

TBSI 清华-伯克利深圳学院 Tsinghua-Berkeley Shenzhen Institute

Tsinghua University

Tsinghua-Berkeley Shenzhen Institute
Department of Industrial Engineering

运小筹

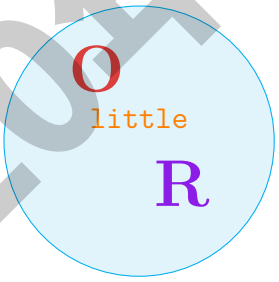
OlittleRer

运小筹月刊　2020 年 12 月

本期作者：王基光　刘兴禄　夏旸　周鹏翔　张祎　修宇璇

本期主编：周鹏翔　徐璐　何彦东　修宇璇

本期发布：段淇耀　段宏达



投稿请联系：hsinglul@163.com，欢迎广大同行的投稿！

目录

0.1 运小筹团队成员

运小筹编委：运小筹团队编委 (成员) 的单位及联系方式如下：

刘兴禄 清华大学，清华-伯克利深圳学院，邮箱：hsinglul@163.com

何彦东 清华大学，深圳国际研究生院 (博士后)，邮箱：ydhe602@163.com

段淇耀 清华大学，清华-伯克利深圳学院，邮箱：duanqy71@163.com

修宇璇 清华大学，清华-伯克利深圳学院，邮箱：yuxuanxiu@gmail.com

曾文佳 清华大学，工业工程系，邮箱：zwj19@mails.tsinghua.edu.cn

夏旸 清华大学，工业工程系，邮箱：xia970201@gmail.com

王梦彤 清华大学，工业工程系，邮箱：wmt15@mails.tsinghua.edu.cn

段宏达 清华大学，工业工程系，邮箱：dhd18@mails.tsinghua.edu.cn

王鑫 清华大学，工业工程系，邮箱：wangxin16@mails. tsinghua. edu. cn

王涵民 清华大学，清华-伯克利深圳学院，邮箱：humminwang@163.com

张祎 清华大学，清华-伯克利深圳学院，邮箱：zhangyihrmm2015@163.com

王基光 清华大学，清华-伯克利深圳学院，邮箱：wangjg2020@163.com

陈琰钰 清华大学，清华-伯克利深圳学院，邮箱：yanyu_chen_98@163.com

周鹏翔 清华大学，清华-伯克利深圳学院，邮箱：zpx20@mails.tsinghua.edu.cn

徐璐 清华大学，清华-伯克利深圳学院，邮箱：xu-l20@mails.tsinghua.edu.cn

臧永森 清华大学，工业工程系，邮箱：zangys15@tsinghua.org.cn

左齐茹仪 清华大学，清华-伯克利深圳学院，邮箱：zqry20@mails.tsinghua.edu.cn

0.2 运小筹公众号简介

本公众号是致力于分享运筹优化 (LP、MIP、NLP、随机规划、鲁棒优化)、凸优化、强化学习等研究领域的内容以及涉及到的算法的代码实现。编程语言和工具包括 Java、Python、Matlab、CPLEX、Gurobi、SCIP 等。

欢迎读者朋友们加入我们的读者群，大家一起探讨学术问题。

Figure 0.2.1: 运小筹读者 *4* 群二维码

Chapter 1

2020-12-05: 手把手教你用 Python 调用 Gurobi 求解最短路问题

王基光，清华大学，清华伯克利深圳学院（硕士在读）
刘兴禄，清华大学，清华伯克利深圳学院（博士在读）

1.1 最短路问题 Shortest Path Problem

最短路径问题是图论研究中的一个经典算法问题, 旨在寻找图 (由结点和路径组成的) 中两结点之间的最短路径。

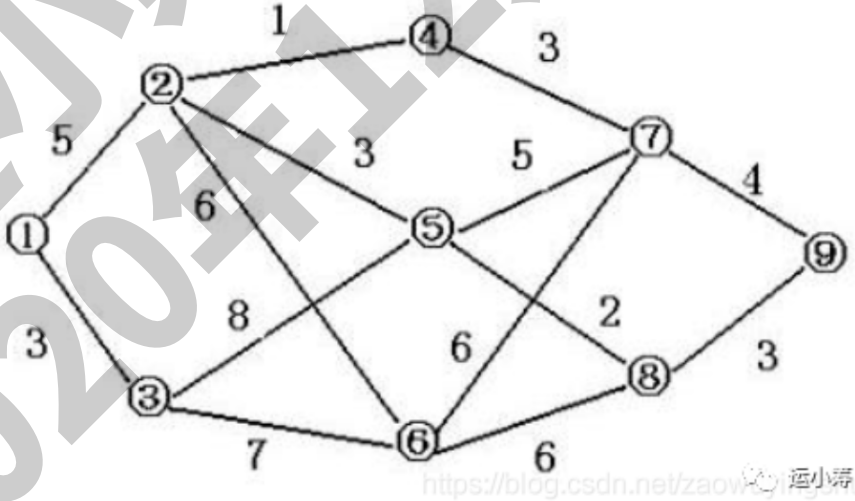


Figure 1.1.1: *network*

1.1.1 符号说明

- c_{ij}: 弧 (i,j) 上的权重，可以理解为距离，流量，时间等
- x_{ij}: 弧 (i,j) 是否被选择通过，是 binary 变量
- V: 图中的点集合

1.1.2 数学模型

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.1.1}$$

$$\sum_{(j,i)\in A} x_{ij} - \sum_{(i,j)\in A} x_{ji} = b_i, \qquad \forall i \in V, \tag{1.1.2}$$

其中

$$b_i = \begin{cases} -1, & \text{if } i = s, \\ 0, & \text{if } i \neq s \text{ and } i \neq t, \\ 1, & \text{if } i = t, \end{cases} \tag{1.1.3}$$

1.1.3 python 调用 gurobi 求解

问题数据

这里我们使用距离矩阵来表示图。

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11
v1	0	2	8	1	∞	∞	∞	∞	∞	∞	∞
v2	2	0	6	∞	1	∞	∞	∞	∞	∞	∞
v3	8	6	0	7	5	1	2	∞	∞	∞	∞
v4	1	∞	7	0	∞	∞	9	∞	∞	∞	∞
v5	∞	1	5	∞	0	3	∞	2	∞	∞	∞
v6	∞	∞	1	∞	3	0	4	∞	6	∞	∞
v7	∞	∞	2	9	∞	4	0	∞	3	1	∞
v8	∞	∞	∞	∞	2	∞	∞	0	7	∞	9
v9	∞	∞	∞	∞	∞	6	3	7	0	1	2
v10	∞	∞	∞	∞	∞	∞	1	∞	1	0	4
v11	∞	∞	∞	∞	∞	∞	∞	9	2	4	0

Figure 1.1.2: *Distance Matrix*

代码

代码如下（示例）：

Code

```
# _*_coding:utf-8 _*_
from __future__ import print_function
from __future__ import division, print_function
from gurobipy import *
import numpy as np
import copy
import time

starttime = time.time()


# 返回最优的路线
def reportMIP(model, Routes):
```

```
    if model.status == GRB.OPTIMAL:
        print("Best MIP Solution: ", model.objVal, "\n")
        var = model.getVars()
        for i in range(model.numVars):
            if (var[i].x > 0):
                print(var[i].varName, " = ", var[i].x)
                print("Optimal route:", Routes[i])


def getValue(var_dict, nodeNum):
    x_value = np.zeros([nodeNum, nodeNum])
    for key in var_dict.keys():
        a = key[0]
        b = key[1]
        x_value[a][b] = var_dict[key].x

    return x_value


def getRoute(x_value):
    x = copy.deepcopy(x_value)
    #     route_temp.append(0)
    previousPoint = 0
    route_temp = [previousPoint]
    count = 0
    while (count != len(x_value)-1):
        # print('previousPoint: ', previousPoint )
        if (x[previousPoint][count] > 0):
            previousPoint = count
            route_temp.append(previousPoint)
            count = 0
            continue
        else:
            count += 1
    route_temp.append(len(x_value)-1)
    return route_temp

if __name__ == "__main__":


    nodeNum =11

# 距离矩阵
    cost =[[0,2,8,1,1000,1000,1000,1000,1000,1000,1000],
           [2,0,6,1000,1,1000,1000,1000,1000,1000,1000]
          ,[8,6,0,7,5,1,2,1000,1000,1000,1000]
          ,[1,1000,7,0,1000,1000,9,100,100,100,100]
          ,[100,1,5,100,0,3,100,2,100,100,100]
           ,[100,100,1,100,3,0,4,100,6,100,100]
           ,[100,100,2,9,100,4,0,100,3,1,100]
```

```
            ,[100,100,100,100,2,100,100,0,7,100,9]
            ,[100,100,100,100,100,6,3,7,0,1,2]
            ,[100,100,100,100,100,100,1,100,1,0,4]
            ,[100,100,100,100,100,100,100,100,9,2,4,0]
        ]
    print("cost", cost)
    model = Model('TSP')

    # creat decision variables, 决策变量
    X = {}
    for i in range(nodeNum):
        for j in range(nodeNum):
            if (i != j):
                X[i, j] = model.addVar(vtype=GRB.BINARY
                                       , name='x_' + str(i) + '_' + str(j)
                                       )

    # set objective function, 目标函数

    obj = LinExpr(0)
    for key in X.keys():
        i = key[0]
        j = key[1]
        obj.addTerms(cost[key[0]][key[1]], X[key])

    model.setObjective(obj, GRB.MINIMIZE)

    # add constraints 出发点的流量约束

    lhs_1 =LinExpr(0)
    lhs_2 =LinExpr(0)
    for j in range(1, nodeNum-1):
        for i in range(0,nodeNum):
            if i == 0:
                lhs_1.addTerms(1, X[i, j])
    model.addConstr(lhs_1 == 1, name='visit_' + str(i)+"start")

    # 终点的流量约束
    for i in range(1,nodeNum-1):
        for j in range(1, nodeNum):
            if j == nodeNum-1:
                lhs_2.addTerms(1, X[i, j])
    model.addConstr(lhs_2 == 1, name='visit_' + str(j) + "end")

    # 其余点的流量约束
    for j in range(1,nodeNum-1):
        lhs3 = LinExpr(0)
        for i in range(0, nodeNum-1):
            if i != j:
                lhs3.addTerms(1, X[i, j])
```

```
        for i in range(1, nodeNum):
            if i != j:
                lhs3.addTerms(-1, X[j, i])
        model.addConstr(lhs3 == 0, name='visit_' + str(j)+'balance_flow')

    model.write('modelshortestpath.lp')
    model.setParam(GRB.Param.MIPGap, 0)
    model.setParam(GRB.Param.TimeLimit, 60)
    model.optimize()
    # 可以输出求解的决策变量
    print(model.ObjVal)
    X_NEW={}
    for var in model.getVars():
        if (var.x > 0):
            print(var.varName, '\t', var.x)
            X_NEW[var.varName]=var.x

    x_value = getValue(X, nodeNum)
    route = getRoute(x_value)
    print('optimal route:', route)
```

1.1.4 结果展示

```
Optimal solution found (tolerance 0.00e+00)
Best objective 1.300000000000e+01, best bound 1.300000000000e+01, gap 0.0000%
optimal route: [0, 1, 4, 5, 2, 6, 9, 8, 10]
```


Figure 1.1.3: *results*

最短距离为: 13 最短路径为: $[0, 1, 4, 5, 2, 6, 9, 8, 10]$

Chapter 2

2020-12-08: 手把手教你用 Python 实现 Dijkstra 算法（附 Python 代码和可视化）

王基光，清华大学，清华伯克利深圳学院（硕士在读）
刘兴禄，清华大学，清华伯克利深圳学院（博士在读）

2.1 前言

迪杰斯特拉算法 (Dijkstra) 是由荷兰计算机科学家狄克斯特拉于 1959 年提出的，因此又叫狄克斯特拉算法。是从一个顶点到其余各顶点的最短路径算法，解决的是有权图中最短路径问题。迪杰斯特拉算法主要特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。该算法在运筹学和数据结构图论部分都有介绍，是一种非常有效的求解单源最短路问题的算法

提示：以下是本篇文章正文内容，下面案例为 Solomon 标准算例

2.2 最短路模型

详见之前的博文: https://blog.csdn.net/zaowuyingshu/article/details/110227481

2.3 迪杰斯特拉算法

原理如下，简单地说就是通过标记的办法来逐渐确定和更新临时节点和最终节点的列表，最终通过回溯上一个节点的办法来得到最短路径，废话少说，直接上代码！

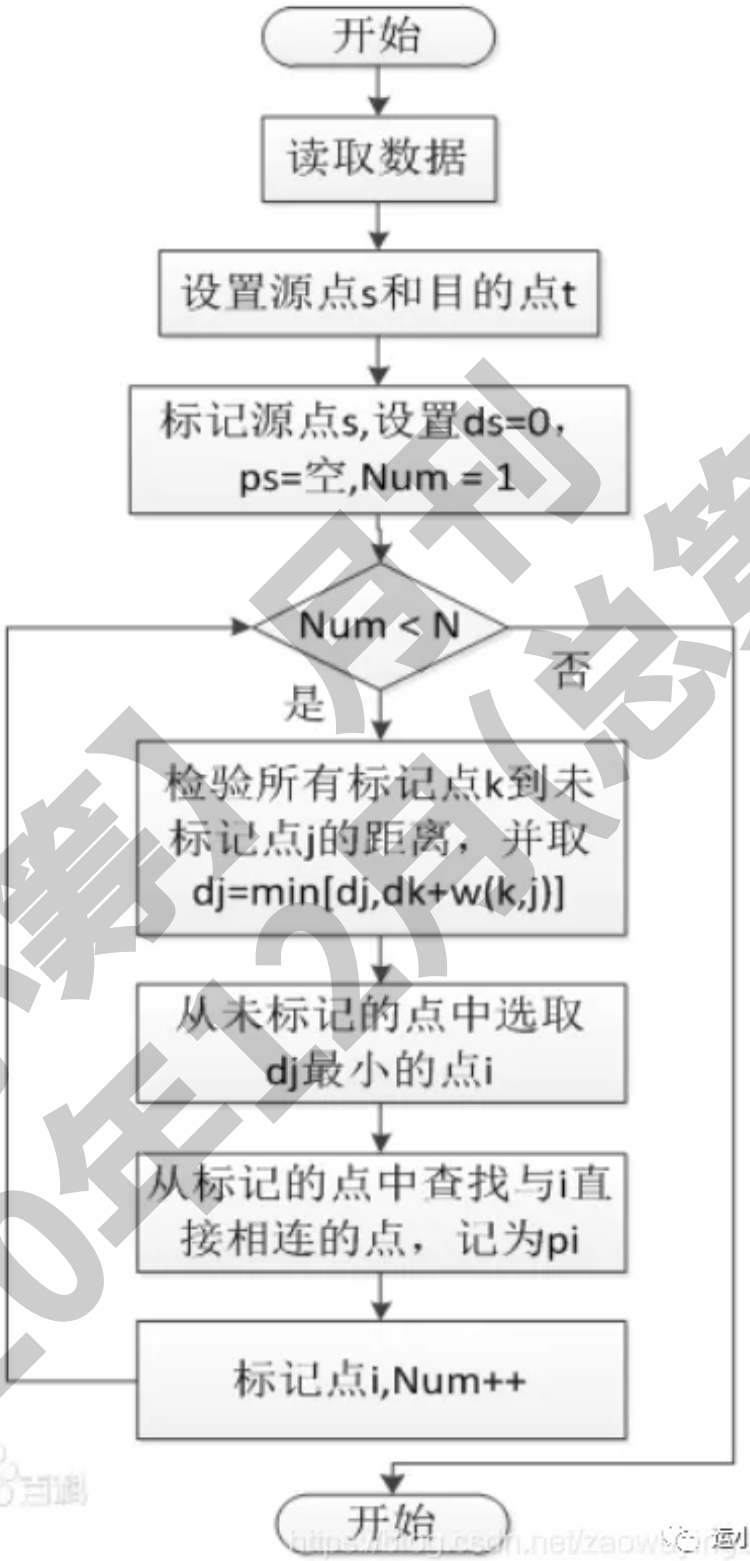


Figure 2.3.1: *Dijkstra* 算法流程图

2.4 python 代码实现

代码如下：

案例中选择使用 Solomon 标准算例的 1000 个节点。随机设置起点和终点为 418,154 号 node。

Code

```
# _*_coding:utf-8 _*_
from __future__ import print_function
from __future__ import division, print_function
from gurobipy import *
import re
import math
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import copy
from matplotlib.lines import lineStyles
import time
import networkx as nx

starttime = time.time()

class Data():
    '''
    the format of solomon dataset
    '''

    def __init__(self):
        self.customerNum = 0  # the number of customers
        self.nodeNum = 0  # the sum of customers and depots
        self.vehicleNum = 0
        self.capacity = 0
        self.cor_X = []
        self.cor_Y = []
        self.demand = []
        self.readyTime = []
        self.dueTime = []
        self.serviceTime = []
        self.disMatrix = {}

    def read_data(self, path, customerNum, depotNum):
        '''
        function to read solomom data from .txt files, notice that it must be solomon dataset
        INPUT
            @ data  : class Data
```

```
40              @ path  : Data path
41              @ customerNum  : the number of customer
42          OutPut : none
43          '''
44          self.customerNum = customerNum
45          self.nodeNum = customerNum + depotNum
46          f = open(path, 'r')
47          lines = f.readlines()
48          count = 0
49          for line in lines:
50              count = count + 1
51              if (count == 5):
52                  line = line[:-1].strip()
53                  str = re.split(r" +", line)
54                  self.vehicleNum = int(str[0])
55                  self.capacity = float(str[1])
56              elif (count >= 10 and count <= 10 + customerNum):
57                  line = line[:-1]
58                  str = re.split(r" +", line)
59                  self.cor_X.append(float(str[2]))
60                  self.cor_Y.append(float(str[3]))
61                  self.demand.append(float(str[4]))
62                  self.readyTime.append(float(str[5]))
63                  self.dueTime.append(float(str[6]))
64                  self.serviceTime.append(float(str[7]))
65
66          # compute the distance matrix
67          self.disMatrix = {}
68          for i in range(0, self.nodeNum):
69              dis_temp = {}
70              for j in range(0, self.nodeNum):
71                  dis_temp[j] = int(math.hypot(self.cor_X[i] - self.cor_X[j], self.cor_Y[i] -
                        ↪  self.cor_Y[j]))
72              self.disMatrix[i] = dis_temp
73
74      def plot_nodes(self):
75          '''
76          Description: function to plot
77          '''
78          Graph = nx.DiGraph()
79          nodes_name = [str(x) for x in list(range(self.nodeNum))]
80          Graph.add_nodes_from(nodes_name)
81          cor_xy = np.array([self.cor_X, self.cor_Y]).T.astype(int)
82          pos_location = {nodes_name[i]: x for i, x in enumerate(cor_xy)}
83          nodes_color_dict = ['r'] + ['gray'] * (self.nodeNum - 1)
84
85          nx.draw_networkx(Graph, pos_location, node_size=200, node_color=nodes_color_dict,
                ↪  labels=None,font_size =8)
86          plt.show()
87
```

```
88      def plot_route(self, route, color='k'):
89          Graph = nx.DiGraph()
90          nodes_name = [route[0]]
91          cor_xy = [[self.cor_X[route[0]], self.cor_Y[route[0]]]]
92          edge = []
93          edges = [[route[0],route[1]]]
94          for i in route[1:]:
95              nodes_name.append(i)
96              cor_xy.append([self.cor_X[i], self.cor_Y[i]])
97              edge.append(i)
98              if len(edge) == 2:
99                  edges.append(copy.deepcopy(edge))
100                 edge.pop(0)
101
102         Graph.add_nodes_from(nodes_name)
103         Graph.add_edges_from(edges)
104
105         pos_location = {nodes_name[i]: x for i, x in enumerate(cor_xy)}
106         nodes_color_dict = ['r'] + ['gray'] * (len(route)-2)+['r']
107
108         nx.draw_networkx(Graph, pos_location, node_size=180, node_color=nodes_color_dict,
            ↪  edge_color=color, labels=None,font_size =8)
109         plt.show()
110
111
112 # function to read data from .txt files
113 def readData(path, nodeNum):
114     nodeNum = nodeNum;
115     cor_X = []
116     cor_Y = []
117
118     f = open(path, 'r');
119     lines = f.readlines();
120     count = 0;
121     # read the info
122     for line in lines:
123         count = count + 1;
124         if (count >= 10 and count <= 10 + nodeNum):
125             line = line[:-1]
126             str = re.split(r" +", line)
127             cor_X.append(float(str[2]))
128             cor_Y.append(float(str[3]))
129
130     # compute the distance matrix
131     disMatrix = [([0] * nodeNum) for p in range(nodeNum)];
132     for i in range(0, nodeNum):
133         for j in range(0, nodeNum):
134             temp = (cor_X[i] - cor_X[j]) ** 2 + (cor_Y[i] - cor_Y[j]) ** 2
135             disMatrix[i][j] = (int)(math.sqrt(temp))
136             temp = 0
```

```
137
138     return disMatrix
139
140
141 def printData(disMatrix):
142     print("-------cost matrix-------\n");
143     for i in range(len(disMatrix)):
144         for j in range(len(disMatrix)):
145             # print("%d   %d" % (i, j));
146             print("%6.1f" % (disMatrix[i][j]), end=" ");
147         #             print(disMatrix[i][j], end = " ");
148         print()
149
150
151
152
153
154 def Dijkstra(Graph, org, des):
155         Q = list(Graph.nodes())
156         for node in Q:
157             if (node == org):
158                 Graph.nodes[node]['min_dis'] = 0
159             else:
160                 Graph.nodes[node]['min_dis'] = np.inf
161         current_node = org
162         while (len(Q) > 0):
163             min_dis = np.inf
164             for node in Q:
165                 if (Graph.nodes[node]['min_dis'] < min_dis):
166                     current_node = node
167                     min_dis = Graph.nodes[node]['min_dis']# 找最小的距离的点
168             if (current_node != None):
169                 Q.remove(current_node)
170             for child in Graph.successors(current_node):
171                 arc = (current_node, child)
172                 dis_temp = Graph.nodes[current_node]['min_dis'] + Graph.edges[arc]['length']
173                 if (dis_temp < Graph.nodes[child]['min_dis']):
174                     Graph.nodes[child]['min_dis'] = dis_temp
175                     Graph.nodes[child]['previous_node'] = current_node
176         min_dis = Graph.nodes[des]['min_dis']
177         current_node =des
178         shortest_path = [current_node]
179         while (current_node != org):
180             current_node = Graph.nodes[current_node]['previous_node']
181             shortest_path.insert(0, current_node)
182         return  shortest_path, min_dis
183
184 if __name__ == "__main__":
185
186     data = Data()
```

```
187     path1 = 'C:/Users/asus/Desktop/vrp/Solomn 标准 VRP 算
        ↪  例/homberger_1000_customer_instances/R1_10_1.txt'
188     data.read_data(path=path1, customerNum=1000, depotNum=1)  # change the path
189
190     nodeNum =1000
191     cost = readData(path1, nodeNum)
192     Nodes=[]
193     Arcs ={}
194     for i in range(nodeNum):
195         Nodes.append(i)
196
197     for j in range(nodeNum):
198          for i in range(nodeNum):
199              if i != j:
200                Arcs[i, j] = cost[i][j]
201
202     Graph = nx.DiGraph()
203     for node in Nodes:
204         Graph.add_node(node, min_dis=0, previous_node=None)
205     for key in Arcs.keys():
206         Graph.add_edge(key[0], key[1], length=Arcs[key])
207     org = 418
208     des = 154
209
210     shortest_path,min_dis=Dijkstra(Graph, org, des)
211     print(shortest_path)
212     print(min_dis)
213     data.plot_route(shortest_path)
214     data.plot_nodes()
```

2.5 结果展示

案例中的 1000 个点，我们选择起点为 418, 终点为 154 号。

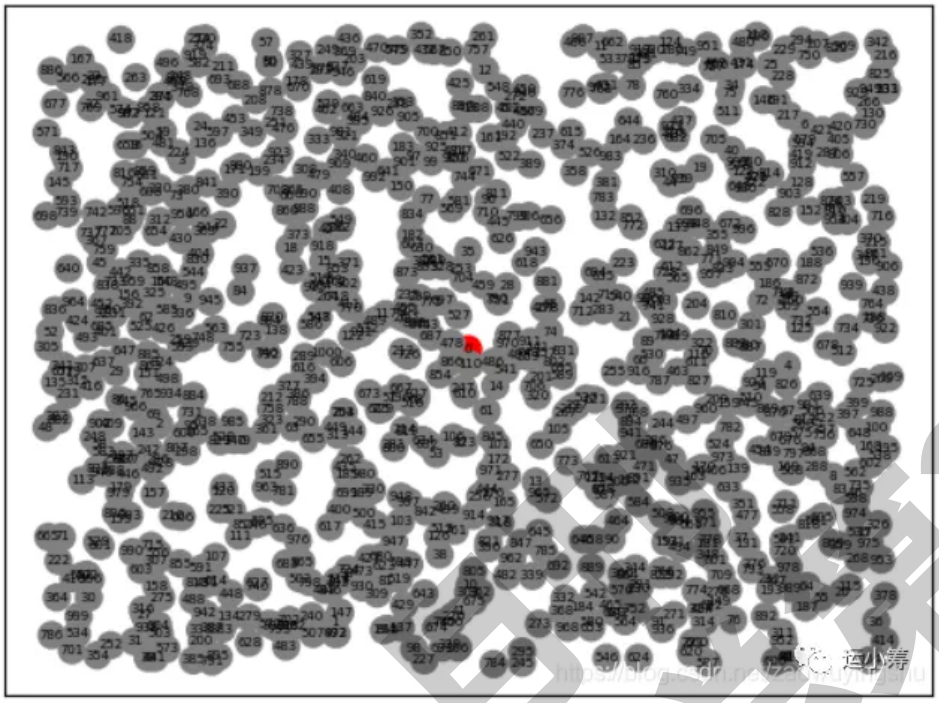


Figure 2.5.1: *1000* 个点的分布图

最短路如下：

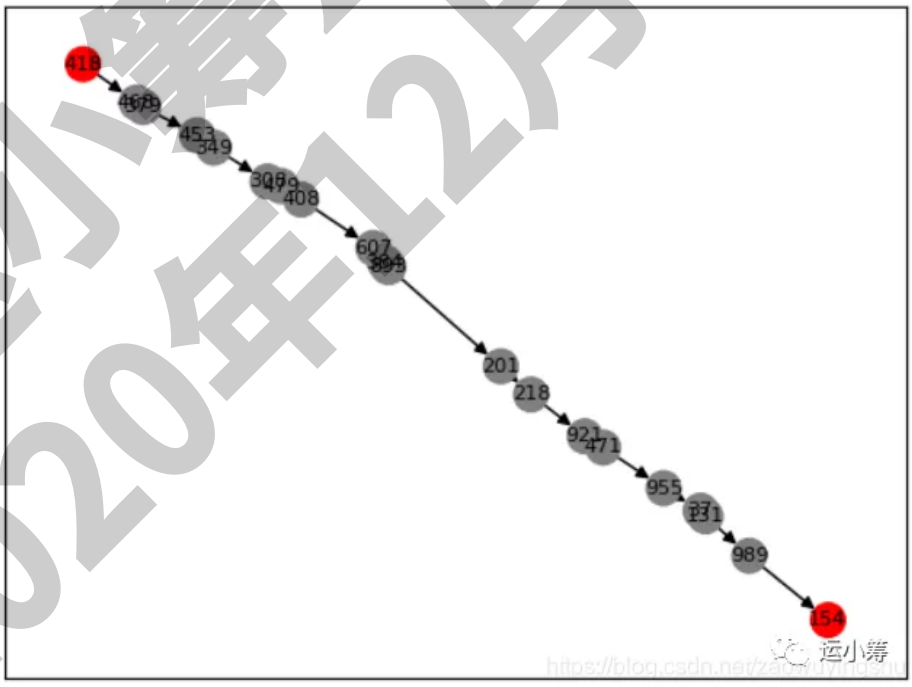


Figure 2.5.2: *1000* 个点的算例的最短路径

结果如下

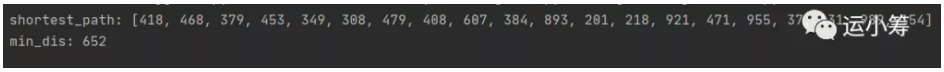


Figure 2.5.3: *1000* 个点的算例的最短路径：结果

感谢阅读！

Chapter 3

2020-12-16: 手把手教你用 Python 实现动态规划 Labeling 算法求解 SPPRC 问题

夏旸，清华大学，工业工程系/深圳国际研究生院 (硕士在读)
王基光，清华大学，清华伯克利深圳学院（硕士在读）
刘兴禄，清华大学，清华伯克利深圳学院（博士在读）

本文中的课件来自清华大学深圳国际研究生院，物流与交通学部戚铭尧教授《物流地理信息系统》课程。

3.1 SPPRC 问题

带资源约束的最短路径问题（shortest path problem with resource constraints）是一个众所周知的 NP-Hard 问题。除了作为网络问题直接应用外，SPPRC 还用作列生成解决方案方法的基础，用于解决车辆路径规划问题和人员排班问题等。　　考虑一个有向图 $G = (N, A)$，$N = \{v_1, v_2, ..., v_i, ..., v_n\}$ 表示节点的集合，并且 $A = \{(i, j)|v_i \in N, v_j \in N, i \neq j\}$ 表示弧的集合。对于每一段弧 $(i, j) \in A$ 都有一个非负的权重 (vrptw 问题中可能为负，需要作特殊处理)，c_{ij} 和 t_{ij}，表示通过这段弧的成本和资源消耗。　　SPPRC 问题包括找到从起始节点 $v_s \in N$ 到结束节点 $v_t \in N$ 的一条路径 P，使该路径的总成本最小化，但不超过最大资源消耗 T。即使只存在一种资源，SPPRC 也是一个 NP-Hard。

3.2 Labelling 算法

直接求解 SPPRC 问题是比较困难的，故通常采用一种伪多项式时间的动态规划算法：*labeling* 算法。通常我们考虑有 L 种资源，可能包括时间、最大装载重量、最大装载体积等。

对于每一条从起始节点 v_s 扩展到点 v_i 的路径 X_{pi} 都有一个标签 (R_i, C_i) 与之相关联。$R_i \equiv (T_i^1, T_i^2, ..., T_i^L)$ 是路径使用的每 L 个资源的数量；C_i 是 *cost*。

此外，我们还需要设置一些优超规则，称为 *dominance rule*，以帮助减少不必要的扩展。令 X_{pi} 和 X_{pi}^* 是从 v_s 扩展到 v_i 的两条不同的路径，与之关联的标签分别为 (R_i, C_i) 和 (R_i^*, C_i^*)。路径 X_{pi} 可以 *dominates* 路径 X_{pi}^* 当且仅当：

$$C_i \leq C_i^*, T_i^l \leq T_i^{l*}, \forall\, l \in (1, ..., L), (R_i, C_i) \neq (R_i^*, C_i^*)$$

下面介绍一个简单的小例子：

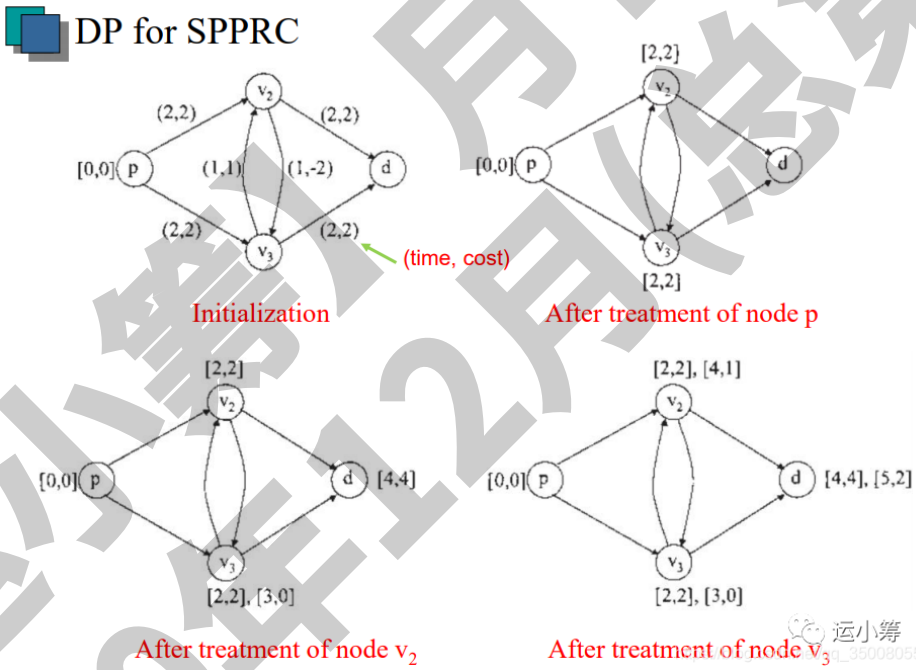


Figure 3.2.1: *results*

一个有向图如上图所示，我们可以看到每一段弧上都有一个非负的权重，分别表示通过该弧的 *time* 与 *cost*。首先初始化出发节点 p 的标签，设置为 $[0, 0]$，随后令其向可达节点 v_2 和 v_3 进行扩展，则分别设置一个新的标签 $[2, 2]$。再由 v_2 向 v_3 和 d 进行扩展，则 v_3 得到一个新的标签 $[3, 0]$，d 得到一个新的标签 $[4, 4]$，同理继续扩展到 d，生成一个新的标签 $[5, 2]$。然后回过头由 v_3 扩展到 v_2，v_2 得到一个新的标签 $[4, 1]$。我们可以发现没有继续向 d 扩展了，因为其生成的标签为 $[6, 2]$，与 d 已有的标签 $[5, 2]$ 相比满足了 *dominance rule* 的要求，

不必再扩展。

3.3 Python 编程实现

考虑这样一个有向图。

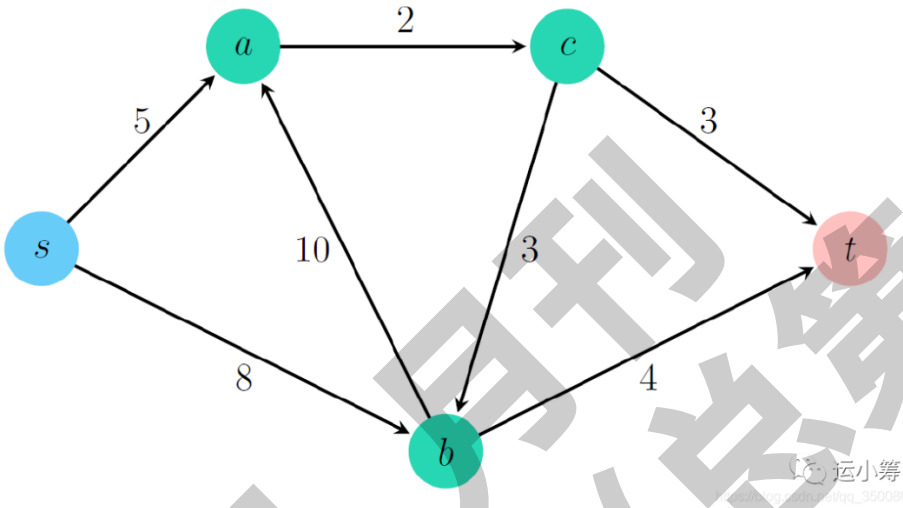


Figure 3.3.1: *results*

3.3.1 首先在 python 中对该图进行定义

Code

```
import numpy as np
import networkx as nx
from time import *
# 点中包含时间窗属性
Nodes = {'s': (0, 0)
        , '1': (6, 14)
        , '2': (9, 12)
        , '3': (8, 12)
         , 't': (9, 15)
       }
# 弧的属性包括 travel_time 与 distance
Arcs = {('s','1'): (8, 3)
       ,('s','2'): (5, 5)
       ,('s','3'): (12, 2)
       ,('1','t'): (4, 7)
       ,('2','t'): (2, 6)
       ,('3','t'): (4, 3)
      }

# create the directed Graph
```

```
Graph = nx.DiGraph()
cnt = 0
# add nodes into the graph
for name in Nodes.keys():
    cnt += 1
    Graph.add_node(name
                   , time_window = (Nodes[name][0], Nodes[name][1])
                   , min_dis = 0
                   )
# add edges into the graph
for key in Arcs.keys():
    Graph.add_edge(key[0], key[1]
                   , travel_time = Arcs[key][0]
                   , length = Arcs[key][1]
                   )
```

3.3.2 创建 Label 类及 SPPRC 相关函数

Code

```
class Label:
    path = []
    travel_time = 0
    distance = 0

# dominance rule
def dominate(Q, Path_set):
    QCopy = copy.deepcopy(Q)
    PathsCopy = copy.deepcopy(Path_set)

    # dominate Q
    for label in QCopy:
        for another_label in Q:
            if (label.path[-1] == another_label.path[
                -1] and label.time < another_label.time and label.dis < another_label.dis):
                Q.remove(another_label)
                print('dominated path (Q) : ', another_label.path)

    # dominate Paths
    for key_1 in PathsCopy.keys():
        for key_2 in PathsCopy.keys():
            if (PathsCopy[key_1].path[-1] == PathsCopy[key_2].path[-1]
                and PathsCopy[key_1].travel_time < PathsCopy[key_2].travel_time
                and PathsCopy[key_1].length < PathsCopy[key_2].length
                and (key_2 in Path_set.keys())):
                Path_set.pop(key_2)
                print('dominated path (P) : ', PathsCopy[key_1].path)

    return Q, Path_set

```

```python
# labeling algorithm
def Labeling_SPPRC(Graph, org, des):
    # initial Q
    Q = []
    Path_set = {}

    # creat initial label
    label = Label()
    label.path = [org]
    label.travel_time = 0
    label.distance = 0
    Q.append(label)

    count = 0

    while(len(Q) > 0):
        count += 1
        current_path = Q.pop()

        # extend the current label
        last_node  = current_path.path[-1]
        for child in Graph.successors(last_node):
            extended_path = copy.deepcopy(current_path)
            arc = (last_node, child)

            # check the feasibility
            arrive_time = current_path.travel_time + Graph.edges[arc]['travel_time']
            time_window = Graph.nodes[child]['time_window']
            if(arrive_time >= time_window[0] and arrive_time <= time_window[1] and last_node != des):
                extended_path.path.append(child)
                extended_path.travel_time += Graph.edges[arc]['travel_time']
                extended_path.distance += Graph.edges[arc]['length']
                Q.append(extended_path)

    Path_set[count] = current_path
    # 调用 dominance rule
    Q, Path_set = dominance(Q, Path_set)

# filtering Paths, only keep feasible solutions
    Path_set_copy = copy.deepcopy(Path_set)
    for key in Path_set_copy.keys():
        if(Path_set[key].path[-1] != des):
            Path_set.pop(key)

    # choose optimal solution
    opt_path = {}
    min_dis = 1e6
    for key in Path_set.keys():
        if(Path_set[key].distance < min_dis):
            min_dis = Path_set[key].distance
```

```
81              opt_path[1] = Path_set[key]
82
83      return Graph, Q, Path_set, opt_path
```

3.3.3 调用 labeling 算法计算

Code

```
1  org = 's'
2  des = 't'
3  begin_time = time()
4  Graph, Q, Path_set, opt_path = Labeling_SPPRC(Graph, org, des)
5  end_time = time()
6  print(" 计算时间： ", end_time - begin_time)
7  print('optimal path : ', opt_path[1].path )
8  print('optimal path (distance): ', opt_path[1].distance)
9  print('optimal path (time): ', opt_path[1].travel_time)
```

最终运行结果如图所示：

```
计算时间：  0.0010027885437011719
optimal path :  ['s', '1', 't']
optimal path (distance):  10
optimal path (time):  12
```

运小筹

Figure 3.3.2: *results*

我们也可以尝试计算更多的点，并且加入 *load capacity* 的约束，对 *Solomon c*101.*txt* 算例的计算结果为：

```
计算时间：  0.981677770614624
optimal path :  ['s', '5', '13', '20', '24', '32', '33', '42', '55', '86', 't']
optimal path (distance):  207.37820490654735
optimal path (load):  170.0
optimal path (time):  207.37820490654735
```

运小筹

Figure 3.3.3: *results*

可视化结果：

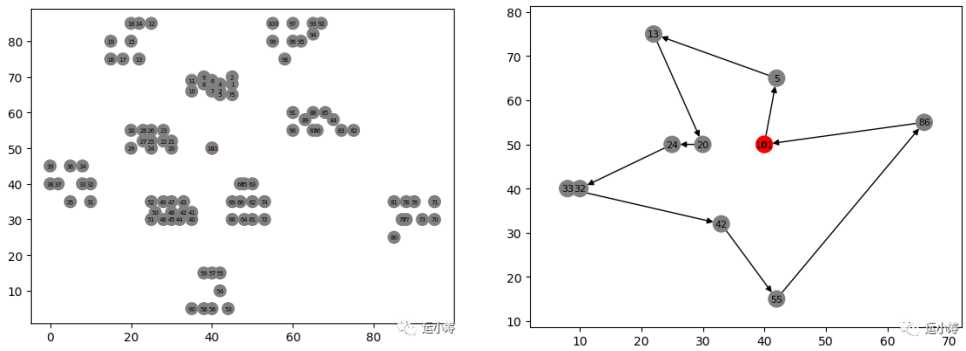


Figure 3.3.4: *SPPRC* 的解：*C101*

对 *Solomon* c1_10_1.*TXT* 算例取 300 个客户点的计算结果为：

```
计算时间:  51.89699983596802
optimal path :  ['s', '6', '17', '20', '25', '41', '55', '73', '156', 't']
optimal path (distance):  1613.8229573371548
optimal path (load):  110.0
optimal path (time):  1613.8229573371548
```

Figure 3.3.5: *results*

至于 500 个点往上，我的电脑就跑不出来啦。。。
感谢大家的阅读。

Chapter 4

2020-12-19：运筹学与管理科学大揭秘—TOP 期刊主编及研究方向一览

周鹏翔，清华大学，清华伯克利深圳学院（硕士在读）
刘兴禄，清华大学，清华伯克利深圳学院（博士在读）

4.1 UTD 期刊列表

德克萨斯大学达拉斯分校 (The University of Texas at Dallas, UTD) 的纳文·金达尔管理学院创建了一个数据库，用于跟踪 24 种主要商业期刊上的出版物，以下简称 UTD24。该数据库包含 1990 年以来在这些期刊上发表的论文和作者单位信息，用于提供 1990 年以来前 100 名商学院排名。本文介绍 UTD24 管理科学领域的期刊主编以及研究方向，并通过介绍 Operation Research 期刊下各领域的主编和研究方向跟运小筹的小伙伴们一起了解运筹学的理论和应用领域。

下面为 24 本期刊的列表。

UTD24 JOURNAL LIST

学科	Name	Since
会计	The Accounting Review	1990
	Journal of Accounting and Economics	1990
	Journal of Accounting Research	1990
金融	Journal of Finance	1990
	Journal of Financial Economics	1990
	The Review of Financial Studies	1990
信息	Information Systems Research	1990
	Journal of Computing	1990
	MIS Quarterly	1990
营销	Journal of Consumer Research	1990
	Journal of Marketing	1990
	Journal of Marketing Research	1990
	Marketing Science	1990
管科	Management Science	1990
	Operations Research	1990
	Journal of Operations Management	1990
	Manufacturing and Service Operations Management	1990
	Production and Operations Management	1990
工商	Academy of Management Journal	1990
	Academy of Management Review	1990
	Administrative Science Quarterly	1990
	Organization Science	1990
	Journal of International Business Studies	1990
	Strategic Management Journal	1990

Figure 4.1.1: *UTD 24(图片来源于百度文库)*

4.2 UTD24-管理科学领域期刊

1. **Management Science** (MS)
2. **Operations Research**(OR)

3. **Journal of Operations Management**(JOM)
4. **Manufacturing and service Operations Management** (MSOM)
5. **Production and Operations Management**(POM)

事实上，对于从事运筹学研究的研究者来讲，有些研究也经常发在信管领域的 **Journal of Computing**(JOC) 上。当论文的计算量较大，数值实验非常充足的时候，有些大牛们也是会尝试一下 JOC 的。因此，对于管科领域来讲，实际上是有 $5+1=6$ 本可选的 UTD 期刊的，占据了 $\frac{1}{4}$ 的席位呢。

下面是这些期刊的一些主编的信息 (本文作者整理)。

Top Journal in the Field of Operation Research

Journal	Editor	Postion	Research Interests
Management Science	David Simchi-Levi	Massachusetts Institute of Technology Institute for Data, Systems, and Society Department of Civil and Environmental Engineering dslevi@mit.edu	Operations Research and Statistical Learning Transportation and logistics systems analysis Supply chain management Revenue and yield management Optimization based decision support systems Flexibility and Risk Management
Operation Research	John Birge	The University of Chicago Booth School of Business john.birge@chicagobooth.edu	Mathematical modeling of systems under uncertainty Stochastic programming Large-scale optimization
Journal of Operations Management	Suzanne de Treville	University of Lausanne Faculty of Business and Economics (HEC) Suzanne.DeTreville@unil.ch	Competitive manufacturing in a high-cost environment Supply chain management Lead time reduction Lean production Process consistency improvement Health care operations Queuing theory-based mathematical modeling of operations
Manufacturing & Service Operations Management	Christopher S. Tang	University of California, Los Angeles UCLA Anderson School of Management chris.tang@anderson.ucla.edu	Optimization models Economic models Supply chain management Innovative operations Socially responsible operations Economic development in emerging markets
Production and Operations Management Society	Chelliah Sriskandarajah	Texas A&M University Mays Business School chelliah@mays.tamu.edu	Supply chain management Logistics Production planning and scheduling Performance evaluation of production/service systems

Figure 4.2.1: 管科 5 本 *UTD* 期刊的主编

2021 年，MSOM 的主编有所变动，因此 2020 年最新的信息主编信息为

Top Journal in the Field of Operation Research

Journal	Editor	Postion	Research Interests
Management Science	David Simchi-Levi	Massachusetts Institute of Technology Institute for Data, Systems, and Society Department of Civil and Environmental Engineering dslevi@mit.edu	Operations Research and Statistical Learning Transportation and logistics systems analysis Supply chain management Revenue and yield management Optimization based decision support systems Flexibility and Risk Management
Operation Research	John Birge	The University of Chicago Booth School of Business john.birge@chicagobooth.edu	Mathematical modeling of systems under uncertainty Stochastic programming Large-scale optimization
Journal of Operations Management	Suzanne de Treville	University of Lausanne Faculty of Business and Economics (HEC) Suzanne.DeTreville@unil.ch	Competitive manufacturing in a high-cost environment Supply chain management Lead time reduction Lean production Process consistency improvement Health care operations Queuing theory-based mathematical modeling of operations
Manufacturing & Service Operations Management	Georgia Perakis	MIT Sloan School of Management georgiap@mit.edu	Mathematical Programming and Machine Learning Loss of Efficiency (Price of Anarchy) Transportation Energy and Sustainability Healthcare Retail Pricing, Revenue Management, Inventory Control and Supply Chains
Production and Operations Management Society	Chelliah Sriskandarajah	Texas A&M University Mays Business School chelliah@mays.tamu.edu	Supply chain management Logistics Production planning and scheduling Performance evaluation of production/service systems

Figure 4.2.2: 管科 5 本 *UTD* 期刊的主编：*2021* 年

以及 OR 的一些领域主编的信息

Operation Research-Area Editors' Introduction

Area	Editor	Postion	Research Interests
Decision Analysis	James E. Smith	Tuck School of Business james.e.smith@tuck.dartmouth.edu	Decision analysis Management science Operations research
Environment, Energy, and Sustainability	Erica Plambeck	Stanford University Graduate School of Business elp@stanford.edu	Environmental sustainability Manufacturing operations Supply chain management
	Daniel Ralph	Cambridge University Judge Business School d.ralph@jbs.cam.ac.uk	Management of systemic and emerging risks Business decision making Risk aversion in electricity markets Methods and models for optimization problems and equilibrium systems.
Financial Engineering	Paul Glasserman	Columbia Business School pg20@columbia.edu	Asset Management/Asset Pricing Financial Instruments Risk Management Statistics
	Steven Kou	Boston University Questrom School of Business kou@bu.edu	FinTech Quantitative Finance Applied Probability Statistics.
Machine Learning and Data Science	Alexandre Belloni	Duke University Fuqua School of Business abn5@duke.edu	Econometrics and Statistics (high-dimensional models, model selection, quantiles) Mathematical Programming (continuous optimization) Mechanism Design (multivariate, dynamics, externalities) Probabilistic Methods (concentration of measure, empirical processes) Applications

Figure 4.2.3: *Operations Research* 部分领域主编的信息 *(1)*

Revenue Management and Market Analytics	Ramesh Johari	Stanford University Department of Management Science and Engineering rjohari@stanford.edu	Design Economic analysis Operation of online platforms Statistical and machine learning techniques
	Gustavo Vulcano	New York University Stern School of Business gv18@stern.nyu.edu	Customer choice and strategic behavioral models Data driven methods for pricing and revenue management Computational methods for network revenue management
Military and Homeland Security	Moshe Kress	University of Texas Operations Research Department Naval Postgraduate School mkress@nps.edu	Military and defense intelligence Counter-insurgency modeling Homeland security problems Military logistics.
Operations and Supply Chains	Dan Adelman	The University of Chicago Booth School of Business	Foundations of the operations research field The link between operational performance metrics and financial performance The electricity smart grid Gasoline supply chains Software-release planning Healthcare delivery
	Tava Olsen	The University of Auckland Business School t.olsen@auckland.ac.nz	Supply chain management, pricing, and inventory control Stochastic models of production, service, and health systems Stochastic games, stochastic control
Optimization	Daniel Kuhn	EPFL College of Management of Technology daniel.kuhn@epfl.ch	Decision-making under uncertainty Stochastic programming and robust optimization Data-driven optimization
	Sam Burer	University of Iowa Tippie College of Business samuel-burer@uiowa.edu	Analytics, operations research, management sciences Discrete and continuous optimization Convex, copositive, semidefinite, and nonlinear optimization Decision making and optimization under uncertainty

Figure 4.2.4: *Operations Research* 部分领域主编的信息 *(2)*

OR Practice	Andres Weintraub	University of Chile Department of Industrial Engineering aweintra@dii.uchile.cl	System Analysis and Modeling in Forestry Mining Problems of Transportation and Logistics Applied Integer Programming Methods in Decision Making Operations Management
Policy Modeling and Public Sector OR	Ozlem Ergun	Northeastern University College of Engineering o.ergun@northeastern.edu	Design and management of large-scale networks Supply chain design and resilience Collaboration and crowdsourcing in logistics Humanitarian logistics
Simulation	Hong Liu	Fudan University School of Management hong_liu@fudan.edu.cn	Modeling & Optimization of Stochastic Simulation Financial Engineering & Risk Management Business Analytics
Stochastic Models	Itai Gurvich	Cornell School of Operations Research and Information Engineering and Cornell Tech gurvich@cornell.edu	Applied Probability Queueing Theory Control Theory Service Operations Healthcare Operations
	Amy Ward	University of Chicago Booth School of Business amy.ward@chicagobooth.edu	Service operations management Development and analysis of stochastic process models
Transportation	Anton J. Kleywegt	School of Industrial and Systems Engineering Georgia Institute of Technology anton@isye.gatech.edu	Optimization and Stochastic Modeling applied to Transportation, Distribution and Logistics Vehicle Routing and Scheduling Distribution Operations, Distribution Network Design Yield Management, Terminal Design and Operations Logistics Planning and Control, Multimodal Transportation Intelligent Transportation Systems

Figure 4.2.5: *Operations Research* 部分领域主编的信息 *(3)*

Chapter 5

2020-12-25：手把手教你用 Python 实现 Dijkstra 算法（伪代码 +Python 实现）

张祎，清华大学，清华伯克利深圳学院（硕士在读）
刘兴禄，清华大学，清华伯克利深圳学院（博士在读）

5.1 最短路问题 Shortest Path Problem

之前的文章中我们已经用迪杰斯特拉算法求解过 Solomon 算例了，今天是用该算法求解有 5 个点连通图的最短路径。大家可以看完这篇再回顾一下之前的文章。https://mp.weixin.qq.com/s?__biz=MzI3NTI2NzcxOA==&mid=2247484466&idx=1&sn=5958d0154b30ef46ca81ebd216d26chksm=eb062ddbdc71a4cd47a4c6edb153d48057ba4d16bc096bedee83ae7e7a12b656d1d06b420950&scene=21#wechat_redirect

如图所示的连通图，以 s 为起点，t 为终点，求一条 s 到 t 的最短路径。

5.2 Dijkstra 算法

Dijkstra 算法采用的是贪心算法策略：

1. 起点临时标号为 0，其余所有点临时标号为 ∞
2. 将所有与该**永久标号**相连的点的**临时标号**更新为**临时标号加边的权重（原临时标号为 ∞ 的点更新为边的权重）**

3. 在所有**临时标号**中选取**最小标号**变为**永久标号**, 并记录该点与上一个被永久标号的点相连
4. 重复步骤 2 和 3，直到所有点都被**永久标号**，算法结束

局限性：该算法要求边不能有负权重

优势：可以求出起点到图中任一点的最短路径.

5.2.1 Dijkstra 算法解决最短路问题（SPP）伪代码

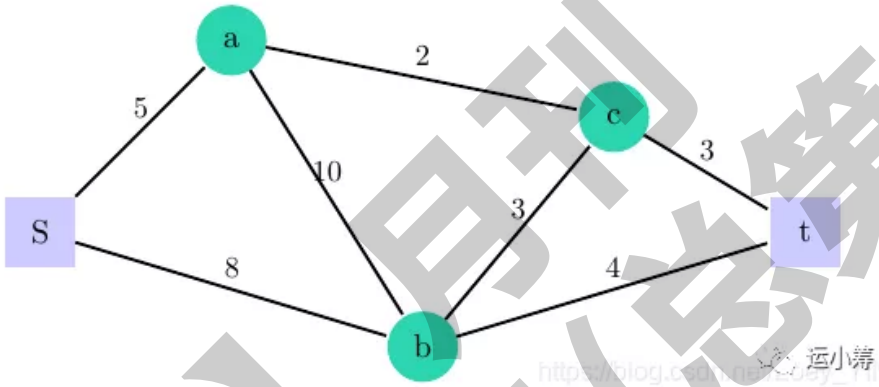


Figure 5.2.1: 例子网络

5.2.2 Dijkstra 算法解决最短路问题（SPP）Python 实现

5.2.3 5 点连通图的最短路径

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import copy
import re
import math
```

输入模块

代码语言表示文首 5 点连通图

Code

```
Nodes = ['s', 'a', 'b', 'c', 't']

Arecs = {('s', 'a'):5, ('s', 'b'):8, ('a', 'c'):2, ('b','a'):10, ('c','b'):3, ('b','t'):4,
 ↪  ('c','t'):3}
```

给图中的点和弧添加属性

Code

```
Graph = nx.DiGraph()
for node in Nodes:
    Graph.add_node(node, min_dis = 0, previous_node = None)
for key in Arecs.keys():
    Graph.add_edge(key[0],key[1],length = Arecs[key])
```

定义用 Dijkstra 算法求解最短路的函数

Code

```
def Dijkstra(Graph , org ,des):
    # 定义 T
    T = list(Graph.nodes())

    # 给各点临时标号
    for node in T:
        if (node == org) :
            Graph.nodes[node]['min_dis'] = 0
        else:
            Graph.nodes[node]['min_dis'] = np.inf
    current_node = org

    while (len(T) > 0):
        min_dis = np.inf

        # 更新永久标号的点
        for node in T:
            if (Graph.nodes[node]['min_dis'] < min_dis):
                current_node = node
                min_dis=Graph.nodes[node]['min_dis']
        if (current_node != None):
            T.remove(current_node)

        # 更新永久标号点相邻点的临时标号
        for child in Graph.successors(current_node):
            arc = (current_node, child)
            dis_t = Graph.nodes[current_node]['min_dis'] + Graph.edges[arc]['length']
            if (dis_t < Graph.nodes[child]['min_dis']):
                Graph.nodes[child]['min_dis'] = dis_t
                Graph.nodes[child]['previous_node'] = current_node

    # 确定最短路径和最短距离
    min_dis = Graph.nodes[des]['min_dis']
    current_node = des
    shortest_path = [current_node]
    while (current_node != org):
        current_node=Graph.nodes[current_node]['previous_node']
```

```
38          shortest_path.insert(0, current_node)
39      return Graph, shortest_path, min_dis
```

调用函数求解文首的最短路径问题

Code

```
org = 's'
des = 't'
Graph, shortest_path, min_dis = Dijkstra(Graph, org, des)
```

结果输出如下：

Code

```
(['s', 'a', 'c', 't'], 10)
```

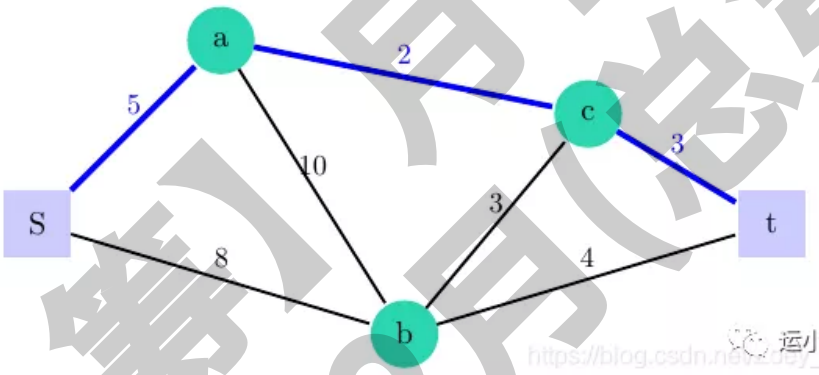


Figure 5.2.2: 最优解

Chapter 6

2020-12-28：运筹学与管理科学 TOP 期刊揭秘—Service Science

修宇璇，清华大学，清华伯克利深圳学院（博士在读）

6.1 Service Science

- INFORMS旗下服务科学领域的旗舰期刊
- SSCI检索
  2019年影响因子为1.623
  2018年影响因子为1.885
  5年影响因子为1.932
- 《FMS管理科学高质量期刊推荐列表》运营管理领域B类期刊
  (在FMS列表中，运营管理领域仅有三个A类期刊，分别是:
  ※ Journal of Operations Management
  ※ Manufacturing & Service Operations Management
  ※ Production and Operations Management
  三者均为UTD24期刊)
- 季刊，每个季度月末出版
- 期刊主页:
  https://pubsonline.informs.org/journal/serv
- ISSN:
  2164-3962
  2164-3970 (Online)

Figure 6.1.1: *Service Science*：期刊概述

Service Science 是 INFORMS 出版的服务科学领域的旗舰期刊，发表与服务有关的所有主题的原创性论文。根据 Service Science 前任主编 Paul P. Maglio 的定义，“服务就是人与人之间共同合作，通过技术创造价值的过程”。在这种情况下，服务科学是对服务和服务系统的研究。它可能涉及到一系列学科的方法和理论，包括运筹学、工业工程、市场营销、计算机科学、心理学、信息系统、设计等。Service Science 期刊重点关注服务科学的理论和实践研究，希望接收从多个学科的角度、方法或概念出发，以服务和服务系统为研究对象，并与新兴的服务科学文献、概念、方法和理论相联系的论文。

除了定期出版的 Regular Issue 之外，Service Science 还会针对服务科学领域的热点问题举办 Special Issue。目前正在征稿的 Special Issue 是“交通赋能的城市服务创新”（Innovation in Transportation-Enabled Urban Services）。本次 Special Issue 重点关注基于城市交通和物流的新型城市服务。这种新型城市服务既包括出行服务本身的创新，例如“出行即服务”（Mobility as a Service，MaaS），也包括交通和物流技术进步带来的新型服务模式，例如食品和生鲜杂货配送。本期 Special Issue 旨在利用服务科学的研究范式提高我们对这些服务的规划、运营和管理的理解。期待的文章主题包括建模、经济/计量分析以及基于优化/数据驱动方法的决策。

本次Special Issue关注的主题如下：

Stream 1: Innovation in traditional urban transportation/logistics systems. Potential topics include (but are not limited to):

- Shared transportation and logistics
- Mobility-as-a-service
- On-demand transportation for underserved/underrepresented groups
- Flexible school bus services
- Smart automated parking
- Last-mile delivery
- Social implications of these innovations (e.g., environment, equity, safety, energy)

Stream 2: Innovation in transportation-enabled urban services. Potential topics include (but are not limited to):

- Food and grocery delivery
- Local freelancing services
- Seamless e-commerce
- On-demand healthcare enabled by transportation
- Personal shopper services
- Urban patrol and emergency responses
- Social implications of these innovations (e.g., environment, equity, safety, energy)

Figure 6.1.2: *Special Issue (1)*

本次Special Issue的关键时间节点:

- Submission of full paper: 30 June 2021
- Feedback from editorial team: 30 September 2021
- Submission of revised manuscript: 31 January 2022
- Final decision (subject to minor revisions): 30 April 2022

Figure 6.1.3: *Special Issue (2)*

本次 Special Issue 的详情请见:https://pubsonline.informs.org/page/serv/calls-for-papers

参考文献

[1] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[2] Wayne L Winston and Jeffrey B Goldberg. *Operations research: applications and algorithms*, volume 3. Thomson/Brooks/Cole Belmont^ eCalif Calif, 2004.