The documentation of MCMF script

1. **Script**
   1. ***Handwritten version***

'''

created by Erx

no package at all :)

using EK+SPFA to find the minimum cost maximum flow

input: follow the hint and input your number / array of numbers (separated by space)

output: a chart about the assignment arrangements

'''

class Graph:

def \_\_init\_\_(self,points):

self.costTab={} #table of costs

self.capTab={} #table of capacities

self.resCap={} #residual/left capacities

self.backPath={} #backward arcs

self.len=len(points)

self.INF=1e9

for i in points:

self.costTab.update({i:{}})

self.capTab.update({i:{}})

self.resCap.update({i:{}})

def addEdge(self,s,t,cap,cost):

self.costTab[s].update({t:cost})

self.capTab[s].update({t:cap})

self.resCap[s].update({t:cap})

def costFlow(self,s,t):

path=self.SPFA(s,t)

while path:

#print(" \n\nthe shortest path:",path)

minFlow=self.findMinFlow(path,s,t)

#print("the bottleneck: ",minFlow)

self.changeFlow(minFlow,path,s,t)

path=self.SPFA(s,t)

return self.backPath

def SPFA(self,s,t):

que=[] #spfa's que

path=[0]\*self.len #record the shortest path

dist=[self.INF]\*self.len #spfa's dist

visited=[False]\*self.len #whether the point is in spfa's que

dist[s]=0

que.append(s);visited[s]=True #the source in que

while que:

nowPoint=que[0]

que.pop(0)

visited[nowPoint]=False #take out the first element in que

for linkPoint,leftCapacity in self.resCap[nowPoint].items(): #its connected points / edges

if (leftCapacity!=0)&(dist[linkPoint]>dist[nowPoint]+self.costTab[nowPoint][linkPoint]):

dist[linkPoint]=dist[nowPoint]+self.costTab[nowPoint][linkPoint] #refresh the shorted path

path[linkPoint]=nowPoint

if not visited[linkPoint]: #add in que

que.append(linkPoint)

visited[linkPoint]=True

if dist[t]!=self.INF:

return path

else:

return False

def findMinFlow(self,path,s,t):

pathNow=t

mm=self.INF

while pathNow!=s:

pathPre=path[pathNow]

if self.resCap[pathPre][pathNow]<mm:

mm=self.resCap[pathPre][pathNow]

pathNow=pathPre

if mm!=self.INF:

return mm

else:

return -1

def changeFlow(self,minFlow,path,s,t):

pathNow = t

#print("backwards change flow path:",pathNow,end="")

while pathNow != s:

pathPre = path[pathNow]

#print("->",pathPre,end="")

self.resCap[pathPre][pathNow]-=minFlow

if pathNow not in self.backPath:

self.backPath[pathNow] = {}

if pathPre not in self.backPath[pathNow]:

self.backPath[pathNow][pathPre] = 0

self.backPath[pathNow][pathPre]+=minFlow

pathNow = pathPre

def input\_patient\_preference\_list(N):

rank=[[]]\*N

#rank: list of lists, every list's order represent a patient's prefered doctor's number, listed from most like to least like.

# e.g. "4 2 3 1" represent the patient likes doctor 4 more than doctor 2, and so on.

for i in range(N):

input\_string=input("Rank Order of Patient No.%d:" %(i+1)) #input seperated by space " "

rank[i]=input\_string.split()

rank[i]=[int(ele) for ele in rank[i]]

return rank

if \_\_name\_\_ == '\_\_main\_\_':

N=int(input("Number of Patients:"))

K=int(input("Number of Doctors:"))

C=[] #same/different capacity for each doctor

input\_string=input("Capacity of Each Doctor:") #input seperated by space " "

C=input\_string.split()

C=[int(ele) for ele in C] #turn into integers

rank=input\_patient\_preference\_list(N) #patients' preferences

points=[] #dots in the graph

for i in range(N+K+2):

points.append(i)

G=Graph(points)

for i in range(N):

G.addEdge(0,i+1,1,0) #from source to patient: capacity 1, cost 0

for i in range(N):

for j in range(K):

G.addEdge(i+1,j+N+1,1,rank[i][j]) #from patient to doctor: capacity 1, cost (preferences)

for i in range(K):

G.addEdge(i+N+1,N+K+1,C[i],0) #from doctor to terminal: capacity (doc\_cap), cost 0

s=0;t=N+K+1

outputTab=G.costFlow(s,t)

title="\nAssignments:\n Doctor"+" "\*8+"Patient(s) "

print(title)

print("-"\*(len(title)-len("\nAssignments:\n)")),end="")

for p,linkage in outputTab.items():

if (p>N)&(p<N+K+1):

print("\n No.%d "%(p-N),end="")

#print("\nDoctor No.%d"%(p-N),"will take in Patient ",end="")

for patient,choice in linkage.items():

print(" No.%d "%(patient),end="")

#print("No. %d "%(patient),end="")

* 1. ***Using Package Version***

from ortools.graph import pywrapgraph

def input\_patient\_preference\_list(N):

rank=[[]]\*N

#rank: list of lists, every list's order represent a patient's prefered doctor's number.

# e.g. "1 2 3 4" represent the patient likes doctor 1 more than doctor 2, and so on.

for i in range(N):

input\_string=input("Rank Order of Patient No.%d:" %(i+1)) #input seperated by space " "

rank[i]=input\_string.split()

rank[i]=[int(ele) for ele in rank[i]]

return rank

def building\_net(N,K,C,rank):

start\_nodes=[]

for i in range(N):

start\_nodes.append(0)

for i in range(N):

for j in range(K):

start\_nodes.append(i+1)

for i in range(K):

start\_nodes.append(i+1+N)

#print(start\_nodes)

end\_nodes=[]

for i in range(N):

end\_nodes.append(i+1)

for i in range(N):

for j in range(K):

end\_nodes.append(j+1+N)

for i in range(K):

end\_nodes.append(N+K+1)

#print(end\_nodes)

capacities=[]

for i in range(N):

capacities.append(1)

for i in range(N):

for j in range(K):

capacities.append(1)

for i in range(K):

capacities.append(C[i])

#print(capacities)

unit\_costs=[]

for i in range(N):

unit\_costs.append(0)

for i in range(N):

for j in range(K):

unit\_costs.append(rank[i].index(j+1)+1)

for i in range(K):

unit\_costs.append(0)

#print(unit\_costs)

supplies=[N]

for i in range(N):

supplies.append(0)

for i in range(K):

supplies.append(0)

supplies.append(-N)

#print(supplies)

return start\_nodes,end\_nodes,capacities,unit\_costs,supplies

def min\_cost\_maximum\_flow(N,start\_nodes,end\_nodes,capacities,unit\_costs,supplies):

min\_cost\_flow = pywrapgraph.SimpleMinCostFlow()

# Add each arc.

for i in range(0, len(start\_nodes)):

min\_cost\_flow.AddArcWithCapacityAndUnitCost(start\_nodes[i], end\_nodes[i],

capacities[i], unit\_costs[i])

# Add node supplies.

for i in range(0, len(supplies)):

min\_cost\_flow.SetNodeSupply(i, supplies[i])

# Find the minimum cost flow between node 0 and node 4.

if min\_cost\_flow.Solve() == min\_cost\_flow.OPTIMAL:

print('')

print('Minimum cost:', min\_cost\_flow.OptimalCost())

print('')

''' #the follows prints the whole network flow

print(' Arc Flow / Capacity Cost')

for i in range(min\_cost\_flow.NumArcs()):

cost = min\_cost\_flow.Flow(i) \* min\_cost\_flow.UnitCost(i)

print('%1s -> %1s %3s / %3s %3s' % (

min\_cost\_flow.Tail(i),

min\_cost\_flow.Head(i),

min\_cost\_flow.Flow(i),

min\_cost\_flow.Capacity(i),

cost))

'''

#print the assignments

print("Assignments as:")

for i in range(min\_cost\_flow.NumArcs()):

cost = min\_cost\_flow.Flow(i) \* min\_cost\_flow.UnitCost(i)

if cost != 0:

print('Patient No. %1s -> Doctor No. %1s' % (min\_cost\_flow.Tail(i), min\_cost\_flow.Head(i) - N))

else:

print('There was an issue with the min cost flow input.')

if \_\_name\_\_ == '\_\_main\_\_':

N=int(input("Number of Patients:"))

K=int(input("Number of Doctors:"))

#C=int(input("Capacity /Doctor:")) #same capacity for every doctor

C=[] #same/different capacity for each doctor

input\_string=input("Capacity of Each Doctor:") #input seperated by space " "

C=input\_string.split()

C=[int(ele) for ele in C] #turn into integers

rank=input\_patient\_preference\_list(N) #patients' preferences

start\_nodes,end\_nodes,capacities,unit\_costs,supplies=building\_net(N, K, C, rank)

min\_cost\_maximum\_flow(N, start\_nodes,end\_nodes,capacities,unit\_costs,supplies)

1. Example result of running script
2. Explanation of algorism
   1. EK
   2. SPFA
   3. Why using Package at the first time (Ortools)