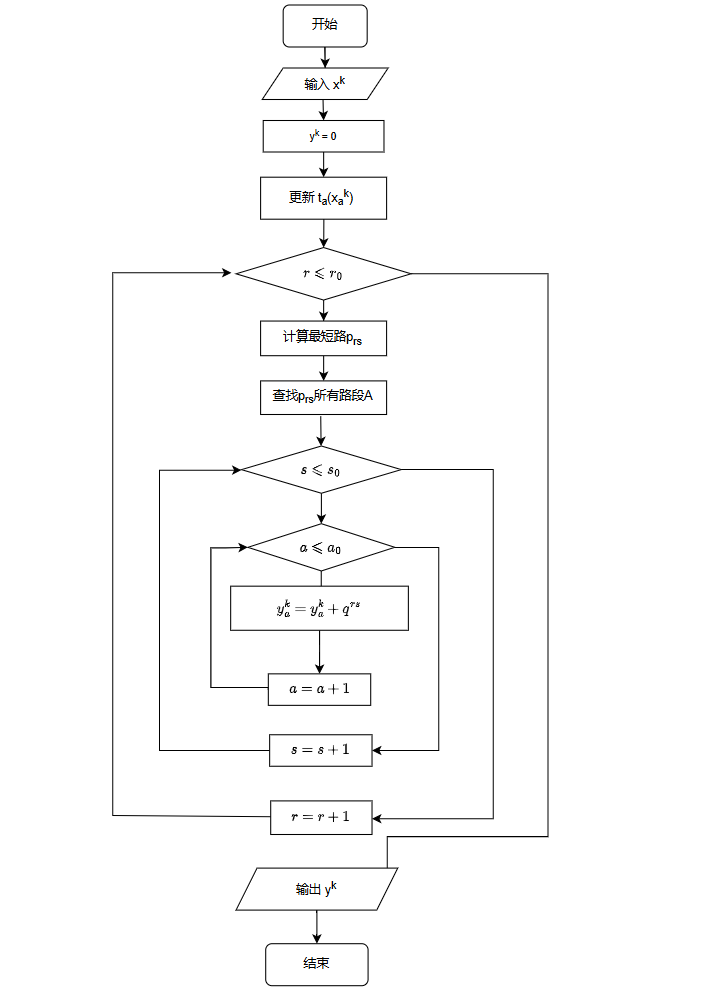
报告

1. 算法的计算流程图
2. 算法一：全有全无交通分配



输入: 路段流量

1：令

2：依据路段流量 更新所有路段 的走行时间

3：for each do

4： 最短路算法求 到所有目的地 的最短路径 ，并计算路段集合

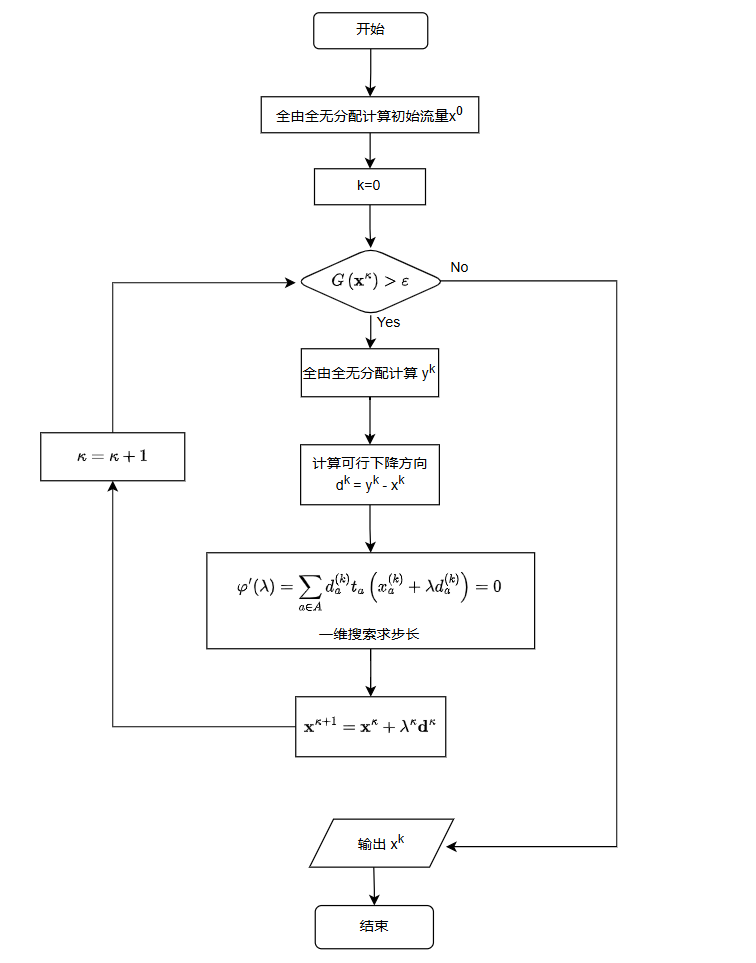
5： for each do

6： for each do

7： 置

输出: 全有全无交通分配得到的路段流量

1. 算法二：Frank-wolf算法求解UE交通分配



输入: 精度

1：基于所有路段的自由流走行时间，做全有全无交通分配，得到初始的路段流量 置迭代次数 0

2: while do

3: 以 为输入，做全有全无交通分配，得到路段流量

4: 计算可行的下降方向

5: 根据 进行一维搜索得到

6: 更新路段流量

7: 置

输出:

其中，为间隙函数，用于评价UE交通分配问题解的精度。

为对的 需求， 为流量下对间的最小出行费用。

1. 程序代码及说明
2. *#1：基本类定义（路段，节点，OD对）*
3. import os
4. class node():*#节点类，包含每个节点的坐标信息*
5. def \_\_init\_\_(self):
6. self.ID=None
7. self.origin=-1
8. self.innode=[]*#进入节点路段集合*
9. self.outnode=[]*#离开节点路段集合*
10. class link(): *#路段类，包含路段的起讫点、自由流行驶时间、容量、BRP路阻函数的参数*
11. def \_\_init\_\_(self):
12. self.ID=None
13. self.O\_link=None*#起点*
14. self.D\_link=None*#终点*
15. self.FFT=None
16. self.Traveltime=None
17. self.Capacity=None
18. self.B=0.15  *#SiouxFalls网络中路网B与Power系数值均相等，方便期间，统一设置一个初始值*
19. self.Power=4
20. class OD():*#OD类，记录每一个OD需求信息，包含起点、重点、流量需求*
21. def \_\_init\_\_(self):
22. self.ID=None
23. self.origin\_node=None
24. self.Destination=[]
25. self.odlinks\_demand=[]
26. *#2：定义路网核心函数（路网信息导入、UE分配计算等）*
27. class network():*#网络类，核心函数*
28. def \_\_init\_\_(self):
29. self.Nodes=[]*#网络包含节点集合*
30. self.Links=[]*#路段集合*
31. self.Origins=[]*#起点集合，即流量发点*
32. self.num\_Nodes=0*#节点个数*
33. self.num\_Links=0*#路段个数*
34. self.num\_Origins=0*#起点个数*
35. self.Linkflows=[]*#路段流量集合*
36. self.Linktimes=[]*#路段行驶时间集合*
37. self.max\_err=0.001*#UE最大误差*
38. self.err=1*#初始UE误差*
39. self.Djpathcost=[]*#最短路阻抗集合（Dijkstra算法求得）*
40. self.Djpath=[]*#最短路集合*
41. def read\_nodes(self,path):*#读取节点信息*
42. f=open(path)
43. file=f.readlines()
44. for i in file:
45. row=i.strip('\n').split('\t')
46. newnode=node()
47. newnode.ID=int(row[0])
48. self.Nodes.append(newnode)
49. f.close()
50. def read\_link(self,path):*#读取路段信息*
51. f=open(path)
52. file=f.readlines()
53. linkid=1
54. for i in file:
55. row=i.strip('\n').split('\t')
56. newlink=link()
57. newlink.ID=linkid
58. newlink.O\_link=self.Nodes[int(row[0])-1]
59. newlink.D\_link=self.Nodes[int(row[1])-1]
60. newlink.FFT=int(row[2])
61. newlink.Capacity=float(row[3])
62. newlink.O\_link.outnode.append(newlink.ID)
63. newlink.D\_link.innode.append(newlink.ID)
64. linkid+=1
65. self.Links.append(newlink)
66. f.close()
67. def read\_od(self,path):*#读取OD信息*
68. f=open(path)
69. file=f.readlines()
70. linkid=1
71. for i in file:
72. row=i.strip('\n').split('\t')
73. newnode=self.Nodes[int(row[0])-1]
74. if newnode.origin==-1:
75. neworigin=OD()
76. self.num\_Origins+=1
77. neworigin.ID=self.num\_Origins
78. neworigin.origin\_node=newnode
79. self.Nodes[int(row[0])-1].origin=neworigin.ID
80. self.Origins.append(neworigin)
81. else:
82. neworigin=self.Origins[newnode.origin-1]
84. self.Origins[newnode.origin-1].Destination.append(self.Nodes[int(row[1])-1].ID)
85. self.Origins[newnode.origin-1].odlinks\_demand.append(float(row[2]))
87. f.close()
88. def Dijkstra(self,start,end):*#Dijkstra求解最短路（也叫标号法）*
89. startpos=0
90. endpos=1
91. path=[]
92. checkpath=[None for i in range(len(self.Nodes))]
93. boolcheckpath=[]
94. self.Djpathcost=[]
95. self.Djpath=[None for i in range(len(self.Nodes))]
96. bscanStatus=[None for i in range(len(self.Nodes))]
97. for i in range(len(self.Nodes)):
98. self.Djpath.append(-1)
99. self.Djpathcost.append(9999999)*#标号法初始路阻最大*
100. boolcheckpath.append(False)
102. self.Djpathcost[start-1]=0
103. checkpath[0]=start-1
104. while startpos!=endpos:
105. if startpos>=len(self.Nodes):
106. startpos=0
107. i=checkpath[startpos]
108. startpos+=1
109. newnode=self.Nodes[i]
110. for j in range(len(newnode.outnode)):
111. newlink=self.Links[newnode.outnode[j]-1]
112. k=newlink.D\_link.ID
113. tt=newlink.Traveltime
114. if self.Djpathcost[k-1]>self.Djpathcost[i]+tt:
115. self.Djpathcost[k-1]=self.Djpathcost[i]+tt
116. self.Djpath[k-1]=i
117. if endpos>=len(self.Nodes):
118. endpos=0
119. checkpath[endpos]=k-1
120. endpos+=1
121. bscanStatus[k-1]=True
122. return self.Djpathcost[end-1]
124. def Dijkstra\_path(self,start,end):*#记录最短路径，与上述函数基本相同，输出结果不同，为了后面调用方便，写了两次*
125. startpos=0
126. endpos=1
127. path=[]
128. checkpath=[None for i in range(len(self.Nodes))]
129. boolcheckpath=[]
130. self.Djpathcost=[]
131. self.Djpath=[None for i in range(len(self.Nodes))]
132. bscanStatus=[None for i in range(len(self.Nodes))]
133. for i in range(len(self.Nodes)):
134. self.Djpath.append(-1)
135. self.Djpathcost.append(9999999)
136. boolcheckpath.append(False)
137. self.Djpathcost[start-1]=0
138. checkpath[0]=start-1
139. while startpos!=endpos:
140. if startpos>=len(self.Nodes):
141. startpos=0
142. i=checkpath[startpos]
143. startpos+=1
144. newnode=self.Nodes[i]
145. for j in range(len(newnode.outnode)):
146. newlink=self.Links[newnode.outnode[j]-1]
147. k=newlink.D\_link.ID
148. tt=newlink.Traveltime
149. if self.Djpathcost[k-1]>self.Djpathcost[i]+tt:
150. self.Djpathcost[k-1]=self.Djpathcost[i]+tt
151. self.Djpath[k-1]=i
152. if endpos>=len(self.Nodes):
153. endpos=0
154. checkpath[endpos]=k-1
155. endpos+=1
156. bscanStatus[k-1]=True
157. Djpathlink=[]
158. point\_out=end-1
159. while True:
160. i=0
161. point\_in=self.Djpath[point\_out]
162. for j in range(len(self.Links)):
163. newlink=self.Links[j]
164. if point\_in==newlink.O\_link.ID-1 and point\_out==newlink.D\_link.ID-1:
165. Djpathlink.insert(0,newlink.ID)
166. point\_out=point\_in
167. i+=1
168. if point\_in==start-1:
169. break
170. return Djpathlink
172. def all\_none(self):*#全有全无分配函数*
173. all\_none\_linkflow=[0 for i in range(len(self.Links))]
174. for i in range(len(self.Links)):
175. self.Links[i].Traveltime=self.Links[i].FFT\*(1+self.Links[i].B\*(self.Linkflows[i]/self.Links[i].Capacity)\*\*self.Links[i].Power)*#更新路段行驶时间*
176. all\_none\_linkflow[i]=0
177. for i in range(len(self.Origins)):
178. o\_node=self.Origins[i].origin\_node.ID
179. for j in range(len(self.Origins[i].Destination)):
180. d\_node=self.Origins[i].Destination[j]
181. demand=self.Origins[i].odlinks\_demand[j]
182. Djpathlink=self.Dijkstra\_path(o\_node,d\_node)*#找最短路*
183. for index in Djpathlink:
184. all\_none\_linkflow[index-1]+=demand*#将流量加载到最短路上*
185. return all\_none\_linkflow
187. def getUEerr(self):*#计算UE误差*
188. sum1=0
189. for i in range(len(self.Links)):
191. self.Links[i].Traveltime=self.Links[i].FFT\*(1+self.Links[i].B\*(self.Linkflows[i]/self.Links[i].Capacity)\*\*self.Links[i].Power)
192. sum1+=self.Links[i].Traveltime\*self.Linkflows[i]*#计算流量与行驶时间的乘积（UE公式中的积分项）*
194. sum2=0
195. for i in range(len(self.Origins)):
196. for j in range(len(self.Origins[i].Destination)):
197. demand=self.Origins[i].odlinks\_demand[j]
198. cost=self.Dijkstra(self.Origins[i].origin\_node.ID,self.Origins[i].Destination[j])
199. sum2+=demand\*cost*#计算需求与行驶时间的乘积*
201. return 1-sum2/sum1
202. def Optfunction(self,Descent,Lamuda):*#计算最优下行方向函数，返回值越大，下降方向越优*
203. Sum=0
204. for i in range(len(self.Links)):
205. x=self.Linkflows[i]+Lamuda\*Descent[i]
206. Sum+=Descent[i]\*(self.Links[i].FFT\*(1+self.Links[i].B\*(x/self.Links[i].Capacity)\*\*self.Links[i].Power))
207. return Sum
209. def Frank\_Wolfe(self):*#Frank-Wolfe主函数*
211. iter=0*#迭代次数*
212. self.Linkflows=[0 for i in range(len(self.Links))]
213. self.Linkflows=self.all\_none()
214. while self.err>self.max\_err:
215. oldlinkflow=self.Linkflows
216. newlinkflow=self.all\_none()
217. Descent=[]
218. for i in range(len(self.Links)):
219. Descent.append(newlinkflow[i]-self.Linkflows[i])
220. Lamuda=0
221. left=0*#二分法求最优步长*
222. right=1
223. mid=0
224. f\_left=self.Optfunction(Descent,left)
225. f\_right=self.Optfunction(Descent,right)
226. f\_mid=0
227. if f\_left\*f\_right>0:
228. if abs(f\_left)>abs(f\_right):
229. Lamuda=right
230. else:
231. Lamuda=left
233. else:
234. while right-left>self.max\_err:
235. mid=(left+right)/2
236. f\_left=self.Optfunction(Descent,left)
237. f\_right=self.Optfunction(Descent,right)
238. f\_mid=self.Optfunction(Descent, mid)
239. if f\_left\*f\_mid>0:
240. left=mid
241. else: right=mid
242. Lamuda=(left+right)/2
244. for i in range(len(self.Links)):*#更新路段流量*
245. self.Linkflows[i]+=Lamuda\*Descent[i]
246. iter+=1
247. self.err=self.getUEerr()
248. *#print(self.Linkflows)*
250. net=network()
251. *#读取路网文件*
252. net.read\_nodes('节点.csv')
253. net.read\_link('路段.csv')
254. net.read\_od('OD需求.csv')
255. net.Frank\_Wolfe()
256. print(net.Linkflows)
257. 网络初始流量
258. 迭代计算过程
259. 验证结果满足Wardrop用户均衡
260. 体会与感受