```python
#####################################################
# algorithm for random search
#####################################################

# Load the libraries needed
import time
import numpy as np
import pandas as pd
from copy import deepcopy
import random

#read 5 datasets
car1 = pd.read_csv('car1.csv')
car1 = np.array(car1)
car2 = pd.read_csv('car2.csv')
car2 = np.array(car2)
car3 = pd.read_csv('car3.csv')
car3 = np.array(car3)
car4 = pd.read_csv('car4.csv')
car4 = np.array(car4)
car5 = pd.read_csv('car5.csv')
car5 = np.array(car5)

time_start=time.time() #recode the program start time

#calculate the makespan
#p_ij=dataset ,nbm=machine number, my_seq=current job sequence

def makespan(current_seq, p_ij, nbm):
    c_ij = np.zeros((nbm, len(current_seq) + 1))
    for j in range(1, len(current_seq) + 1):
        c_ij[0][j] = c_ij[0][j - 1] + p_ij[0][current_seq[j - 1]]

    for i in range(1, nbm):
        for j in range(1, len(current_seq) + 1):
            c_ij[i][j] = max(c_ij[i - 1][j], c_ij[i][j - 1]) + p_ij[i][current_seq[j - 1]]
    return current_seq,c_ij

#Apply the random search with 1000*n solution evaluation and calculate their makespan
def random_search(car_num):
    p_ij=car_num
    nbm=len(p_ij)
    nbj=len(p_ij[0])
    #original job sequence
    seq_origin=list(range(len(p_ij[0])))
    current_seq = []
    result_time=[]
    min_seq=[]
    loop_result={}
    for i in range(1000*nbj):
        seq = deepcopy(seq_origin)
        random.shuffle(seq)
        current_seq,c=makespan(seq,p_ij,nbm)
        mp=c[len(c)-1][len(c[0])-1]
        result_time.append(mp)
        loop_result[i]={'seq':current_seq,'makespan_values':mp}
    min_mp=min(result_time)
    for loop_time in loop_result.values():
        if loop_time['makespan_values']==min_mp:
```

```
            min_seq.append(loop_time['seq'])
    return min_seq,min_mp

#best_time has 30 makespan values , best_seq 30 seqs
def main(n,car_num):
    best_time=[]
    best_seq=[]
    seed_result={}
    for times in range(n):
        np.random.seed(n*times)
        seq,min_mp = random_search(car_num)
#         best_seq.append(seq)
        best_time.append(min_mp)
        seed_result[times]={'seq':seq,'makespan_values':min_mp}
        #print 30 seeds best sequence and makespan for each 1000*n solution evaluations
        print('Seed {0} best sequence:{1} makespan values:{2}\n'.format(times+1,seq,min_mp))
    print('Makespan table:\n mininum:{0}, maximum:{1}, mean:{2}, standard deviation:{3}'.\
          format(np.min(best_time),np.max(best_time),np.mean(best_time),np.std(best_time)))
    for times in seed_result.values():
        if times['makespan_values']==np.min(best_time):
            best_seq.extend(times['seq'])

# remove the same best_seq,not-repeating
    NP_bestseq=[]
    [NP_bestseq.append(i) for i in best_seq if not i in NP_bestseq]
    NP_bestseq.sort()
    return best_time,NP_bestseq

# set 30 seeds
def min_mp(car_num):
    n = 30
    best_time=main(n,car_num)
    return best_time

#chose the car number here
best_time=min_mp(car2)

#recode the program running time
time_end=time.time()
print(' Program Time Cost:',time_end-time_start,'s')
```

```python
##################################################
# algorithm for neh
##################################################

# Load the libraries needed
import pandas as pd
import numpy as np

# read the 5 datasets
car1 = pd.read_csv('car1.csv')
car1 = np.array(car1)
car2 = pd.read_csv('car2.csv')
car2 = np.array(car2)
car3 = pd.read_csv('car3.csv')
car3 = np.array(car3)
car4 = pd.read_csv('car4.csv')
car4 = np.array(car4)
car5 = pd.read_csv('car5.csv')
car5 = np.array(car5)

#chose the car number
p_ij=car2

nbm=len(p_ij)
nbj=len(p_ij[0])

#calculate makespan
def makespan_neh(current_seq, p_ij, nbm):
    c_ij = np.zeros((nbm, len(current_seq) + 1))
    for j in range(1, len(current_seq) + 1):
        c_ij[0][j] = c_ij[0][j - 1] + p_ij[0][current_seq[j - 1]]

    for i in range(1, nbm):
        for j in range(1, len(current_seq) + 1):
            c_ij[i][j] = max(c_ij[i - 1][j], c_ij[i][j - 1]) + p_ij[i][current_seq[j - 1]]
    return c_ij[nbm - 1][len(current_seq)]

def sum_processing_time(index_job, data, nb_machines):
    sum_p = 0
    for i in range(nb_machines):
        sum_p += data[i][index_job]
    return sum_p


#Sort the current sequence by makespan
def order_neh(data, nb_machines, nb_jobs):
    my_seq = []
    for j in range(nb_jobs):
        my_seq.append(j)
    return sorted(my_seq,key=lambda x:sum_processing_time(x, data, nb_machines),reverse=True)

#insert the new job and obtain the new sequence
def insertion(sequence, index_position, value):
    new_seq = sequence[:]
    new_seq.insert(index_position, value)
    return new_seq

#run the neh
#calculate the new makespan after insert the new job
```

```python
#Compare the makespan of each sequence, retaining the best sequence and makespan
def neh(data, nb_machines, nb_jobs):
    order_seq = order_neh(data, nb_machines, nb_jobs)
    seq_current = [order_seq[0]]
    for i in range(1, nb_jobs):
        min_cmax = float("inf")
        for j in range(0, i + 1):
            tmp_seq = insertion(seq_current, j, order_seq[i])
            cmax_tmp = makespan_neh(tmp_seq, data, nb_machines)
            print(tmp_seq, cmax_tmp)
            if min_cmax > cmax_tmp:
                best_seq = tmp_seq
                min_cmax = cmax_tmp
        seq_current = best_seq
    return seq_current,makespan_neh(seq_current, data, nb_machines)

# print the NEH seq and it's makespan
seq, cmax = neh(p_ij, nbm, nbj)
print('Number of Machines:{0},Number of Jobs:{1}'.format(nbm,nbj))
print("NEH sequence:", seq)
print("Makespan:",cmax)
```

```
####################################################
# algorithm for GA
#(Algo4:2-point crossover(C1) with shift mutation(SM))
####################################################

import numpy as np
import pandas as pd
import random

# read the 5 datasets
car1 = pd.read_csv('car1.csv')
car1 = np.array(car1)
car2 = pd.read_csv('car2.csv')
car2 = np.array(car2)
car3 = pd.read_csv('car3.csv')
car3 = np.array(car3)
car4 = pd.read_csv('car4.csv')
car4 = np.array(car4)
car5 = pd.read_csv('car5.csv')
car5 = np.array(car5)

#choose the car number
p_ij=car2

#define the NEH_SEQ
NEH_seq=[4, 7, 5, 6, 2, 0, 3, 1]

#5 cars NEH_seq
#car1 [7, 0, 4, 8, 2, 10, 3, 6, 5, 1, 9]
#car2 [4, 7, 5, 6, 2, 0, 3, 1]
#car3 [18, 7, 11, 15, 19, 4, 0, 9, 12, 2, 1, 17, 8, 6, 5, 10, 3, 14, 13, 16]
#car4 [17, 12, 9, 0, 16, 18, 8, 1, 11, 2, 7, 3, 4, 14, 10, 15, 5, 6, 19, 13]
#car5 [13, 19, 28, 4, 17, 10, 16, 12, 5, 8, 1, 0, 2, 20, 6, 22, 9, 23, 7, 3,
#      15, 29, 25, 26, 14, 11, 24, 21, 18, 27]

nbm=len(p_ij)
nbj=len(p_ij[0])

print('Number of Machines:{0},Number of Jobs:{1}'.format(nbm,nbj))

#set the parameters
Npop = 30     # Number of population
Pc = 1        # Probability of crossover
Pm = 0.8      # Probability of mutation
D=0.95        #Threshold parameter
sig=0.99      #sigema=0.99

print('The parameters we chosen:\n''Population size:{0}\nCrossover probability:{1}\n\
Initial mutation probability:{2}\nThreshold parameter:{3}\n'.format(Npop,Pc,Pm,D))

#Number of evaluations
stopGeneration = 1000*nbj
#(it will take a long time to run the algorithm)
#(to test the algrithm we can set stopGeneration small)
#stopGeneration = 100

#calculate the makespan
def makespan_GA(current_seq, p_ij, nbm):
    c_ij = np.zeros((nbm, len(current_seq) + 1))
```

```python
    for j in range(1, len(current_seq) + 1):
        c_ij[0][j] = c_ij[0][j - 1] + p_ij[0][current_seq[j - 1]]

    for i in range(1, nbm):
        for j in range(1, len(current_seq) + 1):
            c_ij[i][j] = max(c_ij[i - 1][j], c_ij[i][j - 1]) + p_ij[i][current_seq[j - 1]]
    return c_ij[nbm - 1][nbj]

#initialize the population and append the neh_seq to the initial population
def initialization(Npop):
    pop = []
    for i in range(Npop):
        p = list(np.random.permutation(nbj))
        while p in pop:
            p = list(np.random.permutation(nbj))
        pop.append(p)
    pop.append(NEH_seq)
    return pop

#select the population
def selection(pop):
    #popobj[0]=makespans ,popobj[1]=order from 0 to 30 ,totally 31
    popObj = []
    for i in range(len(pop)):
        popObj.append([makespan_GA(pop[i],p_ij,nbm),i])
#sort by makespan,so the plpobj[1]will not be in order
    popObj.sort()
    distr = []
    distrInd = []

    for i in range(len(pop)):
        #append the makespan's order in the distrInd
        #(while the makespan in pop are in order ) ascending in makespan index
        distrInd.append(popObj[i][1])
        #Select parent 1 using 2k/M(M+1) fitness_rank distribution
        prob = (2*(len(pop)-i)) / (len(pop) * (len(pop)+1))
        distr.append(prob)

    parents = []
    for i in range(len(pop)):
        #Select parent 2 using uniform distribution
        parents.append(list(np.random.choice(distrInd, 1, p=distr)))
        parents[i].append(np.random.choice(distrInd))
    return parents

#2-point crossover (C2)
def crossover(parents):
    pos = list(np.random.permutation(np.arange(nbj-1)+1)[:2])

    if pos[0] > pos[1]:
        t = pos[0]
        pos[0] = pos[1]
        pos[1] = t

    child = list(parents[0])

    for i in range(pos[0], pos[1]):
        child[i] = -1

    p = -1
```

```
    for i in range(pos[0], pos[1]):
        while True:
            p = p + 1
            if parents[1][p] not in child:
                child[i] = parents[1][p]
                break
    return child

#shift mutation
def mutation(sol):
    pos = list(np.random.permutation(np.arange(nbj))[:2])

    if pos[0] > pos[1]:
        t = pos[0]
        pos[0] = pos[1]
        pos[1] = t

    remJob = sol[pos[1]]

    for i in range(pos[1], pos[0], -1):
        sol[i] = sol[i-1]

    sol[pos[0]] = remJob

    return sol

#Update the population
def elitistUpdate(oldPop, newPop):
    bestSolInd = 0
    bestSol = makespan_GA(oldPop[0],p_ij,nbm)

    for i in range(1, len(oldPop)):
        tempObj = makespan_GA(oldPop[i],p_ij,nbm)
        if tempObj < bestSol:
            bestSol = tempObj
            bestSolInd = i
    rndInd = random.randint(0,len(newPop)-1)
    newPop[rndInd] = oldPop[bestSolInd]
    return newPop

# find the best solution
def findBestSolution(pop):
    bestObj = makespan_GA(pop[0],p_ij,nbm)
    avgObj = bestObj
    bestInd = 0
    for i in range(1, len(pop)):
        tObj = makespan_GA(pop[i],p_ij,nbm)
        avgObj = avgObj + tObj
        if tObj < bestObj:
            bestObj = tObj
            bestInd = i

    return bestInd, bestObj, avgObj/len(pop)

# Run the algorithm for 'stopGeneration'(1000*n) times generation
def Loop(Npop):

    Pm = 0.8
    # Creating the initial population
    population = initialization(Npop)
```

```
    for i in range(stopGeneration):
        # Selecting parents
        parents = selection(population)
        childs = []

        # Apply crossover
        for p in parents:
            r = random.random()
            y = random.random()
            if r < Pc:
                childs.append(crossover([population[p[0]], population[p[1]]]))
            else:
                if y < 0.5:
                   childs.append(population[p[0]])
                else:
                   childs.append(population[p[1]])

        # Apply mutation
        for c in childs:
            r = random.random()
            if r < Pm:
                c = mutation(c)
                # Update the population
        population_new = elitistUpdate(population, childs)
        # Results Time
        bestSol, minObj, avgObj = findBestSolution(population_new)
        mpset=[]
        mpset.append(minObj)
        if minObj/avgObj > D:
            Pm=sig*Pm
    return min(mpset)

# recode 30 seeds results for each run
def ran_seed(carnum):
    result=[]
    for times in range(30):
        np.random.seed(30*times)
        min_makespan=(Loop(Npop))
        result.append(min_makespan)
        print('seed {0}: makespan value {1}'.format(times+1,min_makespan))
    return result

#print the makespan table
result=ran_seed(p_ij)
print('\nmakespan table:\n''mininum:{0}, maximum:{1}, mean:{2}, standard deviation:{3}'.\
format(np.min(result),np.max(result),np.mean(result),np.std(result)))
```