HW5 Report

Fundamentals of Computational Intelligence

Niki Nezakati
98522094
Spring 2022

(Q1

۱.۱. هر کروموزوم را به صورت یک vector با طول n+1 در نظر می گیریم که n تعداد نود های گراف است که اینجا 14 است. نود های0 تا n به ترتیب در vector قرار می گیرند و مقدار هر خانه برابر عددی از 1 تا k است که نشان دهنده شماره رنگی است که آن خانه با آن رنگ شده است. برای برآورده کردن upper bound برای مسئله، k را یکی بیشتر از بیشترین درجه نود ها در نظر می گیریم که اینجا مقدار k برابر 8 خواهد بود. برای تولید جمعیت 6 کروموزوم به صورت رندوم مقدار دهی می کنیم:

[2 6 5 3 1 4 8 4 6 7 2 3 1 5 3]
[5 8 4 2 7 1 3 1 5 2 5 7 8 4 1]
[1 2 3 4 5 6 7 8 1 2 3 4 5 6 7]
[2 4 5 3 7 8 1 4 2 5 3 2 5 8 5]
[5 4 2 8 3 1 2 3 1 5 6 7 3 2 2]
[4 1 2 7 8 4 5 6 3 1 2 3 7 3 8]

۲.۱. برای محاسبه ی fitness به این صورت عمل می کنیم که به ازای هر یال که نود های دو سر آن رنگ یکسانی دارند مقدار penalty = 1 را در نظر می گیریم و fitness کروموزوم حاصل جمع این penalty ها خواهد بود. $fitness = \Sigma penalty(i, j)$

[2 6 5 3 1 4 8 4 6 7 2 3 1 5 3]
fitness=penalty(2,13)+penalty(3,14)+penalty(4,7)+penalty(11,14)= 4

[5 8 4 2 7 1 3 1 5 2 5 7 8 4 1]
fitness=penalty(2,13)= 1

[1 2 3 4 5 6 7 8 1 2 3 4 5 6 7]
fitness= 0

[2 4 5 3 7 8 1 4 2 5 3 2 5 8 5]
fitness=penalty(12,14)= 1

[5 4 2 8 3 1 2 3 1 5 6 7 3 2 2]
fitness=penalty(4,7)= 1

[4 1 2 7 8 4 5 6 3 1 2 3 7 3 8]
fitness=penalty(0,5)= 1

۳.۱. با استفاده از elitist strategy عضو هایی با fitness بهتر را برای مرحله بعد انتخاب میکنیم.
[1 2 3 4 5 6 7 8 1 2 3 4 5 6 7]
fitness= 0

[2 4 5 3 7 8 1 4 2 5 3 2 5 8 5]
fitness= 1

[5 4 2 8 3 1 2 3 1 5 6 7 3 2 2]
fitness= 1

از cross-over به این صورت استفاده میکنیم که قسمتی رندوم از یک parent برداشته و در child جایگذاری میکنیم و قسمت های باقی مانده را با parent دوم پر میکنیم. Mutation هم با جایگذاری اعداد با کم کردن آنها از k به دست می آید.

Parent 1= [1 2 3 4 5 6 7 8 1 2 3 4 5 6 7]
Child 1= [xxxx 5 6 7 8 1 2 3 4 xxx]
Child 2= [1 2 3 xxx 7 8 1 2 xxxx 7]

Parent 2=[2 4 5 3 7 8 1 4 2 5 3 2 5 8 5]
Parent 3=[5 4 2 8 3 1 2 3 1 5 6 7 3 2 2]
Child 3= [xx 2 8 3 1 xxxx 6 7 3 2 2]

Child 1= [2 4 5 3 5 6 7 8 1 2 3 4 5 8 5]
Child 2= [1 2 3 3 7 8 7 8 1 2 3 2 5 8 7]
Child 3= [2 4 2 8 3 1 1 4 2 5 6 7 3 2 2]

.۴.۱

Parent 1= [1 2 3 4 5 6 7 8 1 2 3 4 5 6 7]
fitness= 0

Parent 2=[2 4 5 3 7 8 1 4 2 5 3 2 5 8 5]
fitness= 1

Parent 3=[5 4 2 8 3 1 2 3 1 5 6 7 3 2 2]
fitness= 1

Child 1= [2 4 5 3 5 6 7 8 1 2 3 4 5 8 5]
fitness=penalty(2,4)+penalty(12,14)= 2

Child 2= [1 2 3 3 7 8 7 8 1 2 3 2 5 8 7]
fitness=penalty(2,3)+penalty(9,11)= 2

Child 3= [2 4 2 8 3 1 1 4 2 5 6 7 3 2 2]
fitness=penalty(0,2)+penalty(2,13)= 2

جمعیت جدید از نظر fitness نسبت به جمعیت اولیه ضعیف تر شده است چون وجود یال بین نود ها فارق از ترتیب قرارگیری آن ها در vector است و cross-over کردن در مراحل اولیه عملکرد مفید را کاهش میدهد.

(Q2

بله میتوان از GP استفاده کرد. مجموعه های Functions و Terminal ها را به صورت زیر تعریف می کنیم:

F={+,-,*,/,sin,cos} , T= {R ∪ xi} ∪ {R ∪ zi}

جمعیت اولیه به اندازه 500 تولید می کنیم و درخت را هم به صورت ramped half and half ایجاد می کنیم. برای fitness تابع زیر را در نظر می گیریم:

$$fitness = \Sigma(\hat{y} - y)^2$$

که در آن $\hat{y}$ جواب بدست آمده توسط کروموزوم است و y جواب اصلی تابع است. از cross-over با احتمال 0.95 به صورت جایگذاری دو زیر درخت رندوم از parent ها در child و از mutation با احتمال 0.05 به صورت جایگزنی یک زیردرخت با درختی که رندوم generate شده است، استفاده می کنیم. تعداد 1000 نسل در نظر میگیریم و iteration ها را تا جایی ادامه می دهیم که :

- به تعداد نسل های مورد نظر پیش رویم.
- به مقدار خطای کمتر از 0.0001 برسیم . $|\hat{y} - y| < 0.0001$

(Q3

با توجه به این که مورچه اول از مسیر کوتاه تر و مورچه دوم از مسیر طولانی تر می رود، پس مورچه اول سریع تر به غدا می رسد و موقع برگشت با احتمال بیشتری همان مسیر کوتاه که از آن آمده است را انتخاب می کند چون آن مسیر دارای pheromone بیشتری است. زمانی که مورچه دوم به غذا می رسد موقع انتخاب مسیر برگشت، چون مسیر طولانی یک واحد pheromone و مسیر کوتاه تر کمی کمتر از 2 واحد pheromone دارد، احتمال انتخاب مسیر کوتاه تر برای بازگشت بیشتر است. در صورت انتخاب نشدن مسیر کوتاه در این iteration ، در iteration های بعدی هم همانند این مرحله، pheromone موجود در مسیر کوتاه تر بیشتر تقویت شده و در نتیجه احتمال انتخاب مسیر کوتاه تر بیشتر می شود.

(Q4

ابتدا هر عضو را برابر با عدد x در نظر میگیریم که میتواند در رنج 10 تا 10- باشد و به تعداد1000 جمعیت را initialize می کنیم:

```
13 def initialize_population(pop_size=1000):
14     global population
15     population = []
16     for i in range(pop_size):
17         population.append(random.uniform(-10, 10))
18
```

سپس برای محاسبه fitness، هر عضو را در معادله قرار می دهیم و مقدار بدست آمده را در لیست population می گذاریم. پس از sort کردن این لیست، اولین عضو ما بهترین candidate است.

```
14 def fitness(x):
15     return abs(168 * x ** 3 - 7.22 * x ** 2 + 15.5 * x - 13.2)
16
17
18 def sort_by_fitness():
19     global population
20     population.sort(key=fitness)
21
```

برای تولید نسل های جدید از cross-over به این صورت استفاده می کنیم که به صورت رندوم و weighted average جمعیت مرتب شده، از یک جفت والد یک جفت فرزند تولید می کنیم:

```
23 def reproduce(parent1, parent2, range=10):
24     midpoint = random.randrange(1, range)
25     child1 = (parent1 * midpoint + parent2 * (range - midpoint)) / range
26     child2 = (parent2 * midpoint + parent1 * (range - midpoint)) / range
27     return child1, child2
28
29 def crossover(c_range=10):
30     global population
31     for i in range(len(population) // 2):
32         x = population[2 * i]
33         y = population[2 * i + 1]
34         child1, child2 = reproduce(x, y, c_range)
35         population[2 * i] = child1
36         population[2 * i + 1] = child2
```

برای mutation هم با احتمال 10% مقدار 0.1 را به هر عضو اضافه یا کم می کنیم:

```
def mutate(alpha):
    global population
    for i in range(len(population)):
        if random.random() < alpha:
            population[i] += random.uniform(-0.1, 0.1)

```

در آخر زمانی که fitness اولین عضو جمعیت (بهترین کاندید) کمتر از 0.000001 شود الگوریتم را متوقف میکنیم و بهترین کاندید را خروجی می دهیم که اینجا مقدار 0.369 خروجی ما می شود.

```
def GA(pop_size=1000, threshold=0.000001, crossover_range=10, alpha=0.1):
    initialize_population(pop_size)
    sort_by_fitness()
    generation = 0
    while fitness(population[0]) > threshold:
        crossover(crossover_range)
        mutate(alpha)
        sort_by_fitness()
        generation += 1
    return population[0]



print(GA())


```

```
0.36928775702973315
```

(Q5

برای حل این مسئله به کمک الگوریتم ant colony ابتدا یک کلاس برای هر task و یک کلاس برای هر machine پیاده سازی میکنیم:

```python
class Task:
  def __init__(self, time_demand: float, machine_demand: float):
    self.time_demand = time_demand
    self.machine_demand = machine_demand

class Machine:
  def __init__(self, machine_id: int, time_supply: float, time_velocity: float, machine_supply: float, machine_
    self.id = machine_id
    self.time_supply = time_supply
    self.time_velocity = time_velocity
    self.machine_supply = machine_supply
    self.machine_capacity = machine_capacity
```

کلاس ACOscheduler را برای الگوریتم پیاده سازی میکنیم. در ابتدا بردار جواب اولیه -1 است، سپس ماشین مربوطه را با توجه به pheromone یکی یکی انتخاب میکنیم و بنابراین می توانیم یک بردار جواب جدید تولید کنیم:

```python
class ACOscheduler:
  def __init__(self, tasks, machines, population_number=100, iterations=500):
    self.tasks = tasks
    self.machines = machines
    self.task_num = len(tasks) # The number of tasks
    self.machine_number = len(machines) # Number of machines
    self.population_number = population_number # Population number
    self.iterations = iterations
    # The pheromone of the machine representing the task selection
    self.pheromone_phs = [[100 for _ in range(self.machine_number)] for _ in range(self.task_num)]
    self.best_pheromone = None

# Generate a new solution vector
  def gen_pheromone_jobs(self):
    ans = [-1 for _ in range(self.task_num)]
    node_free = [node_id for node_id in range(self.machine_number)]
    for let in range(self.task_num):
      ph_sum = np.sum(list(map(lambda j: self.pheromone_phs[let][j], node_free)))
      test_val = 0
      rand_ph = np.random.uniform(0, ph_sum)
      for node_id in node_free:
        test_val += self.pheromone_phs[let][node_id]
        if rand_ph <= test_val:
          ans[let] = node_id
          break
    return ans
```

برای محاسبه fitness، میزان استفاده از هر resource را در هر ماشین محاسبه میکنیم، سپس انحراف استاندارد را برای همه ماشین ها به دست می آوریم و از مجموع انحراف استاندارد نرخ بهره برداری هر resource به عنوان درجه loading balance استفاده می کنیم که هر چه کوچکتر، متعادل تر است. از آنجا که fitness به عنوان بزرگتر بهتر تعریف می شود، بنابراین عکس نتیجه ارزیابی به عنوان fitness در نظر گرفته می شود.

```
# Evaluate the current solution vector
  def evaluate_particle(self, pheromone_jobs: List[int]) -> int:
    time_util = np.zeros(self.machine_number)
    machine_util = np.zeros(self.machine_number)

    for i in range(len(self.machines)):
      time_util[i] = self.machines[i].time_supply
      machine_util[i] = self.machines[i].machine_supply

    for i in range(self.task_num):
      time_util[pheromone_jobs[i]] += self.tasks[i].time_demand
      machine_util[pheromone_jobs[i]] += self.tasks[i].machine_demand

    for i in range(self.machine_number):
      if time_util[i] > self.machines[i].time_velocity:
        return 100
      if machine_util[i] > self.machines[i].machine_capacity:
        return 100
    for i in range(self.machine_number):
      time_util[i] /= self.machines[i].time_velocity
      machine_util[i] /= self.machines[i].machine_capacity

    return np.std(time_util, ddof=1) + np.std(machine_util, ddof=1)


  # Calculate Fitness
  def calculate_fitness(self, pheromone_jobs: List[int]) -> float:
    return 1 / self.evaluate_particle(pheromone_jobs)
```

برای آپدیت کردن pheromone ها، pheromone مسیر با مورچه ها دو برابر می شود، در غیر این صورت تبخیر به نصف می رسد.

```
  # Update pheromones
  def update_pheromones(self):

    for i in range(self.task_num):
      for j in range(self.machine_number):
        if j == self.best_pheromone[i]:
          self.pheromone_phs[i][j] *= 2
        else:
          self.pheromone_phs[i][j] *= 0.5
```

در نهایت با مقدار دهی اولیه به ماشین ها و تسک ها، الگوریتم را اجرا می کنیم. فیتنس با بهبود یافتن به 1.983983586616108 رسیده و ثابت می شود و تسک ها به صورت زیر به ماشین ها assign می شوند.

```
Task : 0  Put it on the machine  1
Task : 1  Put it on the machine  2
Task : 2  Put it on the machine  0
Task : 3  Put it on the machine  0
Task : 4  Put it on the machine  0
Task : 5  Put it on the machine  1
Task : 6  Put it on the machine  0
Task : 7  Put it on the machine  1
Task : 8  Put it on the machine  0
Task : 9  Put it on the machine  2
```

Resources:

https://pythonmana.com/