گزارش پروژه دوم هوش

امیرحسین باوند: 9331028

ابتدا توابع سوالات را معرفی می کنیم .

سوال یک :

ابتدا initial را انجام می دهیم .

**def** \_\_init\_\_(self):  
 number\_of\_cities=int(input("enter number of cities"))  
 self.number\_of\_cities=number\_of\_cities  
 self.a = np.zeros(shape=(number\_of\_cities,number\_of\_cities))  
 **for** i **in** range(0,number\_of\_cities):  
 s=input("enter cities")  
 data=s.split()  
 **for** j **in** range(0,number\_of\_cities):  
 self.a[i][j]=int(data[j])

ابتدا اینیشیال می کنیم . و ورودی را از کاربر دریافت می کنیم.

تابع initial state

**def initialstate**(self):  
  
 a=[]  
 **while** len(a)<self.number\_of\_cities:  
 l=(random.randint(0,self.number\_of\_cities-1))  
 **if**(l **not in** a):  
 a.append(l)  
 y=node(a,**None**,1)  
 **return** y

حالت اولیه را برابر به صورت رندم میسازد.

تابع action

**def actions**(self,node1):  
 ac=[]  
 **for** i **in** range(0,self.number\_of\_cities):  
 **for** j **in** range(0,self.number\_of\_cities):  
 **if**(i!=j):  
 a = []  
 a.append(i)  
 a.append(j)  
 ac.append(a)  
 **return** ac

این تابع به ما تمام action های ممکن را برمیگرداند .

و هر اکشن یک آرایه دوتایی از دو شهر متفاوت است که قرار است جای آن ها را با هم عوض کنیم.

تابع result :

temp=[]  
**for** i **in** range(0,self.number\_of\_cities):  
 temp.append(node1.state[i])  
temp[int(action[0])]=node1.state[int(action[1])]  
temp[action[1]]=node1.state[action[0]]  
y=node(temp,node1,1)  
**return** y

این تابع یک اکشن و نود می گیرد و جای دو شهر را در آن عوض می کند و آرایه جدید پیمایش را به عنوان یک استیت در نظر میگیرد و برمیگرداند.

تابع heuristic

این تابع هزینه هر مسیر را محاسبه میکند و منفی ان را برمیگرداند. دلیل منفی کردن این است که در توابع الگوریتم امتیاز بیشتر یعنی یک جواب بهتر و به همین دلیل منفی میکنیم که این اتفاق در جواب ها بیفتد .

**def heuristic**(self,node1):  
 co=0  
 **for** i **in** range(0,self.number\_of\_cities-1):  
 co=co+self.a[node1.state[i]][node1.state[i+1]]  
  
 co=co+self.a[node1.state[self.number\_of\_cities-1]][node1.state[0]]  
  
 **return** -co

حال سه روش مختلف برای کاهش دمای الگوریتم پیشنهاد میکنیم .

راه اول

هر بار یک t میگیریم و به آن یکی یکی اضافه می کنیم و یک عدد بین صفر و یک انتخاب میکنیم و هر بار t را به توان آن میرسانیم و در صد ضرب میکنیم و به عنوان T برمیگردانیم .

**return** 100\*math.pow(0.9,t)

راه دوم

هر بار یک t میگیریم و به آن یکی اضافه میکنیم و یک عدد بین صفر و یک انتخاب میکنیم و هر بار آن عدد را تقسیم بر t میکنیم و جواب را به عنوان T یا دما بر میگردانیم.

راه سوم

یک کانتر را در نظر میگیریم و برای محاسبه دما آن را به توان دو رسانده و صد را بر آن تقسیم میکنیم .

قسمت ب حال مساله را با الگوریتم های خواسته شده حل میکنیم

ورودی مساله را مطابق با ورودی داده شده در صورت پروژه میدهیم.

ابتدا simple\_hill\_climbing

[0, 1, 2, 3, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

visitenode= 40

expandenode= 2

hazine= 19.0

حال با استفاده از الگوریتم تپه نوردی تصادفی به حل مساله میپردازیم .

[0, 1, 2, 3, 4]

[4, 1, 2, 3, 0]

[2, 1, 4, 3, 0]

[2, 1, 3, 4, 0]

visitenode= 80

expandenode= 4

hazine= 19.0

[2, 1, 3, 4, 0]

وبه جواب بالا میرسیم

حال مساله را با استفاده از الگوریتم تپه نوردی اولین انتخاب حل میکنیم

[0, 1, 2, 3, 4]

-29.0

[3, 1, 2, 0, 4]

visitenode= 3

expandenode= 1

hazine= 19.0

[3, 1, 2, 0, 4]

و به جواب های بالا می رسیم.

حال با اجرای الگوریتم تپه نوردی شروع مجدد تصادفی به جواب های زیر میرسیم

0enter citiesenter citiesenter citiesenter cities[4, 2, 0, 1, 3]

[1, 2, 0, 4, 3]

[1, 2, 0, 4, 3]

visitenode= 40

expandenode= 2

hazine= 19.0

[0, 1, 4, 2, 3]

[2, 1, 4, 0, 3]

[1, 2, 4, 0, 3]

[1, 3, 4, 0, 2]

[1, 3, 4, 0, 2]

visitenode= 80

expandenode= 4

hazine= 19.0

[3, 4, 1, 2, 0]

[4, 3, 1, 2, 0]

[4, 3, 1, 2, 0]

visitenode= 40

expandenode= 2

hazine= 19.0

[0, 3, 2, 1, 4]

[0, 4, 2, 1, 3]

[4, 0, 2, 1, 3]

[4, 0, 2, 1, 3]

visitenode= 60

expandenode= 3

hazine= 19.0

[3, 4, 0, 1, 2]

[0, 4, 3, 1, 2]

[0, 4, 3, 1, 2]

visitenode= 40

expandenode= 2

hazine= 19.0

[1, 2, 3, 0, 4]

[2, 1, 3, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

visitenode= 60

expandenode= 3

hazine= 19.0

[1, 4, 2, 0, 3]

[0, 4, 2, 1, 3]

[4, 0, 2, 1, 3]

[4, 0, 2, 1, 3]

visitenode= 60

expandenode= 3

hazine= 19.0

[2, 3, 1, 4, 0]

[4, 3, 1, 2, 0]

[4, 3, 1, 2, 0]

visitenode= 40

expandenode= 2

hazine= 19.0

[1, 4, 0, 3, 2]

[2, 4, 0, 3, 1]

[3, 4, 0, 2, 1]

[3, 4, 0, 2, 1]

visitenode= 60

expandenode= 3

hazine= 19.0

[4, 3, 2, 1, 0]

[4, 0, 2, 1, 3]

[4, 0, 2, 1, 3]

visitenode= 40

expandenode= 2

hazine= 19.0

و مشاهده می شود که جواب مانند همان جواب های قبلی است اما تعداد نود های مشاهده شده و اکسپند شده افزایش می یابد.

حال با استفاده از الگوریتم سردکردن تدریجی مساله را حل میکنیم و به جواب های زیر میرسیم برای هر یک از روش های سرد کردن محاسبه میکنیم و میگذاریم که الگوریتم در ماکزیمم حالت پنجاه بار به جلو حرکت کند.

ابتدا با روش اول این کار را انجام می دهیم

به جواب های زیر میرسیم

با استفاده از روش اول سرد کردن به جواب های زیر می رسیم

enter citiesenter citiesenter citiesenter cities[4, 1, 2, 3, 0]

[0, 1, 2, 3, 4]

[1, 0, 2, 3, 4]

[1, 0, 4, 3, 2]

[1, 2, 4, 3, 0]

[4, 2, 1, 3, 0]

[4, 2, 1, 3, 0]

[1, 2, 4, 3, 0]

[1, 0, 4, 3, 2]

[1, 0, 2, 3, 4]

[0, 1, 2, 3, 4]

[0, 3, 2, 1, 4]

[4, 3, 2, 1, 0]

[4, 3, 2, 1, 0]

[4, 3, 0, 1, 2]

[4, 3, 0, 1, 2]

[1, 3, 0, 4, 2]

[4, 3, 0, 1, 2]

[4, 0, 3, 1, 2]

[4, 0, 3, 1, 2]

[4, 0, 3, 1, 2]

[4, 0, 3, 1, 2]

[4, 0, 3, 2, 1]

[4, 0, 3, 2, 1]

[4, 0, 2, 3, 1]

[4, 2, 0, 3, 1]

[1, 2, 0, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

[0, 2, 1, 3, 4]

visitenode= 99

expandenode= 21

hazine= 19.0

حال با استفاده از روش دوم سرد کردن به جواب زیر می رسیم

[2, 1, 0, 3, 4]

[1, 2, 0, 3, 4]

[1, 4, 0, 3, 2]

[2, 4, 0, 3, 1]

[2, 4, 1, 3, 0]

[2, 0, 1, 3, 4]

[2, 1, 0, 3, 4]

[2, 1, 4, 3, 0]

[2, 1, 3, 4, 0]

[2, 1, 3, 4, 0]

[3, 1, 2, 4, 0]

[3, 1, 2, 4, 0]

[3, 1, 2, 4, 0]

[3, 1, 2, 4, 0]

[3, 1, 2, 4, 0]

[3, 1, 4, 2, 0]

[3, 1, 4, 2, 0]

[3, 0, 4, 2, 1]

[3, 4, 0, 2, 1]

[3, 4, 0, 2, 1]

[1, 4, 0, 2, 3]

[1, 4, 0, 2, 3]

[1, 4, 2, 0, 3]

[1, 3, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 4, 0, 2]

[3, 1, 4, 0, 2]

[3, 1, 4, 0, 2]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

[3, 1, 2, 0, 4]

visitenode= 99

expandenode= 19

hazine= 19.0

و با روش سوم به جواب زیر می رسیم

enter citiesenter citiesenter citiesenter cities[0, 1, 4, 3, 2]

[0, 1, 2, 3, 4]

[2, 1, 0, 3, 4]

[2, 1, 4, 3, 0]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

[2, 0, 4, 3, 1]

visitenode= 99

expandenode= 5

hazine= 19.0

خواسته شده است که جواب های به دست آمده را با هم مقایسه کنیم.

چیزی که مشاهده میشود این است که همه ی الگوریتم ها به جواب درست رسیده اند و جواب بهینه را پیدا کرده اند اما نکته ای که مطرح است این می باشد که تفاوت این الگوریتم ها در تعداد گره های بسط داده شده و مشاهده شده می باشد.

به طوری که برای الگوریتم تپه نوردی ساده تعداد گره های بسط داده شده برابر 2 و تعداد گره های دیده شده برابر با 40 میباشد همچنین در الگوریتم تپه نوردی تصادفی تعداد گره های دیده شده برابر با 80 و تعداد گره های بسط داده شده برابر با 4 میباشد . هم چنین در الگوریتم تپه نوردی اولین انتخاب تعداد گره های مشاهده شده برابر با 3 میباشد اما ققط یک گره را بسط داده است . هم چنین در روش های مختلف سرد کردن تدریجی در روش اول تعداد گره های بسط داده شده برابر با 21 و در روش دوم تعداد گره های بسط داده شده برابر با 19 و در روش سوم تعداد گره های بسط داده شده برابر با19 میباشد

حال مساله دوم را حل میکنیم .

این مساله بسیاری از توابع problem قبلی را ندارد اما چند تابع اضافه دارد که در این جا همه ی توابع جدید و قدیمی را توضیح می دهیم .

در ابتدا تابع initial را بررسی می کنیم

**def** \_\_init\_\_(self):  
 number = int(input("enter number of knapsak"))  
 self.number=number  
 self.waights = np.zeros(shape=(number))  
 self.values=np.zeros(shape=(number))  
 s = input("enter waights")  
 data=s.split()  
 **for** i **in** range(0,number):  
 self.waights[i]=int(data[i])  
 f = input("enter values")  
 data=f.split()  
 **for** i **in** range(0,number):  
 self.values[i]=int(data[i])  
 capasity = int(input("enter capasity of knapsak"))  
 self.capasity=capasity

در این تابع ورودی ها را میگیریم و ذخیره می کنیم .

در این مساله یک تابع جدید به نام random\_state تعریف میکنیم که یک استیت رندم تولید میکند و از آن برای تولید استیت های اولیه که در این مساله بیست تا هست استفاده می کنیم.

**def random\_state**(self):  
 a=[]  
 **for** i **in** range(0,self.number):  
 r=random.randint(0,1)  
 a.append(r)  
  
 y=node(a,**None**,1)  
  
 **return** y

هم چنین در این مساله تابع heuristic را برابر همان چیزی که در صورت سوال گفته شد تعریف میکنیم با این تفاوت که اگر در یک حالت وزن اجسام بیش از گنجایش مورد نظر شد به آن ارزش صفر نسبت می دهیم .

**def heuristic**(self,node1):  
 h=0  
 w=0  
 **for** i **in** range(0,self.number):  
 **if**((node1.state)[i]==1):  
 h=h+self.values[i]  
 w=w+self.waights[i]  
 **if**(w>self.capasity):  
 **return** 1  
 **else**:  
 **return** h

هم چنین تابع cross\_over را نیز در این مساله داریم که برای ترکیب دو والد و ایجاد یک فرزند از آن استفاده میکنیم و این کار را بر اساس تعداد برش ها n که در بالا معرفی شده است انجام می دهیم .

**def cross\_over**(self,node1,node2):  
 new=[]  
 n=1  
 s=set()  
 **while** len(s)<n:  
 s.add(random.randint(1,self.number-2))  
  
 a=list(s)  
  
 r=0  
  
 **for** i **in** range(0,self.number):  
 **if**(i **in** a ):  
 r=r+1  
 **if**((r%2)==0):  
 new.append((node1.state)[i])  
  
 **else**:  
 new.append((node2.state)[i])  
  
 y = node(new, **None**, 1)  
  
 **return** y

علاوه بر این تابع جهش را نیز داریم که از آن برای جهش دادن یک فرزند استفاده می شود . برای این منظور از تابع mutation استفاده می کنیم.

**def mutation**(self,node1):  
 p=(1/self.number)  
 a=[]  
 **for** i **in** range(0,self.number):  
 a.append((node1.state)[i])  
 **for** i **in** range(0,self.number):  
 ran=random.uniform(0,1)  
 **if**(ran<(1/p)):  
 **if**(a[i]==0):  
 a[i]=1  
 **else**:  
 a[i]=0  
 y = node(a, **None**, 1)  
 **return** y

تابع بالا برای این منظور بر اساس احتمال بیتی ساخته شده است .

حال به این شکل به سوالات خواسته شده بر اساس اجرای الگوریتم ژنتیک پاسخ می دهیم .

قسمت الف:

جواب را ران می کنیم وبه خروجی های زیر میرسیم

best= 38.0

worst= 1

mean= 25.25

best= 38.0

worst= 0

mean= 28.85

best= 38.0

worst= 1

mean= 29.95

best= 38.0

worst= 1

mean= 25.85

best= 38.0

worst= 1

mean= 23.75

best= 35.0

worst= 1

mean= 26.35

best= 35.0

worst= 1

mean= 24.55

best= 38.0

worst= 1

mean= 25.05

best= 38.0

worst= 14.0

mean= 27.35

best= 38.0

worst= 1

mean= 24.1

نمودار را برای چیزهای خواسته شده به دست می آوریم

برای حالت میانگین نیز نمودار را رسم میکنیم.

برای بدترین حالت نیز نمودار زیر به دست می آید

نکته مهمی که وجود دارد این است که الگوریتم را براساس روش کتاب پیاده سازی کردیم که با روش گفته شده در اسلایدها متفاوت است بنابراین نمودارهای به دست آمده این گونه شده اند هم چنین چیزی که مشاهده میشود میانگین با یک شیب بسیار کند به صورت کلی در حال افزایش است.

قسمت ب :

جهش را افزایش می دهیم و به یک دو و نیم ام می رسانیم.

و به جواب های زیر می رسیم.

best= 38.0

worst= 1

mean= 25.95

best= 37.0

worst= 1

mean= 18.35

best= 38.0

worst= 1

mean= 22.1

best= 38.0

worst= 1

mean= 25.85

best= 38.0

worst= 1

mean= 23.5

best= 38.0

worst= 1

mean= 23.15

best= 38.0

worst= 14.0

mean= 29.6

best= 38.0

worst= 14.0

mean= 30.35

best= 38.0

worst= 1

mean= 32.4

best= 38.0

worst= 14.0

mean= 33.5

مشاهده می شود که میانگین جواب های بدست آمده بهتر میشود .

حال احتمال جهش را کاهش میدهیم و به یک چهلم میرسانیم.

best= 38.0

worst= 0

mean= 18.6

best= 38.0

worst= 0

mean= 17.35

best= 38.0

worst= 0

mean= 25.05

best= 38.0

worst= 1

mean= 29.55

best= 38.0

worst= 1

mean= 30.15

best= 38.0

worst= 12.0

mean= 32.5

best= 38.0

worst= 24.0

mean= 34.5

best= 38.0

worst= 24.0

mean= 37.3

best= 38.0

worst= 38.0

mean= 38.0

best= 38.0

worst= 38.0

mean= 38.0

مشاهده میشود که میانگین باز هم افزایش می یابد اما همگرایی زمانش بیشتر طول میکشد.

حال تاثیر افزایش جمعیت را بر این مساله بررسی میکنیم.

جمعیت را از بیست به پنجاه می رسانیم و به جواب زیر میرسیم.

best= 38.0

worst= 1

mean= 23.8

best= 37.0

worst= 1

mean= 26.46

best= 37.0

worst= 0

mean= 27.04

best= 37.0

worst= 1

mean= 28.18

best= 37.0

worst= 1

mean= 26.94

best= 37.0

worst= 0

mean= 30.46

best= 37.0

worst= 1

mean= 32.56

best= 37.0

worst= 1

mean= 31.28

best= 37.0

worst= 1

mean= 31.96

best= 37.0

worst= 1

mean= 31.88

مشخص است که همگرایی الگوریتم در زمان بیشتری انجام میشود