Desktop

May 17, 2021

0.0.1

[1]:
```
from itertools import permutations
import numpy as np
import random
import math
from copy import deepcopy
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

0.1 0.

[2]:
```
params={
    "num_work":20,
    "num_process":10,
    "num_machine":2,
    "file":"input_origin.txt",
    # GA
    "num_group":20,#
    "prob_cross":0.6,#
    "prob_mutate":0.05 #
}
```

0.2 1.

- $ p[i][j]: i   j$

[3]:
```
def read_for_p(file):
    f=open(file,"r")
    p=[[0]*params["num_process"] for _ in range(params["num_work"])]
    for index,lines in enumerate(f.readlines()):
        line=lines.split()
        p[index]=list(map(int,line))
    return p
```

[4]:

```
p=read_for_p(params["file"])
```

[5]:

```
p
```

[5]:

```
[[77, 95, 41, 97, 47, 45, 10, 41, 72, 8],
 [99, 28, 42, 4, 7, 30, 65, 45, 51, 94],
 [74, 25, 92, 29, 4, 21, 47, 36, 61, 9],
 [4, 21, 40, 80, 66, 85, 1, 33, 1, 4],
 [49, 95, 96, 74, 96, 63, 59, 84, 70, 29],
 [53, 59, 75, 19, 13, 50, 82, 60, 9, 13],
 [88, 47, 28, 11, 86, 90, 93, 38, 33, 59],
 [92, 99, 84, 13, 73, 55, 19, 93, 74, 25],
 [2, 49, 86, 46, 58, 42, 24, 79, 12, 17],
 [97, 18, 28, 77, 92, 54, 49, 24, 19, 71],
 [28, 93, 93, 7, 25, 89, 49, 11, 93, 45],
 [64, 22, 91, 56, 46, 27, 32, 70, 94, 5],
 [25, 96, 98, 51, 21, 20, 93, 64, 86, 11],
 [19, 41, 87, 15, 31, 78, 54, 74, 71, 6],
 [81, 1, 74, 56, 8, 55, 3, 92, 28, 5],
 [9, 29, 49, 48, 72, 38, 26, 3, 49, 80],
 [5, 74, 19, 27, 71, 35, 52, 76, 79, 47],
 [8, 66, 40, 71, 17, 61, 84, 49, 52, 56],
 [34, 7, 58, 94, 22, 27, 40, 19, 26, 77],
 [13, 56, 45, 27, 40, 26, 90, 28, 27, 88]]
```

0.3 2. GA

- enconding
- decoding
- fitness
- choose
- cross 1 2 > 1 2 1 2

  father1: 14|653|72, father2: 26|371|45

  son1 : 46|371|52 ,son2: 27|653|14

- mutate >

[6]:

```
class GA_solve_HFSSP:
    def __init__(self,params):
        # GA
        self.num_group=params["num_group"]
        self.prob_cross=params["prob_cross"]
        self.prob_mutate=params["prob_mutate"]
        #
        self.num_work=params["num_work"]
        self.num_process=params["num_process"]
        self.num_machine=params["num_machine"]

    def __encoding(self,num_group,num_work):
        """
           num_group       num_work
          list(permutations)
        np.random.shuffle     >30
        """
        group=[]
        if num_group>math.factorial(num_work)*0.7:
            print("num_group ")
            raise ValueErroe
        while len(group)!=num_group:
            a=random.sample(range(num_work),num_work)
            if a not in group:
                group.append(a)
        return group

    def decoding(self,gene):
        """
           ( )
        gene:
        """
        num_process=self.num_process
        num_machine=self.num_machine
        num_work=len(gene)
        machine_time=[0 for _ in range(num_machine)] #  machine
        gene_time=[0 for _ in range(num_work)]
        #
        def step(gene,machine_time,gene_time,stepnum):
            machine_time=[0 for _ in range(num_machine)] #  machine
            for i in range(len(gene)):
                index=machine_time.index(min(machine_time)) #    machine
                #   machine  i    J
                machine_time[index]=max(machine_time[index],gene_time[ gene[i]␣
↪])+p[gene[i]][stepnum]
                #    i
                gene_time[ gene[i] ]=machine_time[index]
            return machine_time,gene_time

        for j in range(num_process):
            machine_time,gene_time = step(gene,machine_time,gene_time,j)
            #
            gene=[idx for idx,value in sorted(enumerate(gene_time),key=lambda x:
↪x[1])]
```

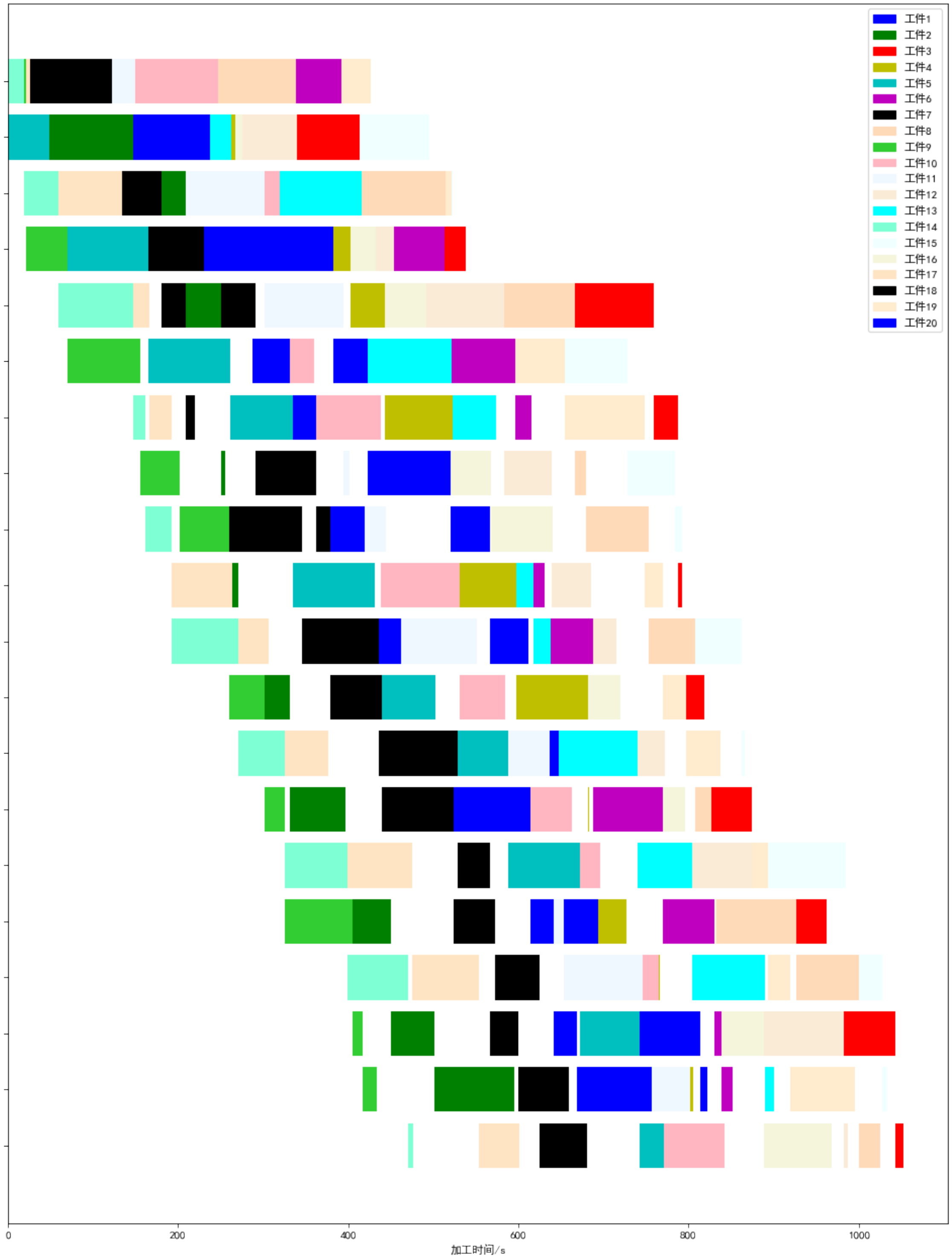
```
        total_time=max(machine_time)
        return total_time

    def __fitness(self,time_list):
        """
          1/(1+x)
        """
        a=np.array(list(map(lambda x:1/(x),time_list)))
        return a/sum(a)

    def __choose(self,fitness_list,group_list):
        """



        """
        a,b=np.random.
↪choice(range(len(group_list)),2,replace=False,p=fitness_list)
        return (group_list[a],group_list[b])

    def __cross(self,sample_tuple):
        """


         1    2         1  2
        """
        a,b=sample_tuple
        assert len(a)==self.num_work

        index1,index2=sorted(np.random.choice(range(self.
↪num_work),2,replace=False))
        new_a,new_b=b[index1:index2+1],a[index1:index2+1]
        dict_a,dict_b=set(b[index1:index2+1]),set(a[index1:index2+1])
        count_a,count_b=0,0
        for index,value in enumerate(a):
            if value not in dict_a:
                if count_a<index1:
                    new_a.insert(count_a,value)
                    count_a +=1
                else:
                    new_a.append(value)
        for index,value in enumerate(b):
            if value not in dict_b:
                if count_b<index1:
                    new_b.insert(count_b,value)
                    count_b+=1
                else:
                    new_b.append(value)
        return new_a,new_b
```

```python
    def __mutate(self,gene):
        gene_new=deepcopy(gene)
        index1,index2=sorted(np.random.choice(range(self.num_work),2,replace=False))
        gene_new[index1],gene_new[index2]=gene_new[index2],gene_new[index1]
        return gene_new

    def fit(self):
        self.group=self.__encoding(self.num_group,self.num_work)
        self.deco=list(map(self.decoding,self.group))
        best_time=min(self.deco)
        best_seq=self.group[self.deco.index(min(self.deco))]

        for epoch in range(300):
            self.fitness=self.__fitness(self.deco)
            self.new_group=[]
            while len(self.new_group)<self.num_group:
                self.sample=self.__choose(self.fitness,self.group)
                cross_seed=random.randint(0,100)
                if cross_seed<self.prob_cross*100:
                    self.son=self.__cross(self.sample)
                else:
                    self.son=self.sample
                mutate_seed=random.randint(0,100)
                if mutate_seed<self.prob_mutate*100:
                    self.mute=list(map(self.__mutate,self.son))
                else:
                    self.mute=self.son

                self.new_group.extend(deepcopy(self.mute))
            self.group=self.new_group
            self.deco=list(map(self.decoding,self.group))

            temp=min(self.deco)
            temp_seq=self.group[self.deco.index(temp)]
            if temp<best_time:
                best_time=temp
                best_seq=deepcopy(temp_seq)

        best_seq1=list(map(lambda x:x+1,best_seq))
        print("  :",best_time)
        print("  :",best_seq1)
        self.gante(best_seq1)
        return best_time,best_seq1

    def gante(self,seq):
        ## decoding
        seq=list(map(lambda x:x-1,seq))
        num_process=self.num_process
        num_machine=self.num_machine
        num_work=self.num_work
        def decode(gene):
            machine_record=[]
            gene_record=[]
            use_record=[]

            machine_time=[0 for _ in range(num_machine)] #  machine
            gene_time=[0 for _ in range(num_work)]
            #
            def step(gene,machine_time,gene_time,stepnum):
                machine_use=[[],[]]
                machine_time=[0 for _ in range(num_machine)] #  machine
                for i in range(len(gene)):
                    index=machine_time.index(min(machine_time)) #    machine
                    machine_use[index].append(gene[i])
                    #  machine  i   J
                    machine_time[index]=max(machine_time[index],gene_time[␣
↪gene[i] ])+p[gene[i]][stepnum]
                    #    i
                    gene_time[ gene[i] ]=machine_time[index]
                return machine_time,gene_time,machine_use

            for j in range(num_process):

                machine_time,gene_time,machine_use =␣
↪step(gene,machine_time,gene_time,j)

                machine_record.append(machine_time)
                gene_record.append(deepcopy(gene_time))
                use_record.append(deepcopy(machine_use))
                #
                gene=[idx for idx,value in␣
↪sorted(enumerate(gene_time),key=lambda x:x[1])]
            total_time=max(machine_time)

            return machine_record,gene_record,use_record

        machine_record,gene_record,use_record=decode(seq)
        color =␣
↪['b','g','r','y','c','m','k','peachpuff','limegreen','lightpink','aliceblue','antiquewhite'


        y,width,left,color_list,label_list=[],[],[],[],[]
        for index1,value1 in enumerate(use_record): ## index1  -1
            for index2,value2  in enumerate(value1): ## index2  -1
                for j in value2: ## j
                    ##  y
                    y.append(num_process*num_machine-index1*2-index2)
                    ##  width
                    width.append(p[j][index1])
                    ##  left
                    left.append(gene_record[index1][j]-p[j][index1])
                    ##  color
                    color_list.append(color[j])
                    word=" {} {}".format(index1+1,index2+1)
                    if word not in label_list:
                        label_list.append(word)
                    else:
                        label_list.append("")
        import matplotlib.pyplot as plt


        plt.rcParams['font.sans-serif'] = ['SimHei']  #
        plt.rcParams['axes.unicode_minus'] = False  #
        plt.figure(figsize=(15,num_process*num_machine),dpi=80)
        plt.barh(y,width,left=left,color=color_list,tick_label=label_list)#
        labels =[" %d"%(f+1) for f in range(num_work)]
        patches = [ mpatches.Patch(color=color[i], label="{:s}".format(labels[i]) ) for i in range(num_work) ]
        plt.legend(handles=patches,loc=1)
    #     plt.grid(linestyle="--",alpha=0.5)
        #XY
        plt.xlabel("  /s")
        plt.ylabel("")
        plt.show()#
```

[7]:

```
test=GA_solve_HFSSP(params)
best_time,best_seq=test.fit()
```

```
: 1052
: [14, 5, 9, 17, 7, 2, 18, 11, 20, 10, 1, 13, 8, 4, 16, 12, 6, 3, 19, 15]
```


工件1
工件2
工件3
工件4
工件5
工件6
工件7
工件8
工件9
工件10
工件11
工件12
工件13
工件14
工件15
工件16
工件17
工件18
工件19
工件20
0
200
400
600
800
1000
加工时间/s

[ ]:

[ ]: