算法设计与分析实验

计算机85 张子诚

实验四

一.问题描述

给定一个1000行$\times$ 20列的0-1矩阵，对于该矩阵的任意一列，其中值为1的元素的数量不超过10%，设有两个非空集合A和B，每个集合由矩阵的若干列构成。集合A和B互斥是指对于矩阵中的任意一行，同时满足下面两个条件

- 若A 中有一个或多个元素在这一行的值是1，则B中的元素在这一行全部是0
- 若B 中有一个或多个元素在这一行的值是1，则A中的元素在这一行全部是0

请你设计一个算法，找出一对互斥集合A和B,使得A和B 包含的列的总数最大。

输入格式

见输入文件`exp6_in.txt`

输出格式

为了减少歧义，一下所说的A 和B 中的元素指的是其选中的列的下标

详细见输出文件`exp6_out.txt`

为了确保输出结果的唯一性，每个集合的输出按照升序排列，如果存在并列的情况，则依次采用以下策略

- A和B 元素个数差的绝对值最小
- A的元素个数要最大
- A的元素和要最小

二.算法设计与分析

问题分析

矩阵中的每一列都有三种选择，加入A集合，加入B集合或者不加入任何一个集合。所以该问题的解空间构成了一个完全三叉树，因此我们可以采取基于深度优先的回溯法遍历产生符合约束条件的解，并且通过不断记录更优的解来获得全局最优解。

回溯法首先从根节点出发，第i层表示对第i列的安排，从左至右第一个子节点表示进入A集合，A中的列数加1，第二个子节点表示进入B集合，B中的列数加1；第三个子节点则表示放弃第i列。

每安排一列，都判断当前列加入某一矩阵后是否与另一个矩阵产生冲突，从而决定可以进入哪一个子节点继续搜索。

每次遍历到叶子节点式，判断当前计算得到的A与B的列数和是否比当前最优值大，若是则更新最优值的记录。并回溯到最近的上一个活结点，选择其余的子树继续搜索。

算法流程

presudo code

- 回溯部分

```
def backtrace(level):
  if(level>19): # 遍历达到叶子节点
    if len(A)>0 and len(B)>0: # 确保集合非空
      if(lenA+lenB>bestlen): # 超出历史最优
        bestlen=lenA+lenB # 最优值
        bestA=recordA # A的最优元素
        bestB=recordB # B的最优元素
      if(lenA+lenB==bestlen): # 并列历史最优
        依次采取以下策略决定是否计入
        1.A和B 元素个数差的绝对值最小
        2.A的元素个数要最大
        3.A的元素和要最小
  else:
    col=mat[:,level] # 当前列
    if (check()函数检查col加入A后是否与B产生冲突):
      A[:,lenA]=col
      lenA+=1
      recordA.append(level)
      backtrace(level+1) #进入下一层
      # 回溯
      recordA.pop()
      lenA-=1
    if (check()函数检查col加入B后是否与A产生冲突):
      B[:,lenB]=col
      lenB+=1
      recordB.append(level)
      backtrace(level+1)
      # 回溯
      recordB.pop()
      lenB-=1
    if(lenA+lenB+19-level>bestlen): # 检查是否有必要不选择当前节点
      backtrace(level+1)

```

- 冲突检查部分
  - 若A 中有一个或多个元素在这一行的值是1，则B中的元素在这一行全部是0
  - 若B 中有一个或多个元素在这一行的值是1，则A中的元素在这一行全部是0

  上述两个条件等价于A,B两个矩阵按行求和后，每行和不能同时大于0

```
def check(X,Y,col,lenX,lenY):
            X[:,lenX]=col # lenx=lenX+1 # 将col加入X
            sumX=np.sum(x[:,:lenX+1],axis=1)
            sumY=np.sum(Y[:,:lenY],axis=1)
            # 检查加入后的X和Y是否互斥
            for i in range(1000):
                if sumX[i]>0 and sumY[i]>0 :
                    return False
            return True
```

- 解的唯一性确定

  这次实验的一个难点在于，如果自己用随机数产生矩阵的话，如果分布选择的不恰当很可能导致矩阵不够稀疏，从而无解，而测试数据给的又过于稀疏所以导致会产生多组解。所以才有了题目中那三个约束条件来确保解的唯一性。理解时应当理解为一种优先级判别机制而不是三个条件都要满足的机制。

  具体来说就是当我满足第一个条件后我就不看后面的，若不满足则在判断是否满足第二个条件时，第一个条件又成了新的约束

三.代码实现与测试

源代码

```
import numpy as np
import time

# 读入元素
def read_data(PATH,mode):
    with open(PATH,mode=mode,encoding='utf-8') as file:
        lines=file.readlines()
        mat=[]
        cnt=0
        for item in lines:
            if cnt==0:
                cnt+=1
                continue
            row=list(map(int,item.strip('\n').split(' ')))
            mat.append(row)
    return np.array(mat)

def check():
    with open("./exp6_out.txt",'r') as file:
        lines=file.readlines()
        for item in lines:
            item=list(map(int,item.strip('\n').split(' ')))
            print(item)

class Sol(object):
    @staticmethod
    def solver(mat):
        # A,B 矩阵
        A,B=np.zeros((1000,20)),np.zeros((1000,20))
        recordA,recordB=[],[]
        lenA,lenB=0,0
        bestlen=0
        bestA,bestB=[],[] # 最优索引
```

```
        sub=20
        sumA=1000
        Length=0

        def check(X,Y,col,lenX,lenY):
            x=X.copy()
            x[:,lenX]=col # lenx=lenX+1 # 将col加入X
            sumX=np.sum(x[:,:lenX+1],axis=1)
            sumY=np.sum(Y[:,:lenY],axis=1)
            # 检查加入后的X和Y是否互斥
            for i in range(1000):
                if sumX[i]>0 and sumY[i]>0 :
                    return False
            return True


        def backtrace(level):
            nonlocal A,B,lenA,lenB,recordA,recordB,bestlen,bestA,bestB,sub,Length,sumA
            if (level>19): #
                if lenA>0 and lenB>0: # A 和 B集合不为空
                    if (lenA+lenB>bestlen): # 超出历史最优
                        bestlen=lenA+lenB
                        bestA=recordA.copy()
                        bestB=recordB.copy()
                        return
                    if (lenA+lenB==bestlen): # 并列历史最优
                        if (abs(lenA-lenB)<sub  and lenA>=lenB): # A和B的个数差的绝对值最小
                            sub=abs(lenA-lenB)
                            sumA=np.sum(recordA)
                            Length=lenA
                            bestlen=lenA+lenB
                            bestA=recordA.copy()
                            bestB=recordB.copy()
                            return

                        if (abs(lenA-lenB)==sub and lenA>Length):# A的长度最大
                            Length=lenA
                            sumA=np.sum(recordA)
                            bestlen=lenA+lenB
                            bestA=recordA.copy()
                            bestB=recordB.copy()
                            return
                        if (np.sum(recordA)<sumA and abs(lenA-lenB)==sub and
lenA==Length): #A的和最小
                            sumA=np.sum(recordA)
                            bestlen=lenB+lenA
                            bestA=recordA.copy()
                            bestB=recordB.copy()
                            return
                    return
                else:
                    col=mat[:,level]
                    if (check(A,B,col,lenA,lenB)): # 尝试将A加入
                        A[:,lenA]=col
                        lenA+=1
                        recordA.append(level)
                        backtrace(level+1)
                        recordA.pop()
                        lenA-=1
                    if (check(B,A,col,lenB,lenA)): # 尝试将B加入
                        B[:,lenB]=col
```

```
                    lenB+=1
                    recordB.append(level)
                    backtrace(level+1)
                    recordB.pop()
                    lenB-=1
                # 该列不进入A也不进入B
                if (lenA+lenB+19-level>bestlen):
                    backtrace(level+1)

        backtrace(0)
        print(bestlen)
        print(bestA)
        print(bestB)
        return

if __name__=='__main__':
    mat=read_data('./exp6_in.txt','r')
    for i in range(0,9):
        Sol.solver(mat[i*1000:(i+1)*1000,:])
    print(0)
    print([])
    print([])

```

结果测试

正确结果(From exp6_out.txt)

```
0 1 2 3 4 5 7 8 9 11
6 10 12 13 14 15 16 17 18 19
1 2 3 4 5 6 7 10 12 13 14 16 17 18 19
0 8 9 11 15
1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 18 19
16
0 1 2 3 4 5 6 7 8 10 11 12 13 14 15 16 17 18 19
9
0 2 3 4 5 6 7 8 10 11 12 13 14 15 16 17 19
1 9 18
0 1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 18 19
17
0 1 2 3 4 5 6 7 9 10 11 13 14 15 16 17 18 19
8 12
0 1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 18 19
14
```

```
PS D:\study\algorithm\ex\ex4\Exp6> & C:/ProgramData/Anaconda3/envs/python37/python.exe d:/study/algorithm/ex/ex4/Exp6/ex6.py
20
[0, 1, 2, 3, 4, 5, 7, 8, 9, 11]
[6, 10, 12, 13, 14, 15, 16, 17, 18, 19]
20
[1, 2, 3, 4, 5, 6, 7, 10, 12, 13, 14, 16, 17, 18, 19]
[0, 8, 9, 11, 15]
20
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[0, 2]
19
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 17, 18, 19]
[16]
20
[0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
[9]
20
[0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19]
[1, 9, 18]
19
[0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19]
[17]
20
[0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 15, 16, 17, 18, 19]
[8, 12]
19
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19]
[14]
0
[]
[]
```