به نام پروردگار زیبایی

محمد عرفان زارع زردینی 98411432

هوش مصنوعی تمرین سری 4 تئوری

دکتر مزینی

سوال 1)

طبق مطالعات انجام شده، مورچه ها به سیگنال های محرک محیطی واکنش نشان می دهند و این محرک و واکنش بر بقیه مورچه ها هم اثر می گذارد و به این محرک، stigmergy گویند. آزمایشات نشان می دهد که مورچه ها از positive feedback برای یافتن کوتاهترین مسیر از کلونی به غذا استفاده می کنند. به این فرایند اتوکاتالیز گویند. پس از گذشت t واحد زمانی ، اگر m1 حشره از پل اول و m2حشره از پل دوم بروند، احتمال آنکه حشره بعدی از پل اول برود مطابق فرمول زیر هست: (پارامتر kو h ، پارامتر مورد نیاز برازش هست .)

$$p_1(m+1) = \frac{(m_1+k)^h}{((m_1+k)^h + (m_2+k)^h)}$$

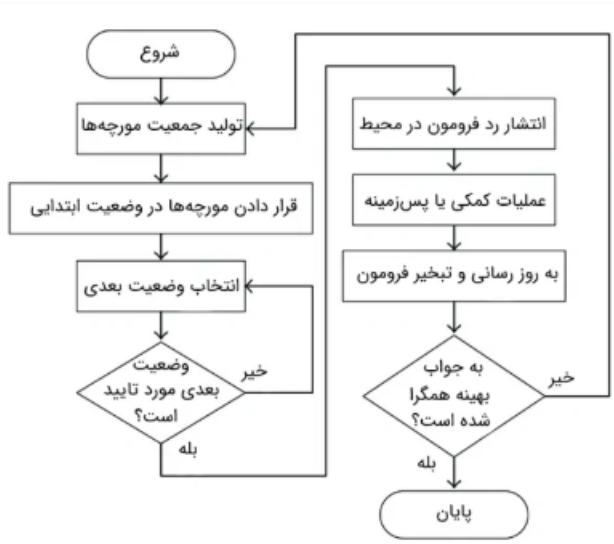
احتمال عبور از پل دوم مکمل همین مقدار هست.طبق شبیه سازی های انجام شده، مقدار kو hبه ترتیب 20و2 رام قدار مناسب دونسته.

برای شبیه سازی با الگوریتم کلونی مورچه ها، ابتدا باید مدل سازی انجام شود برای یک مساله ترکیباتی که می شود:

$$A\ model\ P = (S, \Omega, f)$$

که S فضای جستجو ، امگا متغییر های مساله و f نیز تابع هدف است که باید کمینه شود.

مراحل انجام الگوریتم بدین گونه هست که ابتدا جمعیت حشرات تولید شده و به صورت رندوم در وضعیت اولیه‌شان قرار می‌گیرند. سپس باید وضعیت بعدی حشرات مشخص شود. این عمل تا جایی ادامه پیدا می کند که یک وضعیت قابل قبول برای حشرات پیدا شود. حال که این وضعیت تعیین شد باید رد فرومون در محیط انتشار یابد . سپس عملیات کمکی را انجام و در فرومون را بروز رسانی و تبخیر می‌کنیم. حال بررسی می‌کنیم که ایا جواب بهینه مان همگرا شده است یا خیر . اگر شده بود به الگوریتم پایان می دهیم و اگر نه به قسمت تولید جمعیت حشرات باز می‌گردیم.



به روزرسانی فرومون به شکل زیر است:

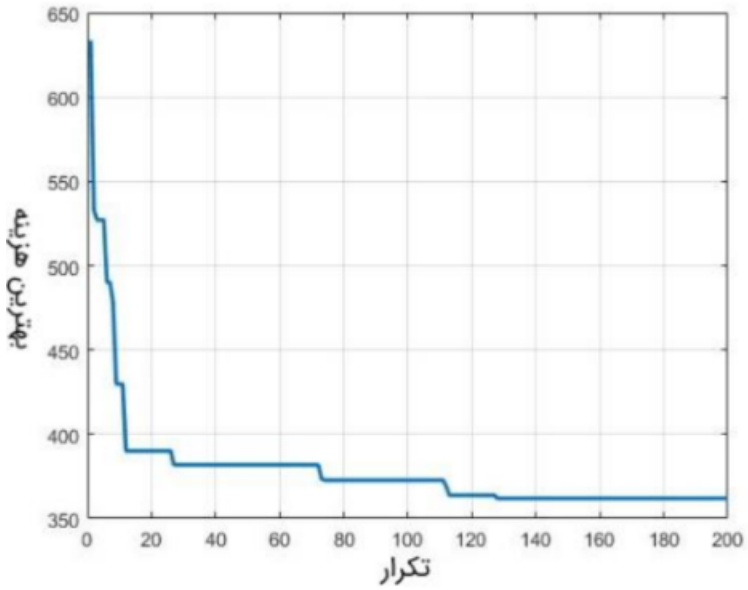
$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho \sum_{s \in S_{upd} | c_{ij} \in s} F(s)$$

فرایند مورد بحث سبب می شود که با افزایش مقدار فرومون های متناظر با مورچه ها در بهترین مسیرهای موجود به سمت بهینه قراردارند، دارای برازندگی بالایی اند. بدین صورت بقیه مورچه ها به مسیر موردنظر همگرا می شوند

اگر بخواهیم این الگوریتم را به شکل دیگر توضیح بدهیم، میتوان گفت که فرض کنیم یکسری موررچه دارند در دو مسیر مستقل درحال حرکت به سمت غذا هستند ، در حالی که هرکدام رد یا فرومون به جای میگذارند که بشکل طبیعی با گذشت زمان متلاشی می شود. مورچه هایی که به شکل رندوم کمترین مسیر را نسبت به بقیه برای رسیدن به غذا پیدا میکنند ، سریعتر به کلونی برمیگردنند وردی که در مسیر برجای میذارند سبب تقویت اثر می شود وسبب می شود مورچه ها بیشتر در این مسیر ها حرکت کنند. این اتفاق سبب می شود که کم کم این مسیر به مسیر پرتردد تر تبدیل شود و تقویت رد فرومون بیشتر میشود.

از پر بحث ترین مسائل حل شده با این الگوریتم ، می توان به مسئله فروشنده دوره گرد اشاره نمود .سوال این گونه هست که ما یکسری شهر داریم و باید تنها با یک بار عبور از شهر ها ، کمترین مسیر را یافت. تاثیر الگوریتم مورچه ها در این سوال بدین گونه هست که در حرکتمیان شهر ها ، مولفه های جواب کاندید اند یعنی که از حرکت از مسیر 1به 2 مولفه جواب کاندید $C_{12}=C_{21}$ مسئله هستند. گراف ساختاری $G=(V,E)$ از طریق ایجاد رابطه میان شهرها با رئوس v در گراف ساختاری می باشد.

مورچه ها گویی از یک نقطه شروع به حرکت و رفتن به شهر های دیگر می کنند و در این راه ، مسیر های پیموده شده را به خاطر سپرده و به محض آنکه تمام شهر ها توسط یک موزچه پیموده شد، شروع ، جواب کاندید تولید می شود.



منبع: https://blog.faradars.org/ant-colony-optimization/

سوال2)

در حل سوال ابتدا وزن ها و مقادیر (value) در نظر بگیریم.

```
w = np.array([2,4,1,3,5,1,7,4])
val = np.array([30,10,20,50,70,15,40,25])
thrshd_knackpack = 25
print('Item num \t Weight \t Values')
for i in range(len(w)):
    print('{0} \t \t {1}\t \t {2}\n'.format(i, w[i], val[i]))
```

خروجی به شکل زیر می شود:

Item Number	Weight	Value
0	2	30
1	4	10
2	1	20
3	3	50
4	5	70
5	1	15
6	7	40
7	4	25

حال جمعیت اولیه برای سوال ایجاد و در ادامه با توجه به جمعیت، مقادیر نسل های بعدی جمعیت هم تولید می شود.

اعدادجمعیت به شکل تصادفی تولید می شوند .

```
init_poulate = np.random.randint(2, size = (10,8))
init_poulate = init_poulate.astype(int)
num_gen = 50
print('init population: \n{}'.format(init_poulate))
```

تعداد نسل ها برای حل سوال 50 در نظر میگیریم. در هر نسل 10 عدد تصادفی تولید می شو که شامل 8رقم باینری است.

نمونه جمعیت تولیدی به شکل زیر است:

```
[[0 1 1 1 1 1 0 1]
 [0 0 0 1 0 0 0 1]
 [1 0 0 1 1 1 1 0]
 [0 0 1 1 1 1 1 0]
 [0 0 0 1 1 0 1 1]
 [1 0 1 1 1 1 0 0]
 [0 0 0 1 1 0 0 1]
 [0 0 0 1 0 0 1 0]
 [1 0 1 1 0 0 1 1]
 [0 1 0 0 1 1 0 1]]
```

حال توابع موردنیاز برای تولید جمعیت پیاده سازی شده اند. توابع mohasebe_fitness و selectionوظیفه انتخاب و مرتب کردن بهترین فیتنس را دارد. Mutation که جهش را تولید نموده و تابع optimizer برای بهینه سازی هست.

تابع crossoverپیاده سازی شده ولی در حل از روش دیگر استفاده می شود.

```python
def mohasebe_fitness(w, val, population, threshold):
    fitness = np.empty(population.shape[0])
    for i in range(population.shape[0]):
        val_sum = np.sum(population[i] * val)
        w_sum = np.sum(population[i] * w)
        if w_sum <= threshold:
            fitness[i] = val_sum
        else :
            fitness[i] = 0
    return fitness.astype(int)
```

تابع fitness برای جمع کردن مقادیر ارزش هر ایتم هست.تابع بالا هم فیتنس را محاسبه می کند.

```python
def selection(fitness, num_parents, population):
    fitness = list(fitness)
    parents = np.empty((num_parents, population.shape[1]))
    for i in range(num_parents):
        max_fitness_idx = np.where(fitness == np.max(fitness))
        parents[i,:] = population[max_fitness_idx[0][0], :]
        fitness[max_fitness_idx[0][0]] = -999999

    return parents
```

تابع با لا عناصر جمعیت را با توجه به فیتنس مرتب می کند.

```
def mutation(parents):
    mutations = np.empty((parents.shape))
    rate_mute = 0.5
    for i in range(parents.shape[0]):
        random_value = rd.random()
        mutations[i,:] = parents[i,:]
        if random_value > rate_mute:
            continue

        randval1 = randint(0,parents.shape[1]-1)
        mutations[i, randval1] = 0 if mutations[i, randval1] == 1 else 1

    return mutations
```

تابع بالا نیز جهش را پیاده میکند.ابتدا عدد رندوم انتخاب میشه. اگر مقدار رندوم از حد استانه ی احتمالی که برای جهش در نظر گرفته بودیم بیشتر بود، رقم رندوم دیگر تولید می شود که ماکزیمم طولش باید تعداد بیت های هر یک از عناصر جمعیت ما باشد.

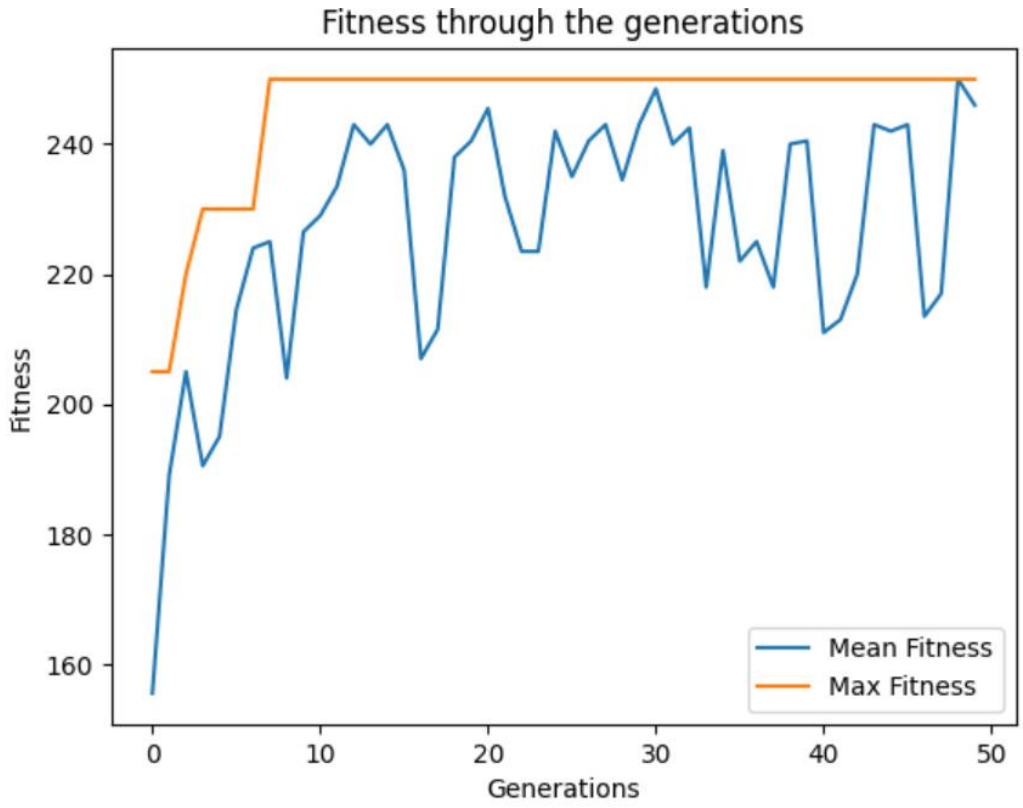
پس از تولید رقم مان، در عناصری که باید جهش رخ بده ، از ان عنصر تصادفی به بعد بیت هایش قرینه می شود.

```
def optimizer(w, val, population, pop_size, num_gen, threshold):
    parameters, fitness_history = [], []
    num_parents = int(pop_size[0]/2)
    num_offsprings = pop_size[0] - num_parents
    for i in range(num_gen):
        fitness = mohasebe_fitness(w, val, population, threshold)
        fitness_history.append(fitness)
        parents = selection(fitness, num_parents, population)
        mutations = mutation(parents)
        population[0:parents.shape[0], :] = parents
        population[parents.shape[0]:, :] = mutations

    print('Last generation: ')
    print(population)
    last_gen_fitness = mohasebe_fitness(w, val, population, threshold)
    print('Fitness of the last generation: ')
    print(last_gen_fitness)
    max_fitness = np.where(last_gen_fitness == np.max(last_gen_fitness))
    parameters.append(population[max_fitness[0][0],:])
    return parameters, fitness_history
```

تابع بالا هم توابع تعریف شدرو فرا میخواند وبا تولید نسل جدید به جواب بهینه ی مد نظرمان برسیم. جواب نهایی می شود به شکل زیر:

```
Last generation:
[[1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 0 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]
 [1 0 1 1 1 1 1 1]]
Fitness of the last generation:
[250 250 250 250 250 180 250 250 250 250]
The optimized parameters for the given inputs are:
[array([1, 0, 1, 1, 1, 1, 1, 1])]
```



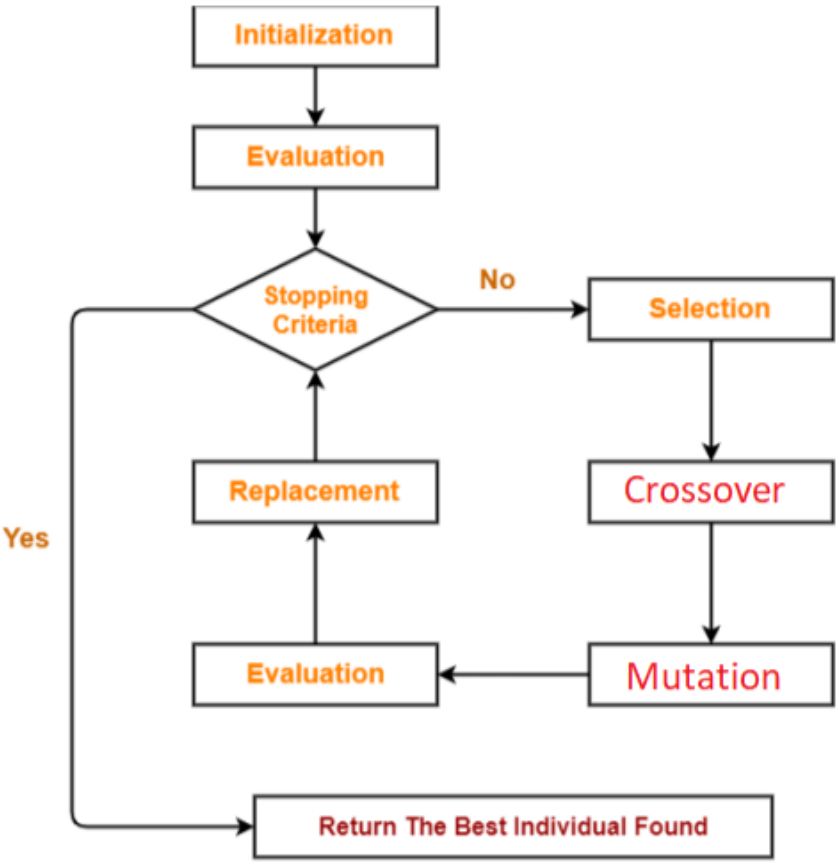

```
(50, 10)
```

منبع حل سوال :

https://arpitbhayani.me/blogs/genetic-knapsack#:~:text=For%20our%20knapsack%20example%2C%20we,the%20next%20steps%20of%20evolution.

[Genetic Algorithm: Part 3 — Knapsack Problem | by Satvik Tiwari | Koderunners | Medium]

سوال3)


Initialization
Evaluation
Stopping Criteria
No
Selection
Crossover
Mutation
Evaluation
Replacement
Yes
Return The Best Individual Found

۹۸۴۱۱۴۳۳ بسم الله الرحمن مرجان زارعی

سوال (۳)

مرحله‌ی اول crossover و مرحله‌ی دوم mutation می‌باشد.

$f(x)=x^2 \quad 0 \le x \le 30$

ابتدا لازم است تا شیوه‌ی نمایش x را مشخص کنیم. حداکثر مقدار x، ۳۰ است پس به ۵ بیت برای نمایش نیاز داریم.

مقادیر اولیه جمعیت نسل اول هم به صورت 13=001101 و 24=011000 و 8=001000 و 16=010000 داده شده‌اند.

x	binary	y	$\frac{f_i}{\sum f}$	Cumulative
۱۳	001101	۱۶۸	0.16	0.16
۲۴	011000	۵۷۶	0.54	0.70
۸	001000	۶۴	0.06	0.76
۱۶	010000	۲۵۶	0.24	1.00

در مرحله‌ی بعد باید جمعیت نسل بعد را بسازیم. ۴ عدد تصادفی بین ۰ تا ۱ تولید می‌کنیم، در صورتی که عدد میان ۰ تا ۰.۱۶ بود عدد ۱۳،
اگر بین ۰.۱۶ تا ۰.۷۰ بود عدد ۲۴، اگر بین ۰.۷۰ تا ۰.۷۶ بود ۸، اگر بین ۰.۷۶ تا ۱ بود عدد ۱۶ در جمعیت جدید
ثبت می‌شود.

فرض می‌کنیم ۴ عدد ۰.۱۵، ۰.۵۰، ۰.۷۰، ۰.۹۰ انتخاب شده‌اند. در نتیجه اعداد جدید ما ۱۳ و ۲۴ و ۲۴ و ۱۶ می‌شوند. حال باید

روی ترتیب دوتایی اعداد crossover و mutation اعمال کنیم. از cross تک‌نقطه‌ای استفاده می‌کنیم.

x	binary		random number (for crossover point)	new_b	new_x	y
۱۳	001101	(★)	۴	001100	۱۲	۱۴۴
۲۴	011000	(★)	۴	011001	۲۵	۶۲۵
۲۴	011000	(★★)	۳	011000	۲۴	۵۷۶
۱۶	010000	(★★)	۳	010000	۱۶	۲۵۶

برای نمونه جهش هم برای مثال می‌تواند بیت آخر جمعیت را روی یک هم رندوم آن مثال رقم سوم ...

جهش انجام داده و مقادیر نهایی به این شکل می‌شود ۲۵=011001 و ۱۲=001100 و ...

(★) 0011|01, 0110|00 ⇒ 001100, 011001

(★★) 011|000, 010|000 ⇒ 011000, 010000