目录

[题 目 基于蚁群算法的0/1背包问题的解决方案 0](#_Toc58006127)

[摘 要 3](#_Toc58006128)

[1 引言 4](#_Toc58006129)

[2 背包问题的数学模型 4](#_Toc58006130)

[3 蚁群算法模型 4](#_Toc58006131)

[4 适用于背包问题的改进蚁群算法 4](#_Toc58006132)

[5 实验与结论 4](#_Toc58006133)

[6 致谢 4](#_Toc58006134)

[7 附录 4](#_Toc58006135)

摘 要

0/1背包问题是一类典型的组合优化问题，且是NP完全问题。常规的求解算法在解决NP困难问题时往往难以胜任，本篇小论文针对0/1背包问题和蚁群算法的特点，提出了一种改进的蚁群算法，引入适合于0/1背包问题的信息素更新模型，使得蚁群算法可以应用到背包问题中。经过Python代码实现并在测试数据集中实验，结果表明蚁群算法同样可以很好地解决背包问题，此算法可以在较少的迭代次数中收敛到一个相对较优的解。

关键词：0/1背包问题 NP完全问题 蚁群算法 启发式算法 Python

# 引言

# 背包问题的数学模型

# 蚁群算法模型

# 适用于背包问题的改进蚁群算法

# 实验与结论

# 致谢

参考文献

附录

main.py

import util

import random

# step 1 : load data

data = util.load('../test-set/test12.CSV')

c,n,v\_max,w,v,res = data # 从数据中获取的各个参数

# step 2 : 初始化参数

m = n # 蚂蚁的数量

alpha = 0.7 # 信息素重要程度因子

beta = 0.3 # 启发函数重要程度因子

rho = 0.1 # 信息素挥发因子

Q = 1 # 常系数

Eta = [v[i]/w[i] for i in range(0,n)] # 启发函数

aver\_Eta = sum(Eta) / len(Eta)

Tau = [ aver\_Eta for i in range(0,n)] # 信息素向量（背包问题中表示每个物品上的信息素的量）

Visited = [] # 访问记录数组 Visited[k] 表示第k只蚂蚁已经访问过的物品

Solution = [] # solution(k,j)表示第k个蚂蚁是否装第j个物品

iter = 0 # 迭代初始值

iter\_max = 60 # 迭代最大次数

# 全局最优

global\_solution = [ 0 for i in range(0,n)]

maxV = 0

iter\_max\_solution = [ [ 0 for j in range(0,n)] for i in range(0,iter\_max)]

iter\_maxV = [ 0 for i in range(0,iter\_max)]

# step 3 : 迭代寻找最佳方案

while iter < iter\_max:

# 清除记录

Visited = [[] for k in range(0,m)]

Solution = [[ 0 for j in range(0,n)] for k in range(0,m) ]

print(iter,"......")

# 计算选择每一个物品的概率

p = []

for pi in range(0,n):

t = (Tau[pi]\*\*alpha)\*(Eta[pi]\*\*beta)

accury = 10000

p.append([pi,int(t\*accury)])

# 逐个蚂蚁选择物品

for k in range(0,m):

p\_new = p[:]

#print("init",p\_new)

# 逐个物品选择

for j in range(0,n):

#print(j,p\_new)

sumP = sum(util.cutByCol(p\_new,1))

roulette = util.sumAdd(p\_new)

# 轮盘赌方式选择物品

x = random.randint(0,sumP)

index = util.selectOne(x,roulette)

# 判断是否能装

if util.canPutIn(Visited,w,c,p\_new[index][0],Solution,k):

Solution[k][p\_new[index][0]] = 1

else:

Solution[k][p\_new[index][0]] = 0

# 记录选择过的物品

Visited[k].append(p\_new[index][0])

# 抛去已经访问过的物品

del p\_new[index]

# 求解每只蚂蚁的总价值量、背包所装总重量、所装东西的单位价值量

total\_v = [ 0 for i in range(0,m)]

for i in range(0,m):

for j in range(0,n):

total\_v[i] += v[j]\*Solution[i][j]

# 求解拥有最大价值的蚂蚁

lmb = util.findMaxValue(Solution,v)

iter\_max\_solution[iter] = Solution[lmb[0]]

iter\_maxV[iter] = lmb[1]

if lmb[1] > maxV:

maxV = lmb[1]

global\_solution = Solution[lmb[0]]

print("max",lmb)

# 计算 delta\_tau

delta\_tau = [ 0 for i in range(0,n)]

for x in range(0,n):

for y in range(0,m):

diff = 0

if Solution[y][x] == 1:

diff = ( v[x] \* Q ) / total\_v[y]

delta\_tau[x] += diff

# 更新信息素

Tau[x] = (1-rho)\*Tau[x] + delta\_tau[x]

iter += 1

print("maxV",maxV)

print("global\_solution",global\_solution)

# step 4 : 存储结果为CSV

util.store('output.csv',iter\_maxV,[maxV],global\_solution)

util.py

import csv

import re

def load(path):

with open(path,'r') as f:

f\_csv = csv.reader(f)

#print(f\_csv)

count = 0

c = 0 # 背包容量

n = 0 # 物品个数

w = [] # w[i] 表示第i件物品的重量

v = [] # v[i] 表示第i件物品的价值

v\_max = 0 # 最大价值总量

res = [] # res[i]表示第i件物品是否选择

pattern = re.compile(" ")

for row in f\_csv:

if count == 0:

pass

elif count == 1:

v\_max = int(row[1])

t = pattern.split(row[0])

c,n = int(t[0]),int(t[1])

else:

wv = pattern.split(row[0])

r = pattern.split(row[1])

i = int(r[0]) - 1

res.append(int(r[1]))

w.append(int(wv[0]))

v.append(int(wv[1]))

count += 1

return (c,n,v\_max,w,v,res)

def store(path,c1,c2,c3):

with open(path,'w') as f:

f\_csv = csv.writer(f)

f\_csv.writerow(c1)

f\_csv.writerow(c2)

f\_csv.writerow(c3)

def confirmData(tp):

c,n,v\_max,w,v,res = tp

#print(c,n,v\_max,w,v,res)

sumW = 0

sumV = 0

for i in range(0,n):

sumW += res[i]\*w[i]

sumV += res[i]\*v[i]

#print(sumW,sumV)

if v\_max == sumV and sumW <= c:

return True

return False

def sumAdd(p):

sum = 0

roulette = [ [p[i][0],p[i][1]] for i in range(0,len(p))]

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

roulette[i][1] = roulette[i][1] + sum

sum = roulette[i][1]

return roulette

def selectOne(x,p):

index = 0

for item in p:

if(x>item[1]):

index += 1

continue

else:

break

return index

def cutByCol(p,n):

col = []

for item in p:

col.append(item[n])

return col

def canPutIn(Visited,w,c,current,Solution,k):

curW = 0

for i in Visited[k]:

curW += w[i]\*Solution[k][i]

if c - curW >= w[current]:

return True

return False

def findMaxValue(Solution,v):

maxV = 0

antN = 0

for i in range(len(Solution)):

val = 0

ant = Solution[i]

for j in range(len(ant)):

val += ant[j]\*v[j]

#print(val)

if val > maxV:

maxV = val

antN = i

return (antN,maxV)

def average(l):

return sum(l)/len(l)