# به نام خدا

## تمرین سری ششم
## درس مبانی هوش محاسباتی
## دکتر ناصر مزینی

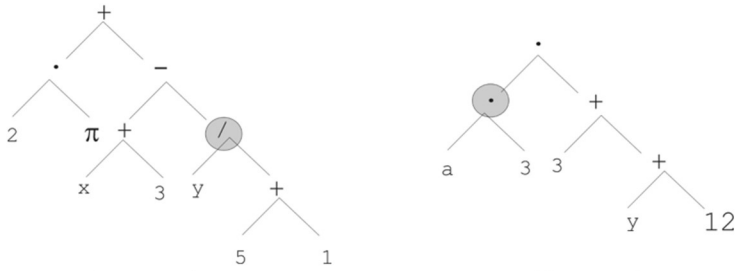

فرزان رحمانی
۹۹۵۲۱۲۷۱

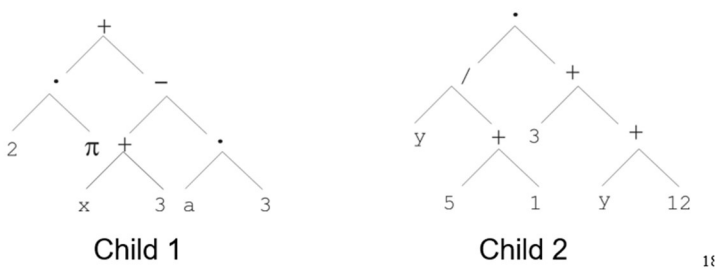

### سوال اول

۱. روند کلی کار و ساختار درخت:

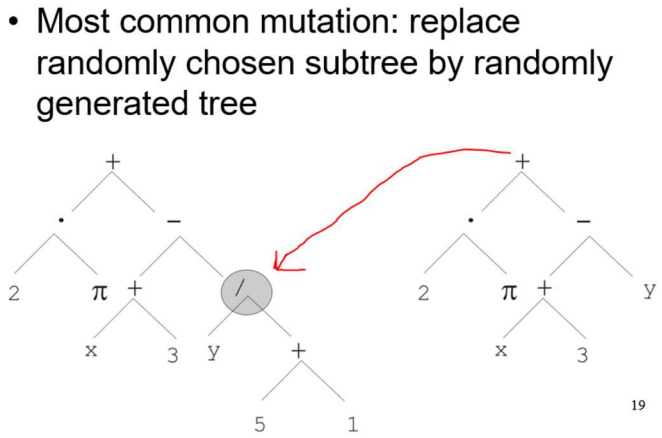
روند کلی کاری:

- ورودی کاربر: این ابزار یک سری توضیحات را از کاربر در مورد طراحی صفحه وب مورد نظر جمع آوری می کند. این ورودی ممکن است شامل موارد زیر باشد:
  - عناصر مورد نیاز (دکمه ها، فیلدهای ورودی، استایل و غیره)
  - تنظیمات چیدمان (Layout preferences)
  - الزامات عملکردی(Functionality requirements)
  - محدودیت های طراحی(Design constraints)
- جمعیت اولیه(Initial Population): این ابزار یک جمعیت اولیه تصادفی از طرح(design) های صفحه وب را تولید می کند که به صورت درخت نمایش داده می شود. هر گره در درخت می تواند یک تگ HTML مانند <button>, <input>,  و غیره را نشان دهد. شاخه ها(branches) نحوه تودرتو شدن تگ ها را تعیین می کنند.
- Fitness Evaluation: هر طرح با استفاده از یک تابع برازندگی ارزیابی می شود (در ادامه در این مورد بیشتر توضیح خواهیم داد) تا مشخص شود که چقدر نیازهای کاربر را برآورده می کند.
- Selection: طرح هایی با بالاترین fitness scores به عنوان والدین برای نسل بعدی انتخاب می شوند.
- تولید مثل(Reproduction): طرح های فرزندان از طریق عملگرهای ژنتیکی ترکیب می شوند مانند:
  - Crossover: ترکیب بخشی از دو درخت والد برای ایجاد درختان جدید(مانند اسلاید زیر)



- Mutation: اصلاح تصادفی بخش‌هایی از درخت برای معرفی تنوع(مانند اسلاید زیر)

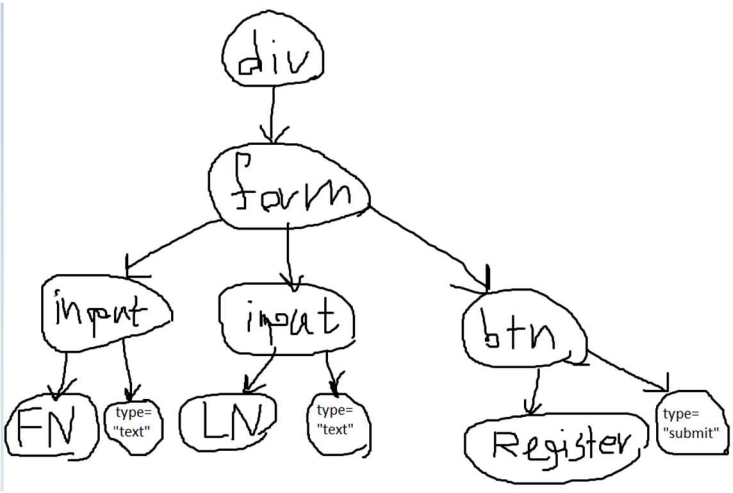



- تکرار(Iteration): مراحل Initial Population تا Reproduction برای چندین نسل تکرار می‌شوند تا طرح‌های بهتری را تکامل دهند.
- خروجی نهایی: این ابزار بهترین طراحی(ها) را به کاربر ارائه می دهد.

مثال ساختار درختی:
به عنوان مثال، ساختار درختی زیر را برای توضیح فرم ثبت نام در نظر بگیرید(برای سادگی تگ های <html> و <body> که باید در ابتدای کد html باشند صرف نظر کردیم.):

```
<div>   -> Root node (representing a container element)
   <form> -> Child node (representing a form element)
      <input type="text" placeholder="First Name"> -> Leaf node
(representing an input element)
      <input type="text" placeholder="Last Name"> -> Leaf node
      <button type="submit">Register</button> -> Leaf node
   </form>
</div>
```

این درخت سلسله مراتبی طرح بندی صفحه وب را با فرمی حاوی دو کادر ورودی برای نام و نام خانوادگی و پس از آن دکمه ای برای تکمیل ثبت نشان می دهد.




همان طور که میبینید Function Set ها شامل تگ های html و Terminal Set شامل مقادیر نهایی فیلدها و پارامترها و متن هایی که داخل تگ ها قرار میگیرند هستند می باشند.

۲. Fitness Function:

تعریف:

تابع fitness یک فرمول ریاضی است که ارزیابی می کند که طراحی صفحه وب تا چه حد نیازهای کاربر را برآورده می کند. به هر طرح امتیازی اختصاص می‌دهد و به ابزار اجازه می‌دهد مواردی را که به راه‌حل مورد نظر کاربر نزدیک‌تر هستند، انتخاب و تکامل دهد. لذا fitness را بر اساس اینکه طراحی تولید شده چقدر با توضیحات هدف ارائه شده توسط کاربر مطابقت دارد، تعریف کنیم.

مثال:

```
fitness =  w1 * (number of required elements present) +
           w2 * (layout similarity to user preferences) +
           w3 * (functionality correctness) -
           w4 * (number of violations of design constraints)
```

توضیح:

- w1، w2، w3، w4 وزن هایی هستند که اهمیت نسبی معیارهای مختلف را کنترل می کنند.
- سه عبارت اول به طرح‌هایی(Designs) پاداش می‌دهند که شامل عناصر مورد نیاز(required elements) کاربر، طرح‌بندی ترجیحی(preferred layout) کاربر و عملکرد صحیح هستند. منظور از عملکرد صحیح **HTML/CSS Validity (**ارزیابی اینکه آیا کد تولید شده معتبر است و از استانداردهای web پیروی می کند**)** است.
- آخرین عبارت، طرح هایی را که محدودیت های طراحی را نقض می کنند (به عنوان مثال، داشتن عناصر همپوشانی(overlapping elements)) جریمه می کند.
- میتوانیم با توجه به کاربرد های خاص عبارات بیشتری با وزن های گوناگون به fitness اضافه کنیم. مانند Structure Accuracy ، Scalability ، Responsivity و غیره.

دلایل انتخاب:

- معیارهای چندگانه را متعادل می کند: هم ساختار (عناصر و چیدمان: elements and layout) و عملکرد(functionality) و هم ترجیحات و محدودیت های کاربر را در نظر می گیرد.
- وزن‌های قابل تنظیم: وزن‌ها را می‌توان برای اولویت‌بندی نیازهای خاص بر اساس نیازهای کاربر سفارشی کرد.
- پتانسیل برای کارایی: اگر به طور موثر پیاده سازی شود، می تواند به طور موثر جستجو را به سمت طرح های بهتر هدایت کند.

همچنین یک روش دیگر برای تعریف تابع fitness می توانست به صورت $Fitness = no.of\ matched\ features\ /\ total\ no.of\ expected\ features$ باشد.

۳. اجرای مثال دستی:

ورودی کاربر:

«یک فرم ثبت نام شامل ۲ عدد باکس ورودی که نام و نام خانوادگی را دریافت می کند. همچنین این فرم در انتهای خود دکمه ای جهت تکمیل ثبت نام دارد.»

جمعیت اولیه (به طور تصادفی ایجاد شده، در اینجا تعداد individual ها را ۳ تا در نظر گرفتیم ولی در عمل می‌تواند مثلا ۱۰۰۰ تا باشد):

- طرح ۱(درخت ۱): فرمی با ۳ باکس ورودی و بدون دکمه (  <input> <input> <input>)
- طرح ۲: فرمی با ۱ باکس ورودی و دکمه ای با عنوان «ارسال» (  <input> <button placeholder="submit">)
- طرح ۳: فرمی با ۲ باکس ورودی و دکمه ای با عنوان «ثبت نام» (  <input> <input> <button placeholder="register">)

ارزیابی fitness:

- طرح ۱: fitness کم (دکمه ندارد، ۱ باکس ورودی اضافی دارد)
- طرح ۲: fitness متوسط (۱ باکس ورودی وجود ندارد، عنوان دکمه «ثبت نام» نیست)
- طرح ۳: fitness بالا (با تمام الزامات ورودی کاربر مطابقت دارد)

:Selection and Reproduction

- طرح ۳ و بعد از آن طرح ۲ با احتمال بیشتری به عنوان والد برای نسل بعدی انتخاب می شود.
- Crossover و mutation برای ایجاد طرح‌های جدید اعمال می‌شوند که احتمالا در طراحی ۳ باعث بهبود می‌شوند.

تکرار(Iteration):

- این روند(ارزیابی، selection، reproduction) تکرار می شود و طرح ها در طول نسل ها با نیازهای کاربر هماهنگ تر می شوند.

خروجی نهایی:

این ابزار بهترین طراحی را ارائه می دهد که باید بسیار شبیه فرم ثبت نام مورد نظر کاربر باشد. مانند ورودی زیر:

```
<html>
    <body>
        <div>
           <form>
              <input type="text" placeholder="First Name">
              <input type="text" placeholder="Last Name">
              <button type="submit">Register</button>
           </form>
        </div>
    </body>
</html>
```

اجرای دستی شامل ابزار ایجاد کد HTML/CSS است که نمایانگر فرم ثبت نام توصیف شده توسط کاربر است. کد به دست آمده باید یک طرح بندی صفحه وب با عناصر مشخص شده مطابق توضیحات ارائه شده ترتیب داده شود. سپس تابع تناسب، این خروجی را بر اساس معیارهایی که قبلاً ذکر شد، ارزیابی می کند تا دقت و کارایی آن را در برآوردن الزامات داده شده ارزیابی کند.

توجه داشته باشید که اجرای واقعی چنین ابزاری شامل برنامه‌نویسی پیچیده با استفاده از الگوریتم‌های ژنتیک، تجزیه ورودی‌های کاربر، تولید کد و ارزیابی تناسب است که به‌جای دستی، خودکار خواهد بود.

مراجع:

https://chat.openai.com/
https://bard.google.com/
https://claude.ai/chats

## سوال دوم

برای حل معادلات شامل چند جمله ای های مختلف با استفاده از الگوریتم ژنتیک با استفاده از  پایتون، باید این مراحل را دنبال کنیم:

۱. بازنمایی(Representation): راهی برای نمایش جواب های بالقوه (کروموزوم ها) برای ریشه های چند جمله ای تعریف میکنیم.
۲. مقدار دهی اولیه(Initialization): یک جمعیت اولیه از راه حل های بالقوه ایجاد میکنیم.
۳. تابع تناسب(Fitness Function): یک تابع تناسب را برای ارزیابی اینکه یک راه حل به خوبی معادله را حل می کند، تعریف میکنیم.
۴. انتخاب(Selection): افراد (راه حل) های مناسب تر را از بین جمعیت بر اساس fitness انتخاب میکنیم.
۵. ترکیب کردن(Crossover): ترکیب راه حل های انتخاب شده برای ایجاد راه حل های جدید (فرزندان).(exploration)
۶. جهش(Mutation): برای افزایش تنوع و راه حل های ابتکاری، تغییرات تصادفی را در برخی راه حل ها ایجاد میکنیم.(exploitaion)
۷. خاتمه(Termination): یک معیار توقف را تعریف میکنیم (به عنوان مثال، یافتن یک راه حل به اندازه کافی دقیق یا رسیدن به حداکثر تعداد تکرار).

در زیر تابع اصلی کد زده شده را میبینیم:

```
def genetic_algorithm(polynomial, population_size=100, generations=1000,
mutation_rate=0.01):
    population = initialize_population(population_size)
    found_root = False
    for generation in range(generations):
        float_population = [binary_to_float(solution) for solution in population]
        fitness_scores = np.array([fitness(solution, polynomial) for solution in
float_population])
        # Check if any solution is a good enough root
        best_solution = population[np.argmin(fitness_scores)]
        if fitness(binary_to_float(best_solution), polynomial) >= 100: # 0.01
difference from orginal root
            print(f"Root found in generation {generation}:
{binary_to_float(best_solution)}")
            found_root = True
            print("Value of polynomial with this solution(error)",
polynomial(binary_to_float(best_solution)))
            break
        parents = np.array([select_parents(population, fitness_scores) for _ in
range(population_size // 2)])
        # Crossover and mutation
        children = np.array([crossover(parent1, parent2) for parent1, parent2 in
parents])
        children = np.array([mutate(child, mutation_rate) for child in
children])
        # Sort parents based on fitness
        sorted_indices = np.argsort(fitness_scores) # small to big
        tmp = np.array(population)
        sorted_parents = tmp[sorted_indices]
```

```
        # Choose the top 50 parents with better fitness
        selected_parents = sorted_parents[population_size // 2:]
        # Flatten the selected parents to update the population
        population[:population_size // 2] = selected_parents.flatten()
        population[population_size // 2:] = children  # updated population
    if (not found_root): # No exact root found.
        best_answer_found = population[np.argmin(fitness_scores)]
        print("Best solution in 1000 generations:",
binary_to_float(best_answer_found))
        print("Fitness Score (1/error): ", np.argmin(fitness_scores))
        print("Value of polynomial with this solution(error)",
polynomial(binary_to_float(best_answer_found)))
```

توضیح بخش های مختلف کد مربوط به فرآیند الگوریتم ژنتیک:

۱. بازنمایی(Representation): معادلات را به صورت آرایه هایی از ضرایب معادله چند جمله ای به صورت lambda function نشان داده می دهیم. همچنین راه حل ها (کروموزوم ها) را به صورت binary string نشان میدهیم. همچنین باید floating point را هم به خوبی در نظر بگیریم و مدیریت کنیم به ویژه هنگام crossover و mutation.(سر این قسمت خیلی باگ خوردم و وقت ازم گرفته شد). همچنین برای اینکه تعداد بیت ها قبل و بعد از floating point ثابت باشند و در هنگام crossover و mutation به مشکل نخوریم به صورت یک متغیر global آنها را تعریف میکنیم.(با این کار اعداد را تقریب میزنیم و طبیعاً جواب دقیق را نمیتوانیم پیدا کنیم.)
۲. مقداردهی اولیه(Initialization): یک جمعیت اولیه از راه حل های بالقوه با ضرایب تصادفی در یک محدوده معین(در اینجا از بازه ۰ تا ۱۰ انتخاب کردیم چون جواب های مثال های داده شده در صورت سوال در این بازه اند) تولید می شود.
۳. تابع تناسب(Fitness Function): fitness_func برازندگی یک راه حل را بر اساس اینکه ریشه های آن معادله را برآورده می کنند محاسبه می کند (خطای کمتر به معنای برازندگی بالاتر است لذا آن را به صورت معکوس مقدار تابع با استفاده از یک کروموزم تعریف میکنیم همچنین برای پایداری عددی از یک epsilon در مخرج استفاده میکنیم تا به مشکل مخرج صفر نخوریم).
۴. انتخاب(Selection): راه حل ها برای نسل بعدی بر اساس fitness score آنها، با استفاده از توزیع احتمال وزن شده بر حسب تناسب آنها(normalized fitness) انتخاب می شوند.
۵. ترکیب کردن(Crossover): راه حل های منتخب برای ایجاد راه حل های جدید با هم ترکیب می شوند و فرآیند crossover ژنتیکی را تقلید می کنند.
۶. جهش(Mutation): تغییرات تصادفی را در برخی راه حل ها برای حفظ تنوع ژنتیکی ایجاد می کند.
۷. خاتمه(Termination): الگوریتم با رسیدن به آستانه تناسب اندام مشخص(خطای کمتر از ۰/۰۱ یا تناسب بالای ۱۰۰) یا پس از حداکثر تعداد نسل(۱۰۰۰ نسل) پایان می یابد.

توجه: این یک الگوریتم ژنتیک ساده شده برای ریشه یابی چند جمله ای است. برای معادلات پیچیده‌تر یا دقت بالاتر، به بهبودها و تنظیمات بیشتری نیاز است، مانند تنظیم پارامترها، پیاده‌سازی نخبه‌گرایی(elitism) یا مدیریت چندین متغیر در معادلات.

توضیحات بیشتر راجع به قسمت های مختلف کد با استفاده از کامنت و markdown در فایل q2.ipnyb در پیوست آمده است.

نتایج حاصل از کد زده شده برای حل معادلات با استفاده از الگوریتم ژنتیک(کد من) و حل معادلات با استفاده از numpy در ادامه آمده است و همان طور که میبینید بعد از ۱۰۰۰ نسل و با population size ۱۰۰ الگوریتم ژنتیک توانسته است که جواب ها را با تقریب خوبی(خطای کمتر از ۰/۰۱) پیدا کند. با استفاده از نسل و با population size های بیشتر و استفاده از بیت های بیشتر در binary string representation میتوانیم جواب هایی با تقریب بهتر پیدا کنیم:

```
(base) E:\uni\7th term\fundamentals of computational intelligence\HWs\HW6>python q2.py

Testing Polynomial 1:
Root found in generation 113: 2.0
Value of polynomial with this solution(error) 0.0

Testing Polynomial 2:
Best solution in 1000 generations: 7.46875
Fitness Score (1/error):  87
Value of polynomial with this solution(error) 0.0322265625

Testing Polynomial 3:
Best solution in 1000 generations: 1.21875
Fitness Score (1/error):  97
Value of polynomial with this solution(error) 0.0330810546875

Testing Polynomial 4:
Best solution in 1000 generations: 0.359375
Fitness Score (1/error):  62
Value of polynomial with this solution(error) 0.07074310302734332

(base) E:\uni\7th term\fundamentals of computational intelligence\HWs\HW6>python find_roots_numpy.py
[2.]
[7.46410162 0.53589838]
[1.21372896+0.j         0.01813552+0.45348417j 0.01813552-0.45348417j]
[-0.15984864+0.4152082j -0.15984864-0.4152082j  0.35851449+0.j       ]

(base) E:\uni\7th term\fundamentals of computational intelligence\HWs\HW6>
```

همچنین برای Representation از gray code استفاده کنیم تا فاصله اعداد یکی باشند و دقت بهتری داشته باشیم ولی ما اینجا به شکل معمولی پیاده سازی کردیم و به جواب رسیدیم.

## Representation of Variables

Consider example problem,
where 127 is 01111111 and 128 is 10000000

The smallest fitness change requires change in every bit !

- Gray coding has representation such that adjacent values vary by a single bit
- Different alphabets possible

### Gray Coding

**Table 3.1** Gray Codes and Binary Codes for Integers 0–15

| Integer | Binary code | Gray code |
|---|---|---|
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0011 |
| 3 | 0011 | 0010 |
| 4 | 0100 | 0110 |
| 5 | 0101 | 0111 |
| 6 | 0110 | 0101 |
| 7 | 0111 | 0100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |
| 10 | 1010 | 1111 |
| 11 | 1011 | 1110 |
| 12 | 1100 | 1010 |
| 13 | 1101 | 1011 |
| 14 | 1110 | 1001 |
| 15 | 1111 | 1000 |

To get Gray code from binary code, the leftmost bit is the same, then
$G_i = XOR(B_i, B_{i-1})$ for i>=2, where $G_i$ is the $i^{th}$ Gray code bit, Bi is $i^{th}$ binary bit.$_{34}$

مراجع:

https://chat.openai.com/

## سوال سوم

حل این مسئله با استفاده از الگوریتم ژنتیک را به طور خلاصه شرح میدهیم:

ساختار ژنوم:

- یک راه حل(ژنوم) را به صورت لیستی از ۳۶ عدد صحیح از ۱ تا ۳۶ نشان داده میشود(آرایه ۱ بعدی به طول ۳۶ که شامل همه اعداد از ۱ تا ۳۶ است) که نشان دهنده اعداد در مربع ردیف به ردیف است. (مشابه TSP)
- مثال: [۱، ۲، ۳، ۴، ۵، ۶، ۷، ۸، ...، ۳۶]

تابع برازندگی:

- fitness score بر اساس تعداد سطرها و ستون ها با اعداد زوج و فرد مساوی تعیین میشود.
- آن را به یکی از دو صورت زیر میتوانیم تعریف کنیم:

روش اول:

$$fitness = 12 - (number\ of\ rows\ with\ unequal\ odd/even + number\ of\ columns\ with\ unequal\ odd/even)$$

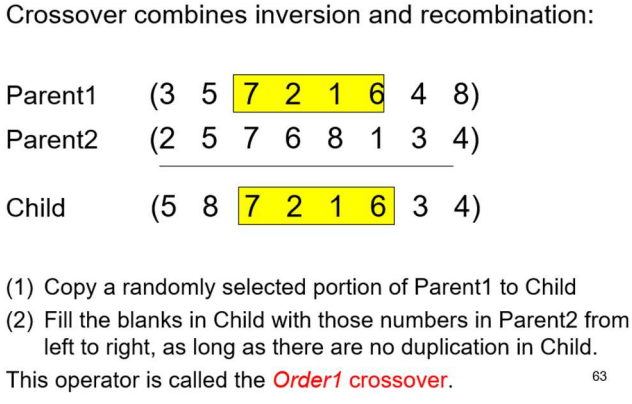
روش دوم:

$$For\ each\ row\ and\ column, calculate\ the\ difference\ between\ the\ number\ of\ odd\ and\ even\ numbers.$$
$$total\ difference = Sum\ of\ absolute\ across\ all\ rows\ and\ columns$$
$$fitness = 1 / (1 + total\ difference)$$

- fitness score های بالاتر نشان دهنده راه حل های بهتر هستند. Fitness score کمتر(بدتر) به معنای عدم برابری بیشتر بین سطر و ستون هاست.

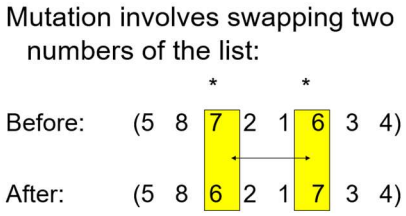
پارامتر ها:

- اندازه جمعیت(num of individuals): از ۱۰۰-۲۰۰ نفر شروع میکنیم.
- روش selection: میتوانیم از Tournament selection یا roulette wheel selection استفاده کنیم.
- روش crossover: نوع خاصی از two-point crossover بنام *Order1* crossover. چون باید همه اعداد از ۱ تا ۳۶ در راه حل فقط یکبار تکرار شوند.




- نرخ crossover: ۰/۷-۰/۹ (احتمال ترکیب ژن های دو parent).
- نرخ mutation(احتمال جهش در هر کروموزوم در جمعیت): ۰/۰۱-۰/۱ (احتمال تغییر تصادفی یک ژن).

- روش جهش(mutation): Swap mutation(تعویض دو عدد تصادفی در آرایه)




- حداکثر نسل: ۱۰۰۰-۵۰۰۰ (حداکثر تعداد تکرارهایی که الگوریتم اجرا می کند).

مراحل الگوریتم ژنتیک:

۱. مقداردهی اولیه:
یک جمعیت اولیه از راه حل های تصادفی (لیست های ۳۶ عددی) ایجاد میکنیم.
۲. ارزیابی برازندگی(fitness function):
fitness score را برای هر فرد در جامعه محاسبه میکنیم.
۳. انتخاب(Selection):
افراد با fitness score بالاتر را به عنوان والدین برای نسل بعدی انتخاب میکنیم.
۴. تولید مثل(Reproduction):
- از crossover برای ترکیب ژن های والدین منتخب استفاده میکنیم و فرزندان جدیدی ایجاد میکنیم.
- اعمال جهش(mutation) برای تغییر تصادفی برخی از ژن ها در فرزندان و معرفی تنوع.

۵. تکرار(Iteration):
مراحل ۲-۴ را برای چندین نسل (حداکثر نسل) تکرار میکنیم.
۶. خروجی نهایی:
بهترین راه حل یافت شده (فردی که بالاترین fitness score را دارد) آرایش مربع ۶*۶(آرایه ۳۶ عددی که در ابتدا تعریف کردیم) مورد نظر را نشان می دهیم.

توجه:

- این رویکرد الگوریتم ژنتیک راه‌حلی را ارائه می‌کند که تلاش می‌کند محدودیت‌ها را برآورده کند، اما ممکن است به دلیل ماهیت تصادفی الگوریتم ژنتیک و پیچیدگی فضای مسئله، راه‌حل بهینه را در همه موارد تضمین نکند. تنظیم پارامترها یا استفاده از روش های مختلف crossover/selection ممکن است نتایج بهتری به همراه داشته باشد. همچنین میتوانیم الگوریتم را چندین بار اجرا کنیم.

پیاده سازی:

- برای پیاده سازی میتوانیم از کتابخانه هایی مانند deap یا pymoo استفاده کنیم.

ملاحظات اضافی:

- میتوانیم اجرای استراتژی نخبه گرایی(elitism) را برای حفظ بهترین افراد در طول نسل ها در نظر بگیریم.
- برای تنظیم دقیق عملکرد الگوریتم، مقادیر پارامترهای مختلف را آزمایش کنیم.
- اگر یافتن راه حل کامل چالش برانگیز باشد، میتوانیم تابع fitness را برای پاداش دادن به راه حل های جزئی یا اولویت بندی محدودیت های خاص تغییر دهیم.

مراجع:

https://chat.openai.com/
https://bard.google.com/
https://claude.ai/chats

## سوال چهارم

در صورت سوال به هر دو الگوریتم ASO و PSO اشاره شده است اما با توجه به بحث انجام شده در گروه ما در اینجا با روش ASO به حل این مسئله می پردازیم. همچنین چون رقم آخر شماره دانشجویی من ۱ است پس باید حالت دوم را تشخیص دهیم.
( s = ( 1 % 3 ) + 1 = 1 + 1 = 2 )

**حل مسئله با روش ACO:**

Ant Colony Optimization (ACO) معمولاً برای مسائل بهینه سازی ترکیبی(combinatorial optimization) استفاده می شود و کاربرد آن برای وظایف تشخیص تصویر کمتر رایج است. با این حال، ما می توانیم سعی کنیم ACO را برای مسئله خاص سوال در تشخیص تصاویر حالت ۲ در بین ۵ تصویر داده شده تطبیق دهیم.

تعریف گراف مسئله:

- هر تصویر را به عنوان یک گره در گراف نشان میدهیم. پس ۵ گره داریم.
- هر جفت گره را با یک یال به هم وصل میکنیم. به یال ها مقادیر فرومون نسبت داده میشود.
- مقادیر فرمون روی یال ها را با یک ثابت مثبت کوچک آغاز میکنیم.

تعریف فرومون و ابتکارات(Phromones and Heuristics):

- هر تصویر را به عنوان یک گره در گراف الگوریتم ACO در نظر میگیریم. گره ها به وسیله یال ها با میزان فرومون مختلف به هم متصل می شوند.
- فرومون ها را در یال های بین گره ها (تصاویر) بر اساس شباهت آنها به حالت ۲ تعریف میکنیم.
- یک Heuristic تعریف میکنیم که دانش قبلی یا شباهت هر تصویر(معکوس اختلاف) را به حالت ۲ را نشان می دهد.

مقدار دهی اول مورچه ها(Initialize Ants):

- مورچه های مصنوعی ای ایجاد میکنیم که با انتخاب احتمالی گره بعدی بر اساس مقادیر فرمون در یال ها، گراف را طی کنند. از یک heuristic استفاده میکنیم که تصاویر نزدیک به حالت ۲ را ترجیح می دهد.(مثلا similarity یا distance از حالت ۲)
- مورچه ها را به صورت تصادفی روی تصاویر قرار میدهیم.

حرکت مورچه:

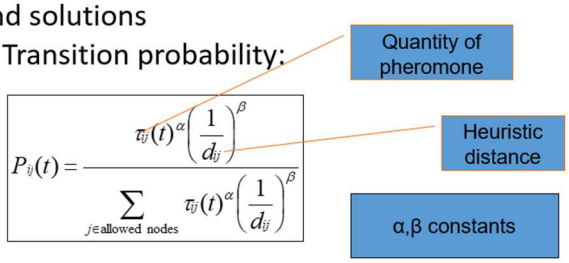
- در هر تکرار، تعدادی مورچه (مثلا N=5) روی هر گره به صورت تصادفی رها میکنیم.
- هر مورچه از گره فعلی خود به گره همسایه به صورت احتمالی حرکت می کند و احتمال آن بر اساس دو عامل زیر انتخاب می کند:
  - دنباله فرومون(سطوح فرومون): شدت فرمون بیشتر در یال باعث جذابیت بیشتر آن می شود.
  - Heuristic Information: امتیاز شباهت بین تصویر فعلی و حالت ۲ را برای هر گره همسایه محاسبه میشود. شباهت بیشتر باید احتمال انتخاب آن یال را افزایش دهد. (در فرمول میتوانیم distance را به صورت معکوس similairity در نظر بگیریم)

  میزان شباهت را به صورت زیر تعریف میکنیم:

$$similarity = 16 - number\ of\ different\ pixels in\ image\ from\ state\ 2$$
$$\max similarity = 16, \min similarity = 0$$
$$distance = \frac{1}{similarity + \varepsilon}, \varepsilon\ is\ for\ numerical\ stability\ (avoiding\ division\ by\ zero)$$
$$\varepsilon = 0.0001$$

- در واقع، مورچه ها را تشویق میکنیم تا به سمت تصاویری که بیشتر شبیه حالت ۲ هستند حرکت کنند.
- سپس مورچه ها را برای عبور از گراف و ایجاد راه حل (مسیرهای تصاویر) اعزام میکنیم.
- فرمول احتمال انتخاب در زیر آمده است که بر گرفته از اسلاید های درسی است:

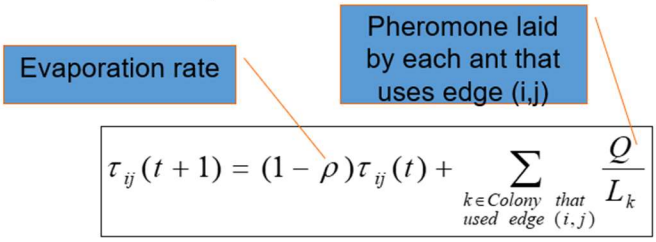
- Find solutions
  - Transition probability:




به روز رسانی فرمون:

- سطوح فرمون را در یال ها بر اساس کیفیت جواب های یافت شده توسط مورچه ها به روز میکنیم.
- برای جلوگیری از رکود، مقداری فرمون را از تمام یال ها تبخیر میکنیم.
- بعد از اینکه همه مورچه ها مسیر خود را کامل کردند، مسیرهای فرومون را در یال هایی که طی کرده اند به روز میکنیم. فرومون بیشتری در یال هایی که مورچه ها از آنها بازدید می کنند جمع میشود که:
  - متعلق به تصاویر حالت ۲ هستند.
  - از گره هایی با شباهت بیشتر به حالت ۲ عبور کردند.

- Pheromone update




تکرار(Iteration) و همگرایی:

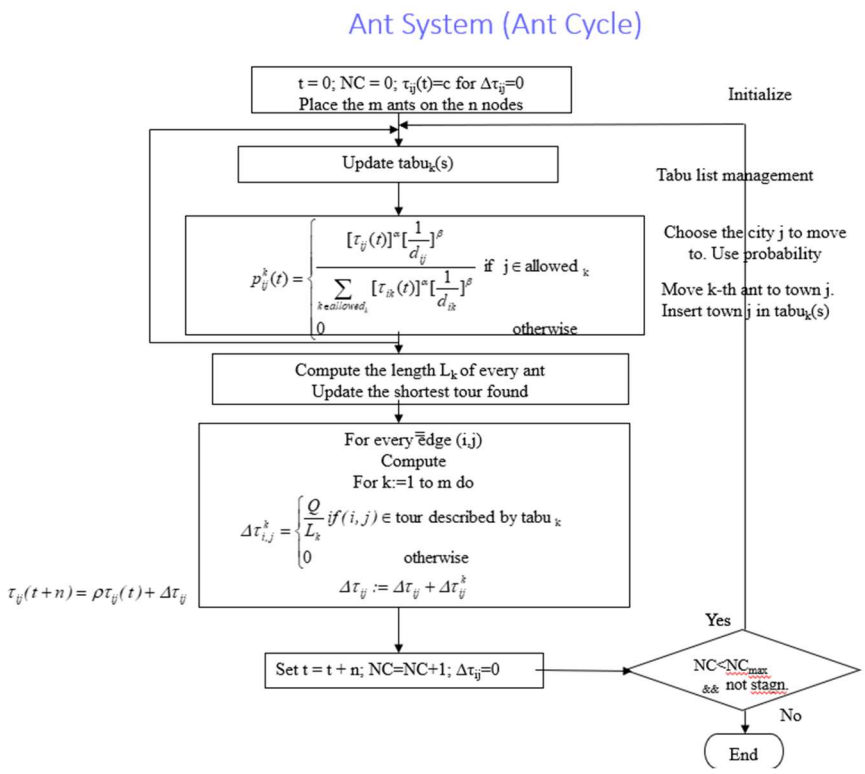
- مراحل قبل(از بعد از initialization) را برای تعداد معینی تکرار، تکرار میکنیم تا همگرا شویم.
- میانگین امتیاز شباهت مسیرهای مورچه را نظارت میکنیم. زمانی که امتیاز تثبیت شد یا به یک آستانه از پیش تعیین شده رسید الگوریتم را متوقف میکنیم.
- تصاویر با میانگین امتیازات شباهت بالا در تکرار نهایی به عنوان حالت ۲ شناخته می شوند.

استخراج بهترین جواب:

- بهترین راه حل پیدا شده توسط مورچه ها را استخراج میکنیم، که مربوط به تصاویری است که به عنوان حالت ۲ تشخیص داده شده است و بیشترین شباهت و کمتری تفاوت را با حال ۲ دارد.
- پس از تکرار نهایی، یال هایی با بالاترین سطح فرومون به تصاویر شبیه به حالت ۲ متصل می شوند. لذا آن گره که یال های متصل به آن بیشترین سطح فرمون را دارند بیشترین شباهت را به حالت ۲ دارد.

توجه: الگوریتم‌های سنتی ACO اغلب برای مسائلی به کار می‌روند که راه‌حل‌ها را می‌توان به عنوان جایگشت عناصر نشان داد، اما در اینجا با ساختارهای تصویر سروکار داریم. ممکن است لازم باشد پارامترهای الگوریتم ACO را تنظیم کنیم. علاوه بر این، ممکن است بخواهیم سایر تکنیک‌های تشخیص تصویر یا رویکردهای یادگیری ماشین را برای عملکرد بهتر در این مسئله خاص بررسی کنیم.

فلو چارت کلی در ادامه آمده است:




توضیحات بیشتر راجع به خود الگوریتم ASO:

- بهینه سازی کلونی مورچه ها یک الگوریتم فراابتکاری(metaheuristic algorithm) است که به طور خاص از چگونگی پیدا کردن کوتاه ترین مسیر بین کلنی خود و منبع غذایی مورچه ها الهام گرفته شده است. از مورچه‌های مصنوعی استفاده می‌کند که در حین حرکت، مسیرهای فرومون را بر جای می‌گذارند، که بر مسیری که مورچه‌های آینده طی می‌کنند تأثیر می‌گذارد.
- در ACO، مورچه ها راه حل ها را به صورت احتمالی می سازند و مسیرهای فرومون بر اساس کیفیت راه حل های یافت شده به روز می شوند. با گذشت زمان، تجمع فرمون منجر به راه حل های بهتری می شود.
- برخی از مزایای ACO این است که همه کاره، قوی و مناسب(versatile, robust, and well-suited) برای مسائل بهینه سازی گسسته مانند مشکل فروشنده دوره گرد است.
- برخی از محدودیت‌ها این است که می‌تواند کند همگرا شود، پارامترها نیاز به تنظیم دقیق دارند، و اگر مسیرهای فرومون تبخیر نشوند، خطر رکود در طول زمان وجود دارد.
- اجزای کلیدی الگوریتم ACO عبارتند از ردیابی های فرمون، اطلاعات اکتشافی، ساخت راه حل، و به روز رسانی فرمون(pheromone trails, heuristic information, solution construction, and pheromone update).

مراجع:

https://chat.openai.com/
https://bard.google.com/
https://claude.ai/chats

# پایان