现； 3.实现所有净工作密度的最佳可用运行时间； 4.低的中间存储
Floyd-Warshall Algorithm	$O(n^3)$	1.系统地修正距离标签，直到它们代表最短路径距离； 2.非常容易实现； 3.为密集网络实现最佳可用运行时间； 4.需要$O(n^2)$中间存储

4. 扩展阅读

4.1 多目标最短路径问题求解

在前边我们介绍最短路径问题时优化的目标是找到从节点i到节点j的最短路径长度，目标是单一的。但是在很多情况下我们不仅追求最短路径，还希望通过这条路径能获取最大收益，这就是多目标最短路径问题。其中**最小成本-时间比问题**是典型的多目标最短路径问题，是指在有向图G上，每条弧都有一个成本和一个旅行时间，我们希望找到一个有向环，它的成本与

旅行时间之比最小。为了清楚说明这个问题我们以“不定期船”的问题举例说明：一艘不定期船从一个港口到另一个港口，载运货物和旅客，从i港到j港的航程赚取p_{ij}单位的利润，需要时间T_{ij}。我们想知道轮船应该去哪些港口，按照什么顺序最后回到出发点时所消耗的时间最少，获得的利益最大。我们可以通过确定一个总利润与总旅行时间之比最大的有向循环来解决这个问题。

在不定期船问题中，我们希望确定一个有向循环$W(G)$使得$(\sum_{(i,j)\in W} p_{ij})/(\sum_{(i,j)\in W} t_{ij})$最大。我们可以将弧的成本定义为$c_{ij}=-p_{ij}$，把问题转化为希望确定一个有向循环$W(G)$，使得$\mu(W)=\sum_{(i,j)\in W} c_{ij}/\sum_{(i,j)\in W} t_{ij}$最小，其中$t_{ij}>0,(i,j)\in A$。之后我们可以通过反复应用负环检测算法来解决这个问题：假设u^*为最优值，对于任意的 u 值，我们重新定义弧长$l_{ij}=c_{ij}-ut_{ij}$，并采用负环检测算法进行搜索：

1.网络 G 中找到一个负环W：即$\sum_{(i,j)\in W} l_{ij}<0$，所以$u>\frac{\sum_{(i,j)\in W} c_{ij}}{\sum_{(i,j)\in W} t_{ij}}\geq u^*$,此时$u$ 为u^*上界；

2.网络 G 中没有负环，但是存在长度为 0 的环W：即$\sum_{(i,j)\in W} l_{ij}=0$，所以$u=\frac{\sum_{(i,j)\in W} c_{ij}}{\sum_{(i,j)\in W} t_{ij}}=u^*$，此时$u=u^*$，为最优值；

3.网络 G 中所有有向环W的长度均为正值：即$\sum_{(i,j)\in W} l_{ij}\geq 0$，所以$u<\frac{\sum_{(i,j)\in W} c_{ij}}{\sum_{(i,j)\in W} t_{ij}}\leq u^*$,此时 u 为u^*下界；

通过前面的分析，我们提出以下的搜索步骤来解决最小成本时间比问题：①随机猜测一个u的值并重新定义弧长$l_{ij}=c_{ij}-ut_{ij}$；②采用最短路搜索算法，进行求解；③如果找到一条负环，则说明u值过大，将u值调小，返回第①步，如果找到一条长度为 0 的环，则找到最优值，结束迭代，如果找到一条长度为正值的环，则说明u值过小，将u值调大，返回第①步。此算法实现的关键在于如何调整u的值？下面我们介绍两种常用的方法。

（1）顺序搜索

假设已知u的上界为u^0，重新定义弧长后采用最短路算法进行搜索，如果找到一条长度为 0 的环，则已找到最优值，结束迭代；如果找到一条负环，则下一次搜索时令$u^1=(\sum_{(i,j)\in W} c_{ij})/(\sum_{(i,j)\in W} t_{ij})$。重复下去便可得到：$u^0>u^1>\cdots>u^k=u^*$，最终得到最优解。

（2）二分搜索

假设我们已知u^*所在区间为$[u_l, u_u]$，每次迭代时取$\mathrm{u} = (u_l + u_u)/2$，重新定义弧长后采用最短路算法进行搜索，如果找到一条负环，则$u > u^*$，下次迭代时更新区间上界$u_u = u$，如果找到一条长度为正值的环，则$u < u^*$，下次迭代时更新区间下界$u_l = u$，如果找到一条长度为 0 的环，则算法结束，找到最优值。在每次迭代时取值区间都缩减为原来的$1/2$，现在我们考虑如何确定初始的u_l, u_u：令C为所有弧长的最大值，则$[-C, C]$即为一个包含u^*的区间。

4.2 大规模最短路径问题求解

在解决实际问题时我们面临的网络规模可能非常大，例如国家高速公路、省主干公路与农村公路等组成的国家公路网络，又如城市快速路、主干路、街道等主城的城市公路网络，本文所介绍的几种算法显然不太适合直接求解这样规模的网络最短路径问题。

这里介绍两种解决大规模网络最短路径问题的思路。第一种，将大规模网络依据某些网络属性进行“分解”，对“分解”后的子网络进行最短路径求解，然后“组合”子网络最短路径得到原网络的近似最短路径，这种算法称为 Hierarchical Algorithm（HA）。第二种，基于并行计算框架，将算法由串行计算改进为并行计算，提高计算效率。本文以智能交通系统中近似最短路求解[6]为例给出第一种思路的一般步骤，第二种思路读者可参考文献[7]进行学习。

（1）基本思想

设计该算法的一个关键思想是根据交通网络的自然层次结构，建立一个层次化的网络拓扑结构。该网络拓扑结构包含两级网络：第一级为上层网络，这类网络对应交通网络中决策频率低（需求频率低），但决策权重重要的道路，例如高速公路；第二级为下层网络，这类网络对应交通网络中的决策频率高（需求频率高）的道路，例如街道、城市快速路等。上层网络中的节点和弧是整个网络的宏观体现，并且上层网络包裹的若干区域即为下层网络。下层网络是整个网络的某个局部的微观体现。通过这种分解，我们可以通过组合包含起点的下层网络的近似最短路径、上层网络的近似最短路径、包含终点的下层网络的近似最短路来求解网络中任意节点对之间的近似最短路。

这里需要解释下为什么对下层网络求解的最短路径称为近似最短路径。因为在求解任意两点间的最短路径时应该考虑网络中所有的弧，而求解下层网络内节点间的最短路径时仅考虑了局部范围内的弧，因此可能得到的是局部最优解，故称为近似最短路径更为合适。

（2）关键步骤

步骤一：网络分解（Decomposition）

从原始网络 G 中提取一条强连接，但不一定是完全稠密的道路，称为上层网络，$\hat{G} = (\hat{V}, \hat{A}, \hat{C}), \hat{V} \in V, \hat{A} \in \hat{V} \times \hat{V}$，$\hat{C}$为非负弧长函数，$\hat{V}$中的节点称为上层节点（宏节点），$\hat{A}$中的弧称为上层弧，上层网络中的路径称为上层路径。根据上层网络所包裹的区域，将原始网络 G 划分为H个下层网络，$G_h = (V_h, A_h, C_h), V_h \in V, V = \cup_h V_h, A_h = A \cap (V_h * V_h), \hat{V} \cap V_h \neq \emptyset, \forall h = 1,2, \dots H$。$V_h$中的节点称为下层节点，$A_h$中的弧称为下层弧，下层网络中的路径称为下层路径。此外，对于下层网络的划分方法不唯一，只要满足每个下层网络至少包含一个上层节点即可。注意：在划分下层网络时，上层网络的节点被包含在某些下层网络中。

对于任意上层弧$(\mathrm{I}, \mathrm{J}) \in \hat{A}$都对应原始网络$G$中从节点$I$到节点$J$的某些下层路径（不一定是最短路径，也可能是近似最短路径，具体取决于网络结构），其弧长为$\hat{C}(I, J)$。

步骤二：约束最短路（Constrained Shortest Paths）

分别求解上层网络$\hat{G}$和下层网络$G_h, h = 1,2, \dots H$中任意节点对的最短路径（或者根据实际问题需求求解所有最短路径，或者部分最短路径）：

上层网络最短路径：$\hat{f}: V * \hat{V} \to R$

下层网络最短路径：$f_h: V_{\mathrm{h}} * V_{\mathrm{h}} \to R, h = 1,2, \dots H$

步骤三：组合（Combination）

假设节点$i \in V_{\mathrm{h}}$，节点$j \in V_l$，如果$h = l$，则节点 i 和 j 属于同一下层网络，因此从节点 i 到节点 j 的近似最短路为步骤二所求结果。如果$h \neq l$，则分以下情况进行最短路径组合：

(i)从下层网络G_h下层节点 i 到上层网络$\hat{G}$上层节点 I 的最短路径，且$I = V_h \cap \hat{V}$。也就是说上层节点 I 是上层网络 $\hat{G}$ 和下层网络 G_h 共享节点；

(ii)上层网络$\hat{G}$中从节点 I 到节点 J 的最短路径，且上层网络节点 J 属于上层网络$\hat{G}$和下层网络 G_l 的共享节点；

(iii)下层网络 G_l 中从共享节点 I 到下层节点 j 的最短路径。

这里的(i)和(iii)中的上层节点 I 和 J 其实也是下层网络中的节点。简单来说在求解原网络任意两个节点间的最短路径时分两种情况考虑：起点和终点在同一个下层网络，起点和终点不在同一个下层网络。我们注意到在步骤三组合最短路径时可有多种方法选择，由此衍生出不同类型的 HA 算法。在下一章节将进行介绍。

（3）步骤解读

（1）网络分解方法

在层次算法的分解阶段，显然有许多可能的方法来选择上层节点、上层弧和下层网络。然而，在一个特定的应用场景中，弧通常有一个自然的层次结构。例如，在一个交通网络中，上层网络可以是由高速公路和主干公路组成，上层节点可以是高速公路的出入口，上层弧段可以是高速公路和主干公路。下层网络可以是由这些高速公路和主干公路包裹的子交通网络。有时候为了从原始交通网络中提取一个具有强连接、高密度的上层网络可以适当调整网络分解规则。

（2）最短路组合规则

如果在组合阶段，不同的下层网络分别包含起点和终点，如果每个下层网络只包含一个上层节点，那么如何根据 HA 的描述来构造近似路径是很明确的。然而，在实际中，下层网络中可能包含多个上层节点。在这种情况下，我们需要在这些上层节点中做出选择。一个直观的想法是选择最近的上层节点。换句话说，我们选择距离起点最近的上层节点，距离终点最近的上层节点。假设节点$i \in G_h$为起点，节点$j \in G_l$ $(h \neq l)$为终点，可以按以下规则选择上层节点:

$$I^* = argmin_{I \in V_h \cap \hat{V}} \ f_k(i,I) and \ J^* = argmin_{J \in V_l \cap \hat{V}} \ f_l(J,j) \quad (8)$$

基于以上节点选择规则的 HA 算法称为 Nearest HA。

显然，在所有 HA 变体中，就所获得的解决方案的质量而言(但就计算时间而言，也是最昂贵的一个)，最好的选择规则是选择产生最短近似路径的一对上层节点：

$$(I^*,J^*) = argmin_{(I,J) \in (V_h \cap \hat{V}) \times (V_l \cap \hat{V})} \ [f_h(i,I) + \hat{f}(I,J) + f_l(J,j)] \quad (9)$$

基于以上节点选择规则的 HA 算法称为 Best HA。

有时会有多个下层网络包含起点(或终点)。在这种情况下，我们将调整 Nearest HA 的上层节点选择规则：

$$I^* = argmin_{I \in \cup_{h:i \in V_h} V_h \cap \hat{V}} f_h(i,I) \quad (10)$$

D 对 Best HA 的调整类似。

（4）案例解读

假设某交通网络如图 4-1（a）所示，接下来我们按照上述步骤求解其任意节点间的近似最短路：

步骤一：网络分解。这里我们以枢纽节点及主干道构成的网络为上层网络，被上层网络所包裹的区域为若干子网络，“分解”结果如图 4-1（b）所示。4-1（b）中共有 4 个下层网络，即 H=4，分别命名为下层网络 1,2,3,4：下层网络 1 共包含普通节点 1-1,1-2,1-3,1-4,1-5 以及上

层节点 I,II,III,IV；下层网络 2 共包含普通节点 2-1,2-2,2-3 以及上层节点 III,IV,V；下层网络 3 共包含普通节点 3-1,3-2,3-3,3-4 以及上层节点 II,V；下层网络 4 共包含普通节点 4-1,4-2,4-3,4-4 以及上层节点 VI,VII。对于如何将上层网络节点分配到下层网络中没有严格要求，但应保证每个下层网络至少含有 1 个上层节点。而且分配结果将影响解的质量。

步骤二：约束最短路径。分别求出上层网络任意节点间的近似最短路径（各枢纽节点间的最短路径）、每个下层网络内任意节点间的近似最短路径。

步骤三：组合。这里需要分情况进行讨论。以节点 I 到节点 1-5 的最短路径为例，因为二者同属一个下层网络，因此步骤二中所求的最短路径即为节点 I 到节点 1-5 的近似最短路径，即对应（i）的情形；以节点 1-4 到节点 I 的最短路径为例，因为二者同属一个下层网络，因此步骤二中所求的最短路径即为节点 1-4 到节点 I 的近似最短路径，即对应（iii）的情形；以节点 1-1 到节点 2-1 的最短路径为例，因二者不在同一个下层网络，因此需要借助上层网络节点组合最短路径，首先节点 1-1 可以到达的上层节点有 I, II, III，节点 2-1 可以达到的上层节点有 III, IV, V。若采用 Nearest HA 组合规则，则分别找出距离节点 1-1 和节点 2-1 最近的上层节点，假设分别为 I, III，则节点 1-1 到节点 2-1 的近似最短路径为 $L_{1\text{-}1,I}+L_{I,III}+L_{III,2\text{-}1}$;若采用 Best HA 组合规则，则需要依据（10）进行枚举求解，对应（ii）的情形。以节点 1-1 到节点 1-5 的最短路径为例，因为二者同属一个下层网络，因此步骤二中所求的最短路径即为节点 1-1 到节点 1-5 的近似最短路径，和（i）、（iii）属于同一情形。

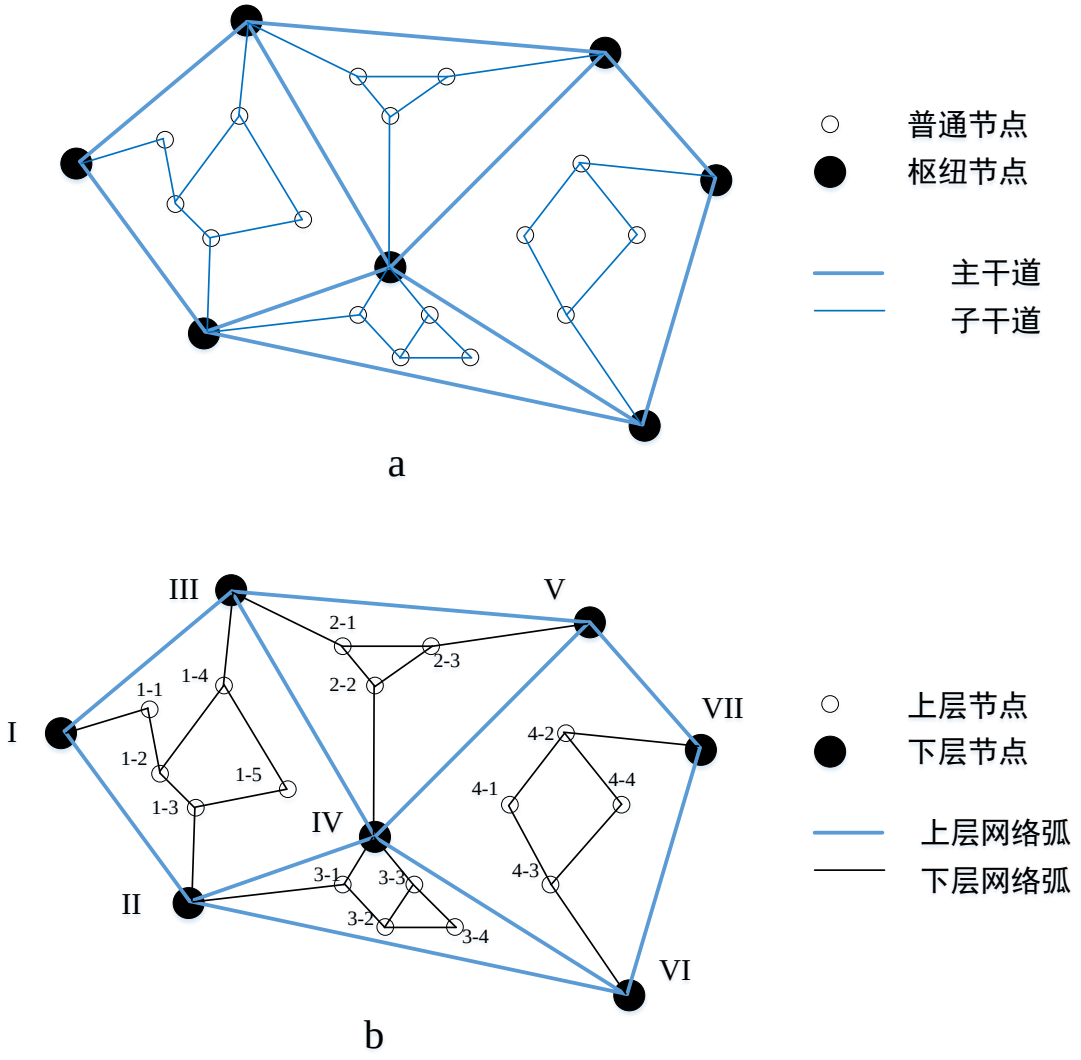


图 4-1　大规模最短路径近似算法

4.3　Time_Dependent Shortest Path Problems（时变最短路问题）

在交通领域，考虑时间因素的最短路问题是研究的热点，这类问题我们称为“Dynamic Shortest Path Problems”。这类问题具有广泛应用，如街道网络交通优化、实时智能交通系统设计等。这类问题通常具有以下特征：对于给定的有向网络$G=(N,A)$，网络中的每条弧(i,j)都与一个旅行时间（延迟）$d_{ij}(t)$相关联，它表示如果在 t 时刻从节点 i 沿着弧(i,j)出发，将会在$t+d_{ij}(t)$时刻达到节点j；此外，网络中的每条弧(i,j)还与成本$c_{ij}(t)$相关联，也就是说 t 时刻从节点 i 沿着弧(i,j)出发，达到节点 j 时花费的成本为$c_{ij}(t)$；对于某些问题来说，如果在节点i发生等待（由于某些原因需要在节点i进行停留，如在转运节点进行必要的装卸作业产生的等待），则还需要考虑在对应节点处的等待费用$w_i(t)$。

这里以文献[8-9]中的例子对时变最短路问题的特点进行简要说明。如图 4-2，弧（3,4）的长度为 1 的概率为 0.1，为 101 的概率为 0.9，可以认为弧（3，4）的状态是隐含的关于时间的函数。当我们在求解从节点 1 到节点 4 的最短路径时无法直接采用前边介绍的任何一种算法。该问题的解是由一些离散的值构成的，如：

1-2-3-4, cost=3, Pr=0.1;

1-2-3-4, cost=103, Pr=0.9;

1-2-3-1-2-3-4, cost=6, Pr=0.09;

1-2-3-1-2-3-4, cost=106,Pr=0.81;

….

我们可以计算出期望的最短路长度$\mathrm{E}(\mathrm{cost}) = 0.1 \times \sum_{i=1}^{\infty} 3 \times (i+1) \times 0.9^{i} = 30$。

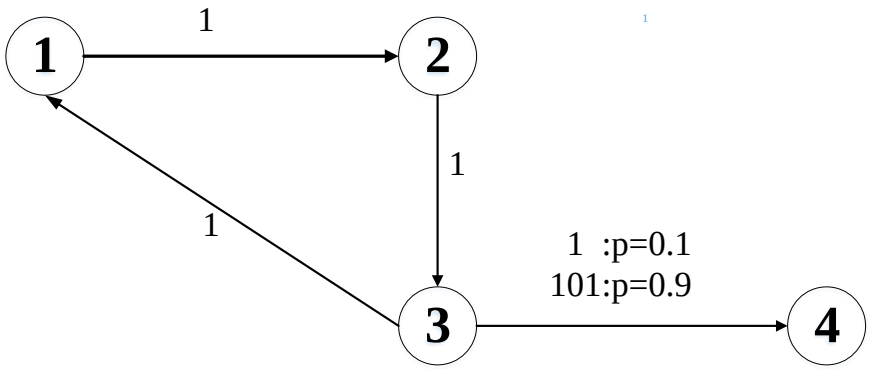


图 4-2　弧长不确定的有向网络

从上述例子我们可以总结出时变最短路问题的特点：在求解此类最短路径时每访问一次某些节点（如图 4-2 节点 3）就需要根据现有信息重新估计后续路径、最短路径中可能有环的存在等等

由于时变最短路问题较为复杂性，因此我们需要根据具体问题性质，如旅行时间（延迟）和时间成本的函数性质（离散函数、连续函数）、在网络节点发生等待的概率（不发生等待、每个节点都发生等待、只有部分节点发生等待）、从源节点出发时间的选择（固定时刻出发、不考虑出发时间、在任意可能时刻出发）以及具体问题类型来建立数学模型并设计算法求解。

一般可将时变最短路问题分为以下具体经典问题：

（1）Minimum time for a specific departure time

在给定网络源节点 r ，出发时刻 t 的条件下找出从源节点 r 到达其他非源节点 i 的最早达到时间。

（2）Minimum time for all departure times

在给定网络源节点 r 的条件下找出任意时刻从源节点 r 出发时，到达其他非源节点 i 的最小成本路径。

（3）Time windows

网络中的每个节点 i 都与一个时间窗口$[a_i, b_i]$相关联，并且只能在该时间窗口内到达该节点和离开该节点。有时候，问题也允许到达节点 i 的时刻早于 a_i，并等待至 a_i 时刻。求从源节点r到达其他非源节点 i 的最早达到时间。

关于 Dynamic Shortest Path Problems 的具体内容读者可以参考相关文献[10-11]学习，这里不再深入研究。

4.4 基于 ADMM 框架的 VRP 问题求解

在 VRP 问题的精确算法中，学者们通常使用不同的算法框架对问题进行分解，如：分支定价(Branch and Price)、分支切割定价（Branch and Cut and Price）、拉格朗日分解（Lagrangian Decomposition）和交替方向乘子法（Alternating direction method of multipliers, ADMM）等。然而，这些方法分解得到的子问题最终都指向了一类问题，那就是 Elementary Resource Constrained Shortest Path Problem (ESPPRC)。本文所介绍的 Label Correcting Algorithm 可以有效解决此类问题。接下来，本节简要介绍 ADMM 的框架以及 Label Correcting Algorithm 在其中的应用。有兴趣的读者可以通过文献[2]了解等多信息。

表 4-1　ADMM 算法框架

Algorithm ADMM
Step 1:初始化
1:　初始化迭代次数 k=0;
2:　初始化拉格朗日乘子λ_p^0 and 二次项参数ρ^0;
3:　初始化上界$\{X_{UB}^0\}$和下界$\{X_{LB}^0\}$;
4:　初始化下界 LB^*=-∞ ，上界 UB^*=+∞;
Step 2:依次求解每辆车对应的增光拉格朗日函数的最小值;
Step 2.1：每辆车的子问题调用 Label Correcting Algorithm;
5:　for each vehicle v∈V
//更新路径费用c_{ij}^v
6:　Find the Elemntary Resource Constrained Shortest Path for vehicle v by calling Label Correcting Algorithm;
7:　end
Step 2.2:更新拉格朗日乘子和二次项惩罚参数
Step 3：生成上界可行解并更新上界值

8:　若 ADMM 返回可行解，则直接与现有上界比较；如果 ADMM 未返回可行解，则先通过启发式算法得到可行解；

Step 4：生成下界可行解并更新下界值

9:　通过拉格朗日松弛求得问题下界，这里子问题依旧采用 Label Correcting Algorithm 求解；

Setp 5：计算 Gap

10:　计算上下界的 gap，$\mathbf{gap} = \frac{UB^* - LB^*}{UB^*} \times 100\%$；

附录

附录 1：负环检测

附录 1.1 单源最短路径算法负环检测

根据前边的介绍，我们知道单源 Label Correcting Algorithms 可以有效解决简单有向网络的最短路径问题。而且我们已经知道，当网络中含有负弧长时 Label Setting Algorithms 已经不适用了，那 Label Correcting Algorithms 又是如何处理负环呢？接下来我们就针对 Label Correcting Algorithms 的特点设计几种负环检测方法，帮助 Label Correcting Algorithms 处理含有负环网络最短路径问题。

（1）基于最小路径长度

在介绍最优性定理时我们讨论过，如果网络含有负环，则任意距离标签都不满足最优性条件，也就是说距离标签会无限减小，算法永远不会结束。但我们应注意到，如果一个网络不含负环时，任何一条路径的长度不会小于$-nC$（n：网络弧数量，C最大弧长度），因此当我们发现某些节点 k 的距离标签小于$-nC$时应结束算法，我们可以通过追溯节点 k 的前向节点来得到负环。

（2）基于前向图

首先回顾一下 3.2.1 小节的内容，通过对图 3-3 的分析我们知道，当网络中不含负环时通过前向节点可以构造一棵以源节点为根的前向树，而当网络中含有负环时无法构造一棵以源节点为根的前向树。因此我们可以根据这个特点来检测网络中是否含有负环：

我们将源点s记为标记点，而其他节点记为未标记点，然后选择一个未标记点k将其标记，从节点k开始追溯其前向节点并将追溯的点进行标记，直到找到第一个已标记节点i，如果$k = i$，则该网络含有负环。每次检测所需要的时间为$\boldsymbol{O(n)}$，因此在算法执行过程中检测的次数将影响算法的效率。我们可以以一定间隔执行负环检测。

（3）FIFO Label Correcting Algorithm

我们知道 FIFO Label Correcting Algorithm 每次迭代时检查一个节点，而每个节点最多被遍历$n-1$次（最多迭代$n-1$次），因此我们可以记录每个节点被遍历的次数，如果节点实际遍历次数大于$n-1$，则该网络含有负环。该方法仅对 FIFO Label Correcting Algorithm 有效。

以上介绍了三种负环检测方法，其中方法 1 和方法 2 适用于所有 Label Correcting Algorithms，而方法 3 仅适用于 FIFO Label Correcting Algorithm。

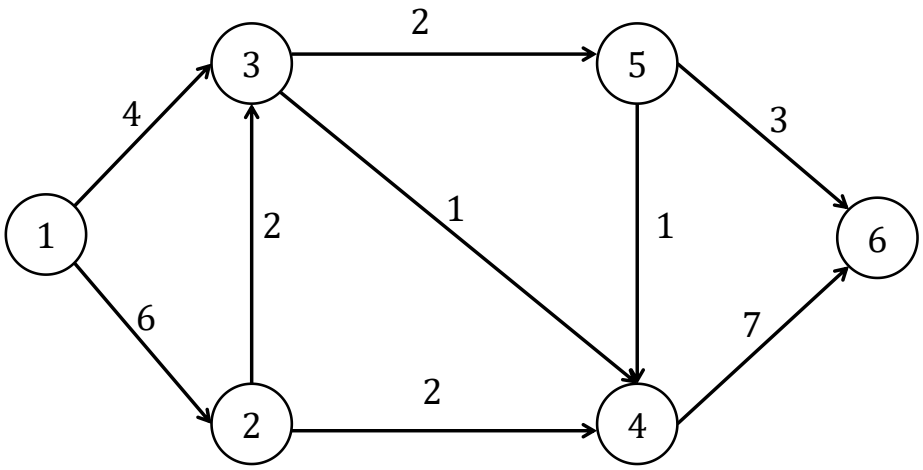
附录 1.2 多源最短路径算法负环检测

接下来我们介绍多源 Label Correcting Algorithms 处理含有负环的网络最短路径问题方法。All-pairs Label Correcting Algorithms 的负环检测相对简单，只需要在更新距离标签时附加以下判断条件：

$$\begin{aligned}&\text{If i=j, check whether } d[i,j]<0;\\&\text{If i!=j, check whether } d[i,j]<-nC\end{aligned}\qquad(11)$$

通过以上判断条件算法便能处理含有负环的最短路径问题：如果满足以上任何条件，则此网络含有负环。首先考虑第一种情况：当算法第一次更新距离标签$d[i,i]<0$时，必然存在节点$k\neq i$,使得$d[i,i]=d[i,k]+d[k,i]$，也就是说存在一条从节点 i 到节点 k 和从节点 k 到节点 i 的路径，并且两条路径的长度之和$d[i,i]<0$，又因为这两条路径构成一个环，所以网络必定含有一个负环。接下来考虑第二种情况：当算法第一次更新距离标签$d[i,i]<-nC$时，我们将这条道路分解为从节点 i 到节点 j 的一条路径 P 和一些环，我们前边提到最短路径的取值范围是$[-nC,nC]$，因此必然存在负环使得$d[i,j]<-nC$。

附录 2：简单有向网络



附录 3：NeXTA 辅助工具

为便于读者理解本文所介绍的最短路径求解算法以及相应代码输出结果，本文提供了由周老师开发的 NeXTA 可视化工具，可对算法输入、算法输出进行可视化，给读者以直观感受。软件及其相关资源链接如下：

https://github.com/zephyr-data-specs/GMNS/

https://github.com/xzhou99/NeXTA-GMNS

https://github.com/xzhou99/NeXTA-GMNS/tree/master/examples/GMNS_AMS_Networks/6-node_network

这里仅给出简单的最短路径可视化步骤：

（1） NeXTA 输入文件

网络节点文件（node.csv）与网络弧文件（road_link.csv）：如第 3 章所介绍格式即可。

最短路径文件（agent.csv）:算法输出文件，格式如下图所示：

A	B	C	D	E
agent_id	o_zone_id	d_zone_id	node_sequence	distance
1	1	2	1;2	6
2	1	3	1;3	4
3	1	4	1;3;4	5
4	1	5	1;3;5	6
5	1	6	1;3;5;6	9
6				99999
7	2	3	2;3	2
8	2	4	2;4	2
9	2	5	2;3;5	4

agent_id：最短路径编号；

o_zone_id：路径起点所属节点类型；

d_zone_id：路径终点所属节点类型；

node_sequence：路径节点顺序；

distance：路径长度（根据具体问题也可代表费用等值）；

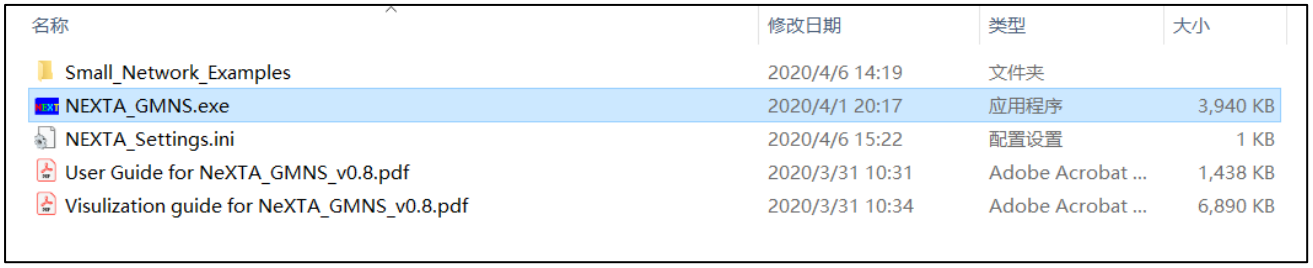
99999：路径长度为无穷大（本文算法以此代表路径长度无穷大）；

若某条路径不存在时可以将对应的 o_zone_id，d_zone_id，node_sequence 设为空，或者删除该路径。

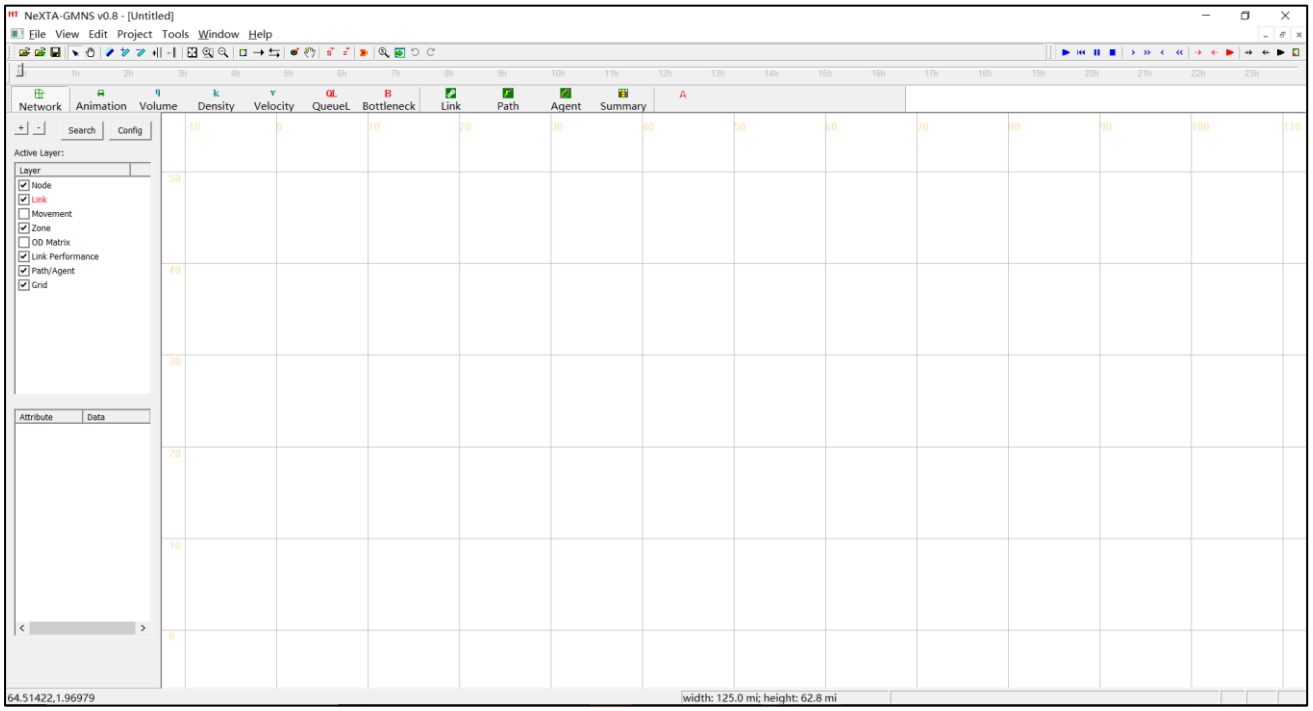
（2）基本操作流程

①将 node.csv，road_link.csv，agent.csv 文件置于“Small_Network_Examples 文件夹”；

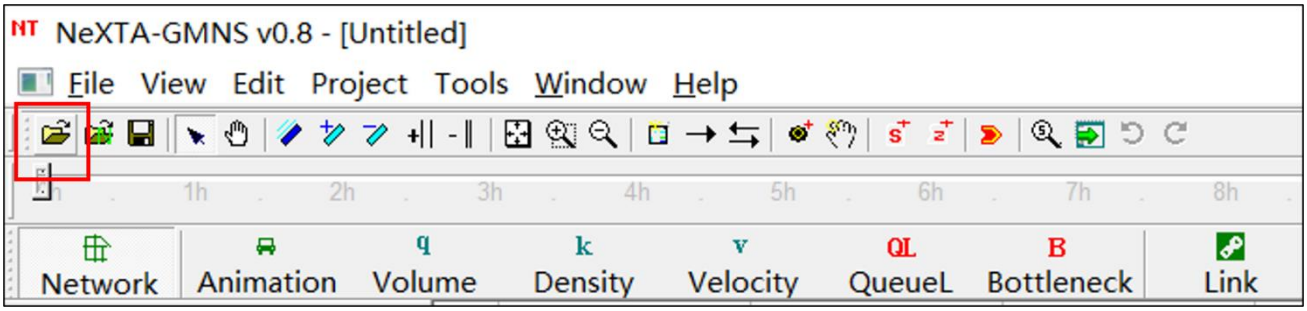
②双击软件图标，打开软件；

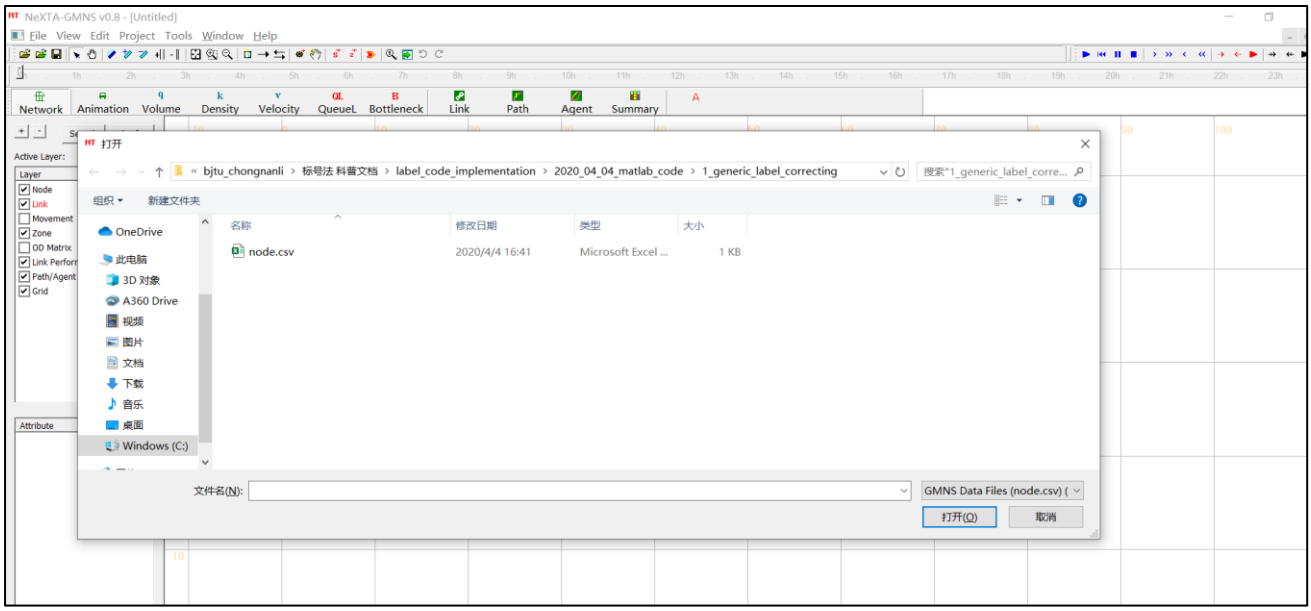


打开后，软件界面如下图所示：

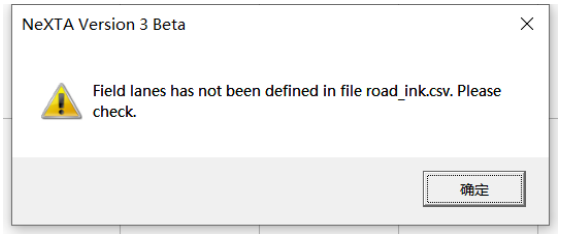


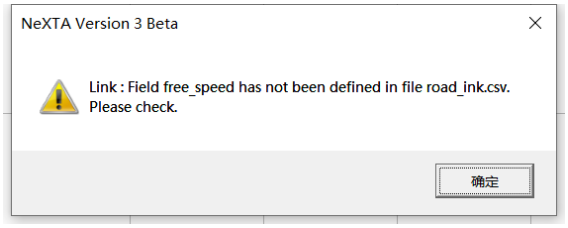
③读取 node.csv 文件，构建网络。点击“文件夹图标”，弹出导航窗格，找到“Small_Network_Examples 文件夹”下 node.csv 文件，选中打开。

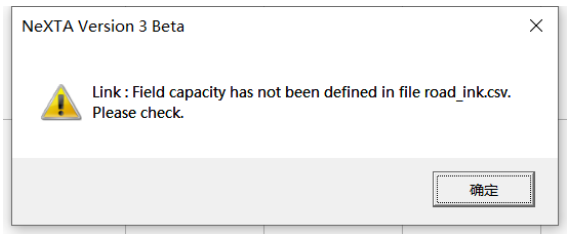




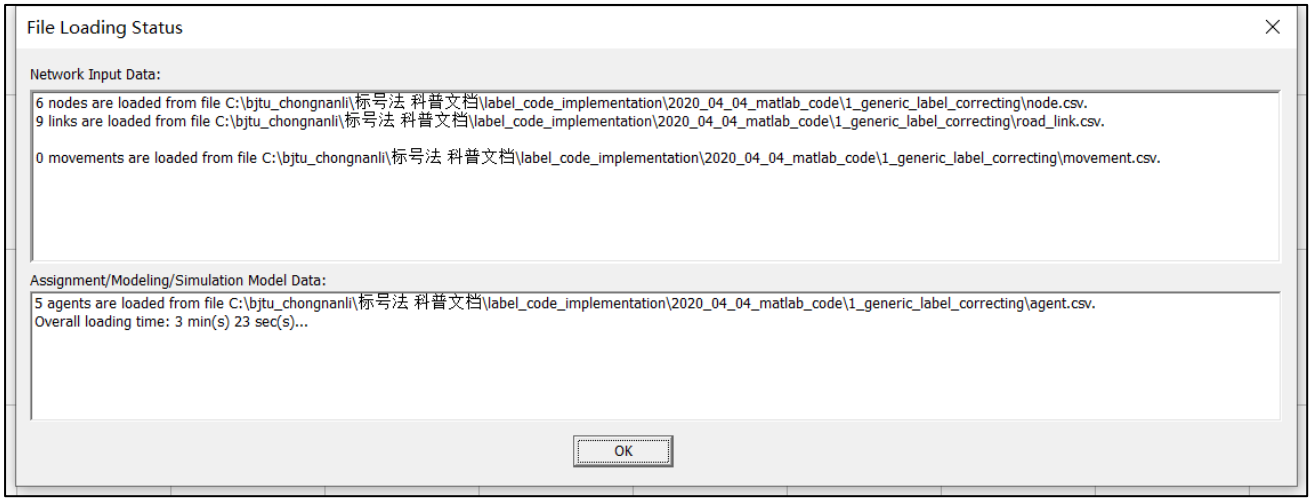
之后软件会弹出如下的一系列警告提示，这些提示用户目前可以不用关心，均点击确定即可。



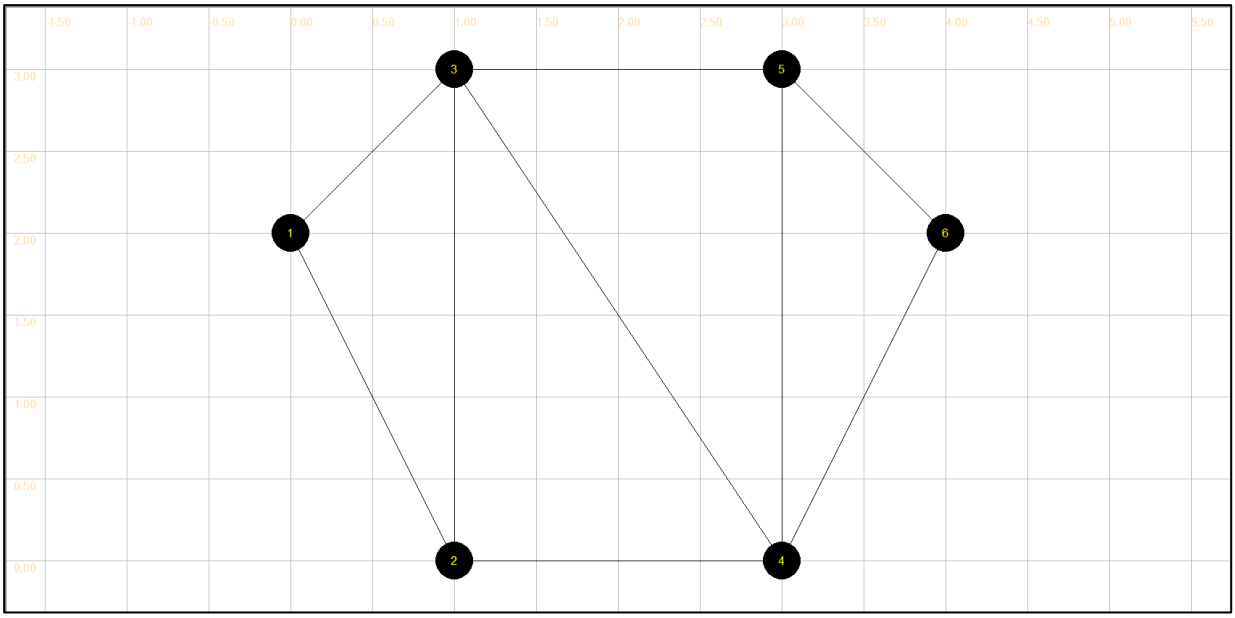




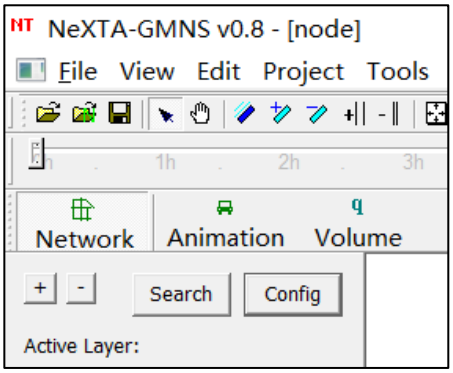
之后，软件会弹出一个名为“File Loading Status”的导航窗格，根据其显示信息，软件成功地将 node.csv 与 road_link.csv 两个文件读取了，且节点与弧的数量信息也是正确的。同时，软件将 agent.csv 中的最短路信息也读取成功了。

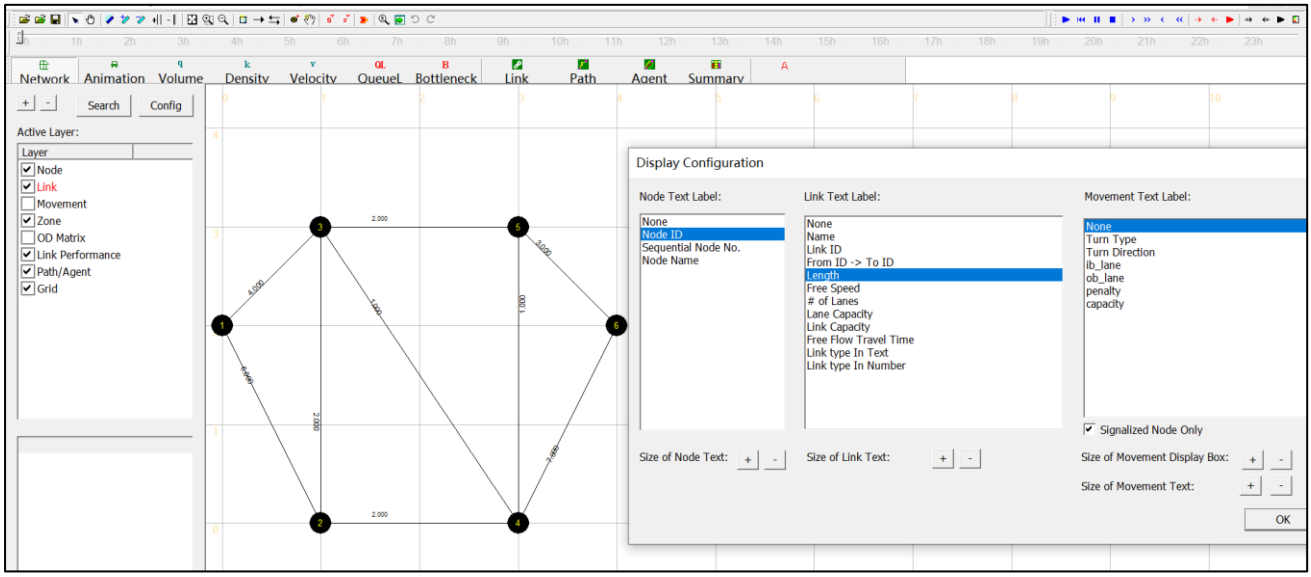


点击 OK 之后，软件主界面展示了读取的网络（可以通过鼠标对网络进行拖动和缩放）。

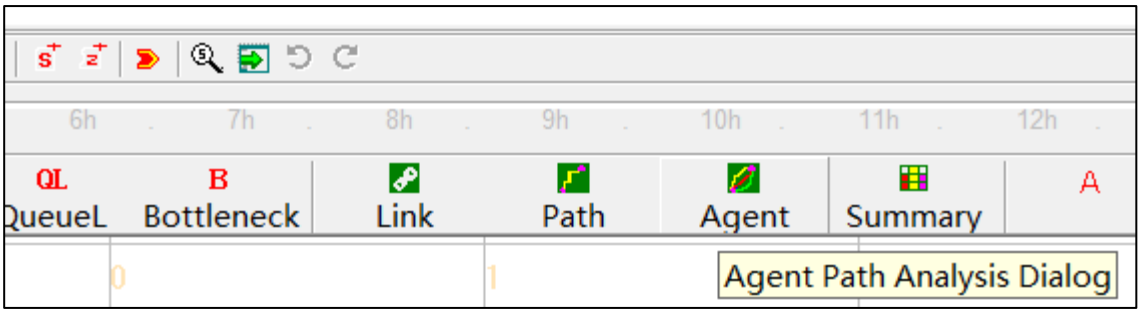


④设置网络属性。点击软件左侧的“Config 按钮”，在弹出的导航窗格中点击 Link Text 分组下的 Length 选项，这时网络的弧上就标明了距离信息。用户如果想要显示其他信息，可以点击相关按钮实现。

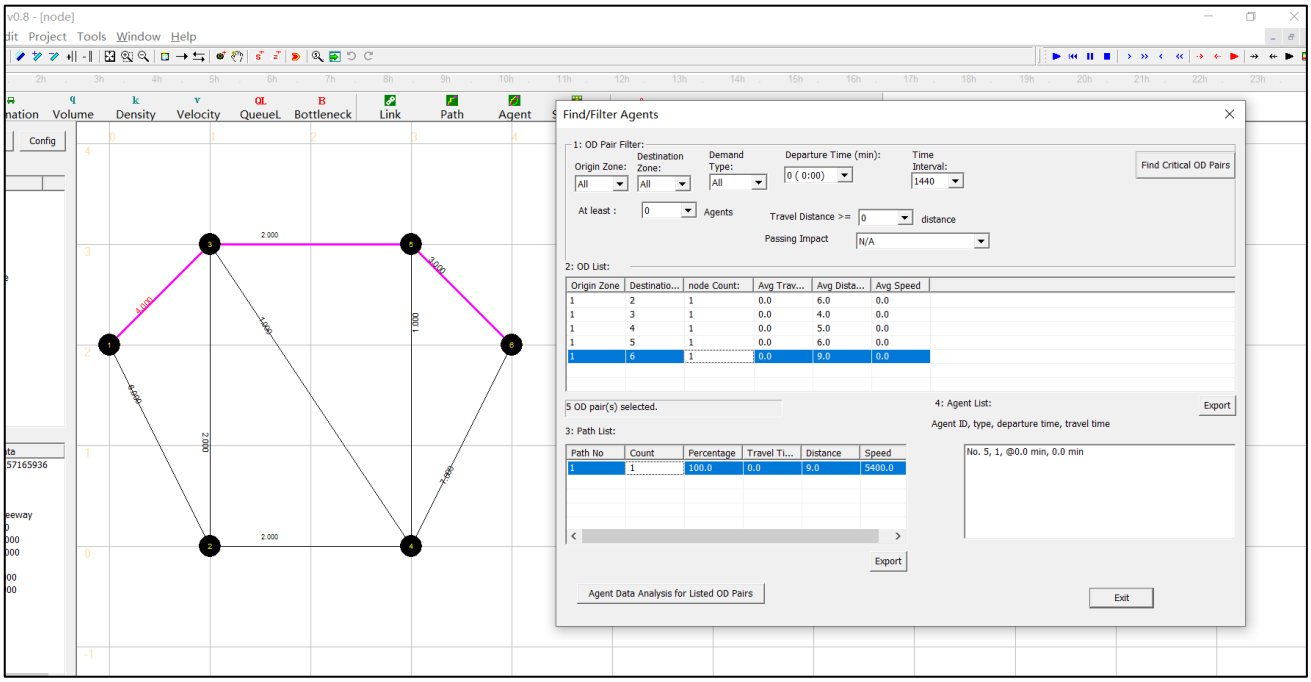




⑤可视化最短路径。点击软件上方的“Agent 按钮”



如图所示，在弹出的导航窗格内，先在“OD List”栏中选中需要可视化的最短路径，相应路径信息会出现在“Path List”栏中，然后单击“Path List”栏中的最短路径即可。此时，网络中紫色的弧便展示了最短路径。



参考文献

[1]. Noon F B Z E . Shortest Path Algorithms: An Evaluation using Real Road Networks[J]. Transportation Science, 1998, 32(1):65-73.

[2]. Yao Y, Zhu X, Dong H, et al. ADMM-based problem decomposition scheme for vehicle routing problem with time windows[J]. Transportation Research Part B: Methodological, 2019, 129: 156-174.

[3]. Ahuja R K , Ahuja R K , Ahuja R K . Network Flows[M]. Optimization. Elsevier North-Holland, Inc. 1989.

[4]. Mahmassani, Hani S. Development and testing of dynamic traffic assignment and simulation procedures for ATIS/ATMS applications[J]. 1994.

[5]. U. Pape. Implementation and Efficiency of Moore-Algorithms for the Shortest Route Problem[J]. Mathematical Programming, 1974: 212-222.

[6]. Chou Y L, Romeijn H E, Smith R L. Approximating shortest paths in large-scale networks with an application to intelligent transportation systems[J]. INFORMS journal on Computing, 1998, 10(2): 163-179.

[7]. Habbal M B, Koutsopoulos H N, Lerman S R. A decomposition algorithm for the all-pairs shortest path problem on massively parallel computer architectures[J]. Transportation Science, 1994, 28(4): 292-308.

[8]. Waller S T , Ziliaskopoulos A K . On the online shortest path problem with limited arc cost dependencies[J]. Networks, 2002, 40(4):216-227.

[9]. AK Ziliaskopoulos, HS Mahmassani. Real-time multivehicle truckload pickup and delivery problems[J]. Transportation Research Record, 1993: 94-100.

[10]. Pallottino, Stefano, and Maria Grazia Scutella. Shortest path algorithms in transportation models: classical and innovative aspects[J]. Equilibrium and advanced transportation modelling. Springer, Boston, MA, 1998: 245-281.

[11]. Yang, Lixing, and Xuesong Zhou. Constraint reformulation and a Lagrangian relaxation-based solution algorithm for a least expected time path problem[J]. Transportation Research Part B: Methodological, 2014, 59: 22-44.