گزارش تمرین دوم

آرمان غفارنیا 99243056

بخش simulated annealing

یک کلاس برای موجودات تعریف میکنیم که دو پارامتر value و fitness دارد که یکی تعداد پرانتزی که این موجود true میکند را نگه می دارد و یکی نشان می دهد به ازای هر متغیر چه مقدار true یا false ای داریم

```
class answer :
    def __init__(self,value,fitness):
        self.value = value
        self.fitness = fitness
```

ابتدا از فایل cnf میخوانیم (محتوایی که باید به دست آوریم در خط چهارم هستند و پس از آنها پرانتز ها آمده)

```
c
c SAT instance in DIMACS CNF input format.
c
p cnf 100 429
26  -87  -91  0
4  48  -91  0
24  35  53  0
```

```
def read ():
    read_f = open("input.cnf", "r")
    text = read_f.read()
    list_row = text.split("\n")
    detail_line = list_row[3].split(" ")
    global var_numbers
    var_numbers = int(detail_line[2])
    global p_numbers
    p_numbers = int(detail_line[3])
    first_line = 4
    last_line = p_numbers + first_line
    global paranteses
    for i in range(first_line , last_line):
        in_parantes = list()
        temp = list_row[i].split("  ")
        for j in range(0 , 3):
            in_parantes.append(int(temp[j]))

        paranteses.append(in_parantes)
```

به کمک مقادیر خوانده شده از فایل تعداد پرانتز ها و تعداد متغیر ها را به دست می آوریم

سپس به کمک split پرانتز ها را در یک لیست قرار می دهیم که هر کدام یک لیست از مقادیر داخل پرانتز است

```
def creat_answer():
    global first_population
    x = list()
    for j in range(0 , var_numbers):
            temp = random.randint(1,10)
            print(temp)
            if temp < 5 :
                x.append(False)
            if temp >= 5 :
                x.append(True)
    fit = solver(x)
    val = answer(x, fit)
    first_population = val
```

تابع بالا اقدام به تولید موجود اول به صورت تصادفی میکند به این صورت که عدد تصادفی بین 1 و 10 تولید میکند و بسته به این که از 5 بزرگتر است یا خیر مقدار درست یا نادرست به ازای آن متغیر قرار میدهد

و پس از محاسبه fitness موجود را در first_population قرار میدهد

```python
def solver(member):
    fitness_score = 0
    for i in paranteses :
        f = -1;
        for j in i:
            if j > 0:
                if member[j-1] == True and f == -1:
                    fitness_score += 1
                    f = 1;
            if j < 0:
                if  member[-j-1] == False and f== -1:
                    fitness_score += 1
                    f=1;
    return fitness_score
```

تابع بالا اقدام به محاسبه fitness(تعداد پرانتز های true شده) میکند به این صورت که مجموعه value ها را دریافت میکند و با یک for تو در تو اقدام به بررسی داخل پرانتز ها میکند و اگر یکی از آنها ترو باشد(یا نقیض آن فالس باشد) یکی به فیتنس اضافه میکند

```
read()
T = 875
creat_answer()
xlist = []
ylist = []
for i in range(50000):
    xlist.append(i)
    value_temp = first_population.value



    r = random.randint(0, var_numbers -1)
    r2 = random.randint(0, var_numbers-1)
    value_temp[r] = not value_temp[r]
    value_temp[r2] = not value_temp[r2]


    new_fitness = solver(value_temp)
```

این بخش main کد است که از فایل می خواند اقدام به تولید اولین موجود می کند و یک مقدار اولیه به T میدهد

سپس در یک حلقه با تعداد چرخش بالا که جواب مناسب را پیدا کند :

موجود بعدی را می سازیم و تصمیم میگیریم که جایگزین شود یا نه

به این صورت که با عوض کردن تصادفی 2 تا از value های موجود اول موجود جدید را میسازیم و شایستگی جدید آن را ذخیره میکنیم

```
    if new_fitness > first_population.fitness :

        first_population = answer(value_temp, new_fitness)
    else:

        r2 = random.uniform(0, 1)


        x = (first_population.fitness - new_fitness)

        check = math.exp(-(x)/T)

        if r2 < check :
            first_population = answer(value_temp, new_fitness)

    ylist.append(first_population.fitness)

    T = T * 0.97


matplotlib.pyplot.scatter(xlist,ylist)
matplotlib.pyplot.show()
```

اگر فیتنس جدید از قبلی بزرگتر باشد که قطعا موجود جایگزین می شود ولی اگر بهتر نباشد در صورتی جایگزین می شود که عدد رندوم تولید شده از فرمول آبکاری کوچکتر باشد

برای جایگزینی از تابع سازنده موجود با value جدید و fitness جدید می سازد

و هر بار دما به میزان کمی کاهش می یابد

خروجی :

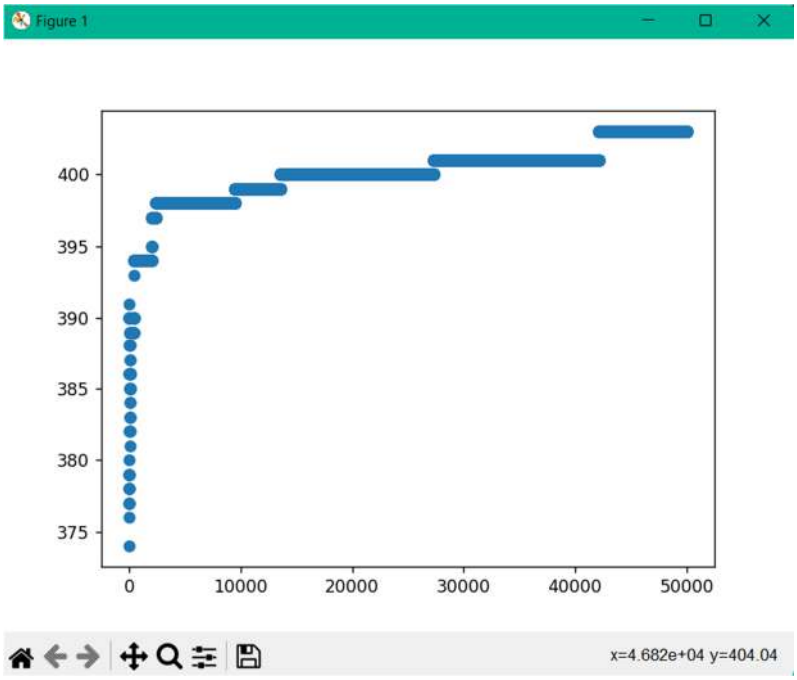
Figure 1
400
395
390
385
380
375
0
10000
20000
30000
40000
50000
x=4.682e+04 y=404.04

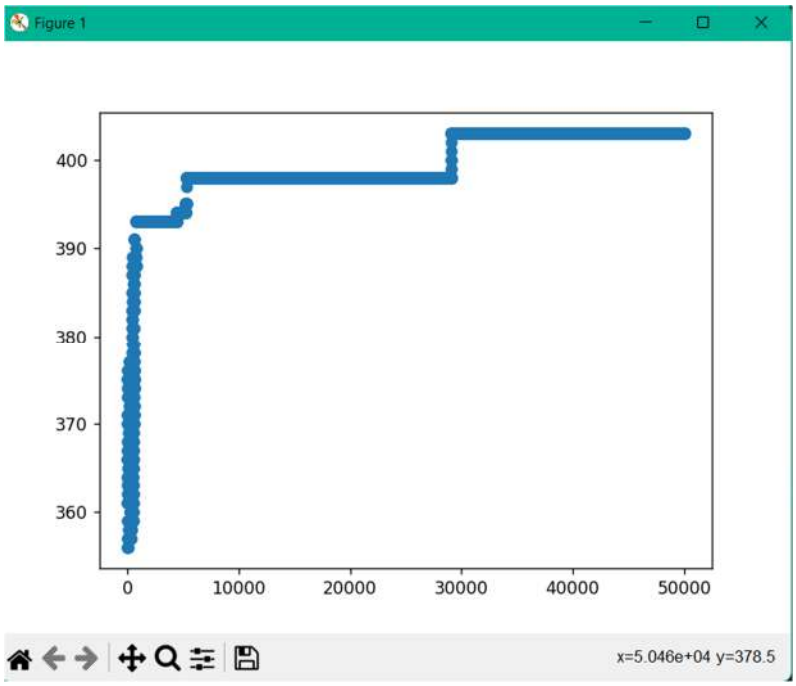
Figure 1
400
390
380
370
360
0
10000
20000
30000
40000
50000
x=5.046e+04 y=378.5

بخش genetic

در این بخش تابعی که جمعیت اولیه را تولید میکند به جای یکی 12 تا موجود تصادفی ایجاد میکند و به یک لیست اضافه می کند

```
def creat_answer():
    global first_population
    for i in range(0,first_population_number):
        x = list()
        for j in range(0 , var_numbers):
            temp = random.randint(1,10)
            print(temp)
            if temp < 5 :
                x.append(False)
            if temp >= 5 :
                x.append(True)
        fit = solver(x)
        val = answer(x, fit)
        first_population.append(val)
```

سایر بخش ها همانند بخش بالا است

```
read()
creat_answer()
xlist = []
ylist = []
templist = list()
for i in range(2000):
    xlist.append(i)

    templist = list()

    print([first_population[i].fitness for i in range(first_population_number) ])

    first_population.sort(key = lambda e:e.fitness)

    print([first_population[i].fitness for i in range(first_population_number) ])

    for j in range(6):
        u = first_population[len(first_population) - j -1 ]
        templist.append(u)
```

این بخش main کد است که از فایل می خواند اقدام به تولید اولین موجودات میکند

سپس در یک حلقه با تعداد چرخش بالا که جواب مناسب را پیدا کند :

یک templist داریم که در هر بار چرخش حلقه 6 موجودی که فیتنس بهتری دارند را به عنوان والد انتخاب میکند

برای این کار باید لیست موجودات اولیه را برحسی فیتنس مرتب کنیم و از آخر لیست بخوانیم

```
print([templist[i].fitness for i in range(6) ])
first_population.clear()
first_population.extend(templist)


for j in range(3):
    crossoverlist = list()
    for k in range(2):
        rand = random.randint(0, len(templist)-1)
        crossoverlist.append(templist[rand])

    cros1 = answer(crossoverlist[0].value, 0)
    cros2 = answer(crossoverlist[1].value, 0)
    crossoverpoint = random.randint(5, 95)
    for k in range(crossoverpoint,100):
        temp = cros1.value[k]
        cros1.value[k] = cros2.value[k]
        cros2.value[k] = temp
```

جمعیت بعدی ما را این 6 تا والد و فرزندان آنها تشکیل می دهد پس لیست جمعیت را خالی کرده و والد ها را به آن اضافی میکنیم

سپس 3 مرتبه و هر بار 2 والد را به صورت تصادفی انتخاب میکنیم که به کمک بازترکیبی خطی 2 بچه تولید میکنند

به این صورت که از یک نقطه تصادفی به بعد جای value های آنها در آرایه عوض میشود

```
        rand = random.randint(0, 99)


        cros1.value[rand] = not cros1.value[rand]
        cros2.value[rand] = not cros2.value[rand]



        cros1.fitness = solver(cros1.value)
        cros2.fitness = solver(cros2.value)
        first_population.append(cros1)

        print([first_population[i].fitness for i in range(len(first_population)) ])

        first_population.append(cros2)

        print([first_population[i].fitness for i in range(len(first_population)) ])


    print("==================================")

    best_pop = []
    for l in range(len(first_population)):
        fit = solver(first_population[l].value)
        best_pop.append(fit)

    ylist.append(max(best_pop))
```

با تولید 2 مقدار تصادفی اثدام به جهش دادن آن اندیس آرایه value از بچه ها میکنیم

سپس فیتنس بچه ها را محاسبه میکنیم و به کمک تابع سازنده موجود object آن ها را میسازیم و به جمعیت بعدی اضافه میکنیم

در آخر هر بار حلقه کلی هم بهترین فیتنس جمعیت را نشان میدهیم تا بتوانیم مقایسه کنیم

خروجی :

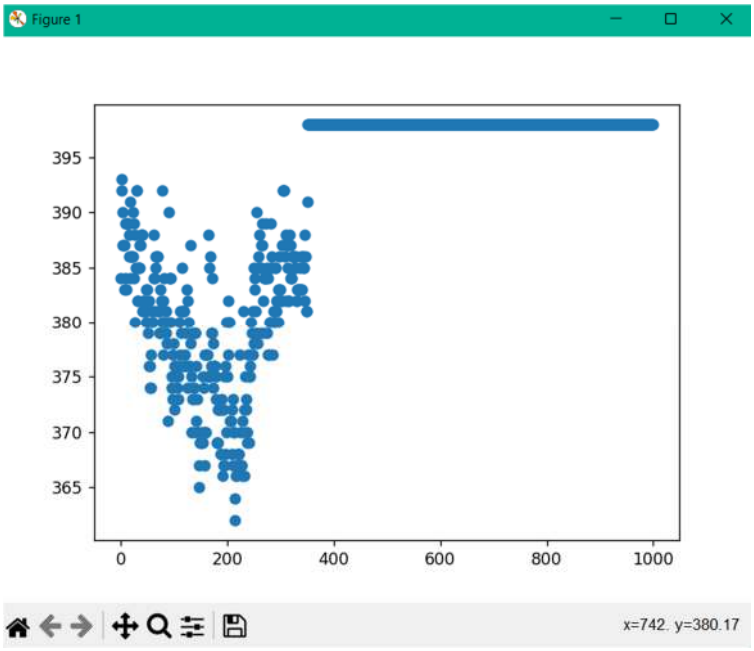
Figure 1
395
390
385
380
375
370
365
0
200
400
600
800
1000
x=742. y=380.17

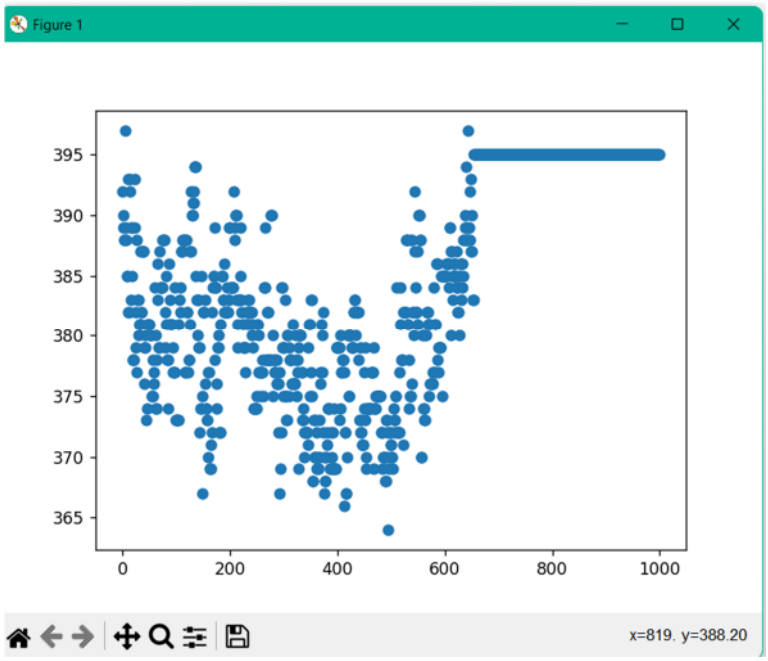
Figure 1
395
390
385
380
375
370
365
0
200
400
600
800
1000
x=819. y=388.20

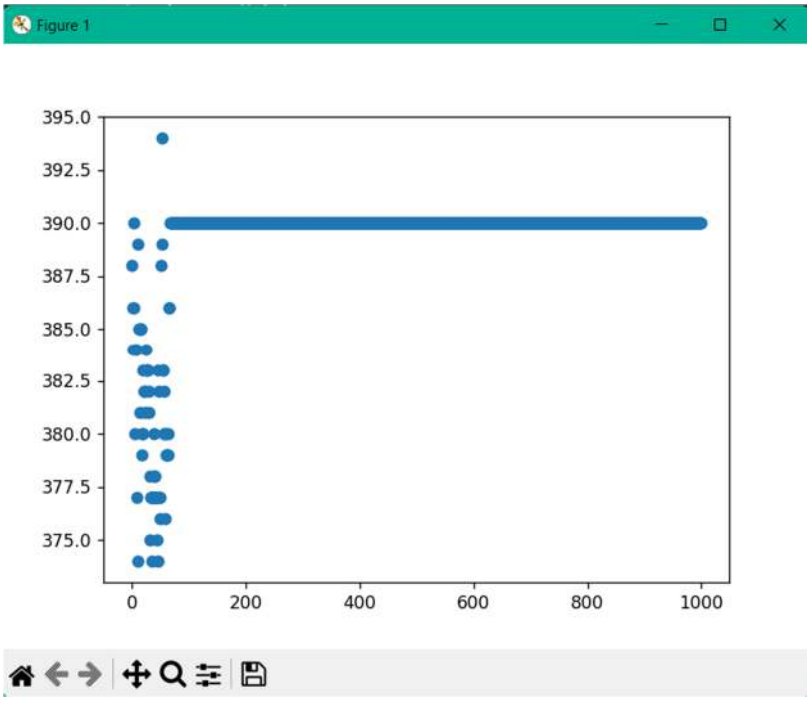
Figure 1
395.0
392.5
390.0
387.5
385.0
382.5
380.0
377.5
375.0
0
200
400
600
800
1000