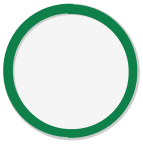
Dupli Checker

PLAGIARISM SCAN REPORT

0%
Plagiarised



100%
Unique

Date	2021-10-25
Words	702
Characters	6013

Content Checked For Plagiarism

```
# importing required libraries
from random import randint
import math

# In this new state is created using random number ever time and its goodness is caculated using similarities and distances
def create_new_state(arr,score):
 new_arr=arr[:]
 # Two random index generated
 a = randint(0,n-1)
 b = randint(0,n-1)
 # Swapped the index values
 if b!=a:
  new_arr[a],new_arr[b] = new_arr[b],new_arr[a]
 elif b!=n-1:
  new_arr[a],new_arr[b+1] = new_arr[b+1],new_arr[a]
 else:
  new_arr[a],new_arr[b - 1] = new_arr[b - 1],new_arr[a]
 # Calculating the session in which the elements lies
 session1 = int(math.floor(a / k))
 session2 = int(math.floor(b / k))
 # If the session is same no change in goodness return new array and same score
 if (session2==session1):
  return (new_arr,score)
 # If sessions are in same time slot but parallel we only need to calculate the similarities as distances will be same
 # Calculate the similarity of index a whith other papers in same sessions similar for index b
 # here session1*k,(session1+1)*k implies the range of elements that belong in the same session
 if (math.floor(a/(k*p))==math.floor(b/(k*p))):
  similar_b1_a=0
  similar_b1_b=0
  similar_b2_b=0
  similar_b2_a=0
  for i in range(session1*k,(session1+1)*k):
   if i != a:
    similar_b1_a+=similar[arr[a]][arr[i]]
    similar_b1_b+=similar[arr[b]][arr[i]]
  for i in range(session2*k,(session2+1)*k):
   if i!=b:
```

```
        similar_b2_a += similar[arr[a]][arr[i]]
        similar_b2_b += similar[arr[b]][arr[i]]
  # Since elements are swapped the similarities of b needs to be subtracted and a need to be added for block 2 and vice vera for block 1
    return (new_arr,score-similar_b1_a+similar_b1_b-similar_b2_b+similar_b2_a)
  # If sessions are in different time slots we need to calculate the similarities and distances
  # Calculate the similarity of index a whith other papers in same sessions similar for index b
  # here session1*k,(session1+1)*k implies the range of elements that belong in the same session
  # here time_slot_1 and time_slot_2 represent the time slots of swapped indexes
  else:
    session1 = int(math.floor(a / k))
    session2 = int(math.floor(b / k))
    time_slot_1=int(math.floor(a / (k*p)))
    time_slot_2 = int(math.floor(b / (k*p)))
    similar_b1_a = 0
    similar_b1_b = 0
    similar_b2_b = 0
    similar_b2_a = 0
    diff1 = 0
    diff2 = 0
    diff3 = 0
    diff4 = 0
    for i in range(session1 * k, session1 * k + k):
      similar_b1_a += similar[arr[a]][arr[i]]
      if i!=a:
        similar_b1_b+=similar[arr[b]][arr[i]]
    for i in range(session2 * k, session2 * k + k):
      similar_b2_b += similar[arr[b]][arr[i]]
      if i!=b:
        similar_b2_a += similar[arr[a]][arr[i]]
    for i in range(time_slot_1*k*p,session1 * k):
      diff1 += distance[arr[a]][arr[i]]
      diff2 += distance[arr[b]][arr[i]]
    if ((session1+1)%p!=0):
      for i in range(session1 * k + k,(time_slot_1+1)*k*p):
        diff1 += distance[arr[a]][arr[i]]
        diff2 += distance[arr[b]][arr[i]]
    for i in range(time_slot_2*k*p,session2 * k):
      diff3 += distance[arr[b]][arr[i]]
      diff4 += distance[arr[a]][arr[i]]
    if ((session2+1) % p != 0):
      for i in range(session2 * k + k,(time_slot_2+1)*k*p):
        diff3 += distance[arr[b]][arr[i]]
        diff4 += distance[arr[a]][arr[i]]
    # Since elements are swapped the similarities of b needs to be subtracted and a need to be added for block 2 and vice vera for block 1
    # Same for Distances
    return (new_arr, score - similar_b1_a + similar_b1_b - similar_b2_b + similar_b2_a+c*(-diff1+diff2-diff3+diff4))

# calculates the goodness
def goodness(arr):
    g=0
    for i in range(p*t):
        for j in range(k-1):
```

```
            for x in range(j+1,k):
                g+=similar[arr[j+k*i]][arr[x+k*i]]
    for a in range(t):
        for j in range(p):
            for i in range(a*p*k+k*j,a*p*k+k*(j+1)):
                for x in range(a*p*k+k*(j+1),(a+1)*p*k):
                    g+=c*distance[arr[i]][arr[x]]
    return round(g,2)

# used for local_search as to get the best solution or goal state
def local_search(arr):
    count=0
    c=0
    goodness_arr=goodness(arr)
    # Limiting the number of iterations to n^2-n
    while(count<=n**2-n):
        arr_new,goodness_new = create_new_state(arr,goodness_arr)
        count+=1
        c+=1
        # If Need to see all the states and there goodness use this below print
        #print(arr_new,goodness_new)
        if goodness_new>goodness_arr:
            arr=arr_new
            goodness_arr = goodness_new
            count=0
            #print(arr_new,goodness_new)
        # if we do not get a answer in these many iterations we break
        if c==5000:
            break
    return arr

# Printing the Final schedule
def print_schedule(arr,k,t,p):
    for i in range(p):
        for j in range(t):
            for l in range(k):
                print(final_arr[l+j*p*k+i*k]+1,end=" ")
            if j!=t-1:
                print("|",end=" ")
            else:
                print()

# Taking input from the user
k = int(input())
p = int(input())
t = int(input())
c = float(input())
# n = total number of papers
n = k*p*t
#print(n,k,p,t,c)
# taking distances and calculating similarities
distance = []
similar = []
for i in range(n):
```

```
    distance.append(list(float(y) for y in input().split()))
    similar.append(list(1-float(y) for y in distance[i]))

# creating the arr variable and initializing values 0 to n-1 also can be called initial state.
arr = []
for i in range(n):
  arr+=[i]
#print(arr)

# Caculating the goodness of initial state
s=goodness(arr)

# Finialized array or schedule output
final_arr=local_search(arr)

# Printing the final schedule
print_schedule(final_arr,k,t,p)
```

Matched Source

No plagiarism found