دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

گزارش درس کارشناسی
هوش مصنوعی

# گزارش پروژه‌ی ششم

## هستی برقراریان

استاد درس
مهدی قطعی

مدرس کارگاه
بهنام یوسفی‌مهر

خرداد ۱۴۰۲

# چکیده

در این پروژه، تلاش می‌کنیم تا با زبان برنامه‌نویسی python، برنامه‌ای بنویسیم که با استفاده از الگوریتم تکامل تدریجی، مسئله‌ی n-وزیر را حل کند.

# فصل اول

## مقدمه

## مقدمه

الگوریتم های تکاملی (EAs) دسته‌ای از روش‌های محاسباتی هستند که از فرآیند تکامل طبیعی الهام گرفته شده‌اند. آن‌ها معمولاً در مسائل هوش مصنوعی (AI) و بهینه‌سازی برای یافتن راه‌حل‌های تقریبی برای مسائل پیچیده یا ناشناخته استفاده می‌شوند. ایده‌ی اصلی پشت الگوریتم‌های تکاملی، استفاده از اصول داروینی انتخاب، بازتولید و جهش برای جمعیتی از راه‌حل‌های کاندید است که به طور مکرر آن‌ها را در طول زمان بهبود می‌بخشد.

یک نمای کلی از نحوه عملکرد یک الگوریتم تکاملی:

۱. مقداردهی اولیه: جمعیتی از راه‌حل‌های بالقوه، که اغلب به عنوان مجموعه‌ای از افراد نامزد نشان داده می‌شوند، به‌طور تصادفی برای شروع الگوریتم تولید می‌شوند.

۲. ارزشیابی: هر فرد در جامعه بر اساس یک تابع تناسب ارزیابی می شود که عملکرد یا کیفیت آن را در حل مشکل مورد نظر اندازه گیری می کند. تابع ارزیابی مشخص می کند که هر فرد چقدر مشکل را حل می کند.

۳. انتخاب: افراد با ارزش های ارزیابی بالاتر به احتمال زیاد به عنوان والدین برای نسل بعدی انتخاب می شوند. این فرآیند انتخاب طبیعی را تقلید می کند، زیرا افرادی که ارزیابی بهتری دارند، احتمال بیشتری برای انتقال مواد ژنتیکی خود به نسل بعدی دارند.

۴. تولید مثل: افراد منتخب تحت عملگرهای ژنتیکی مانند متقاطع و جهش قرار می گیرند. متقاطع شامل ترکیب اطلاعات ژنتیکی دو والدین برای ایجاد فرزندان است، در حالی که جهش تغییرات تصادفی کوچکی را در ماده ژنتیکی فرزندان ایجاد می کند.

۵. جایگزینی: فرزندان جدید جایگزین بخشی از جمعیت قبلی می‌شوند، معمولاً افراد با ارزش تناسب اندام پایین‌تر. این تضمین می کند که جمعیت در طول زمان به سمت راه حل های بهتر تکامل می یابد.

۶. خاتمه: الگوریتم از طریق مراحل انتخاب، بازتولید و جایگزینی به تکرار ادامه می‌دهد تا زمانی که یک شرط خاتمه برآورده شود. این شرایط می تواند حداکثر تعداد نسل، آستانه تناسب اندام مورد نظر یا محدودیت زمانی از پیش تعریف شده باشد.

از طریق این فرآیند تکراری، الگوریتم تکاملی فضای راه حل را بررسی می‌کند، به تدریج میانگین تناسب جمعیت را بهبود می‌بخشد و به سمت راه حل های بهتر همگرا می شود. ایده این است که در طول نسل‌های متوالی، الگوریتم فرآیند تکامل طبیعی را تقلید می‌کند، جایی که افراد مناسب‌تر احتمال بیشتری برای زنده ماندن و تولید مثل دارند و ویژگی‌های مفید خود را منتقل می‌کنند.

الگوریتم‌های تکاملی به دلیل توانایی‌شان در مدیریت فضاهای جستجوی پیچیده و چندوجهی، جایی که تکنیک‌های بهینه‌سازی سنتی ممکن است با مشکل مواجه شوند، شناخته شده‌اند. آنها با موفقیت برای طیف گسترده ای از مشکلات هوش مصنوعی، از جمله بهینه سازی، یادگیری ماشینی، روباتیک، و بازی کردن و غیره استفاده شده اند.

حال در این مقاله، در فصل دوم به توضیح برخی کدها پرداخته و سپس در فصل سوم، جمع‌بندی و نتیجه‌گیری را خواهیم داشت.

# فصل دوم

## توضیح کدها

# توضیح کدها

کد اول:

```
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

این چند خط، کتابخانه NumPy را برای محاسبات عددی، ماژول تصادفی برای عملیات تصادفی و کتابخانه Matplotlib را برای ایجاد نمودار وارد می‌کند.

```
def checkFitness(pop):
    fit = np.zeros((pop[:,1].size,1))
    for index, solution in enumerate(pop):
        for ia,a in enumerate(solution, start = 1):
            for ib, b in enumerate(solution[ia:(len(solution))], start = ia+1):
                # print("Index:",index,"valor de A:",a,"/",ia,"Valor de B:",b,"/",ib)
#Control
                if abs(a-b) == abs(ia-ib):
                    fit[index,0] = fit[index,0] + 1
                # if abs(a-b) == 0:
                #     fit[index,0] = fit[index,0] + 1
```

```
return fit
```

به یک پارامتر pop نیاز دارد، که فرض می شود یک آرایه numpy دو بعدی است که جمعیتی از راه حل ها را نشان می دهد.

کد تناسب هر راه حل در جامعه را بر اساس تعداد جفت عناصری که در یک مورب الگوی صفحه شطرنج قرار دارند محاسبه می کند. هر چه مقدار تناسب یک راه حل بیشتر باشد، عناصر بیشتری در آن والد روی مورب ها قرار می گیرند.

```
def order_crossover(p1, p2, size):

    def fillGene(f,p):
        for ia, a in enumerate(p):
            if a not in f:
                for ib, b in enumerate(f):
                    if b == 0 :
                        f[ib] = a
                        break
        return f
```

کد عملیات متقاطع ترتیب را با ترکیب ژن‌های دو والد (p1 و p2) برای ایجاد یک والد فرزند جدید (f) انجام می‌دهد. ژن‌های p1 به ترتیب اصلی خود در f قرار می‌گیرند، در حالی که ژن‌های باقی‌مانده از p2 پر می‌شوند و ترتیب خود را نیز حفظ می‌کنند.

```
    f1 = np.zeros(len(p1))
    f2 = np.zeros(len(p2))

    c = random.randint(0, (len(p1)-size))

    f1[c:c+size] = p1[c:c+size]
    f2[c:c+size] = p2[c:c+size]

    f1 = fillGene(f1,p2)
    f2 = fillGene(f2,p1)

    offsprings = np.vstack([f1,f2])
    return offsprings
```

این کد عملیات متقاطع سفارش را با استفاده از آرایه های numpy انجام می دهد. با کپی کردن یک بخش از ژن ها از هر یک از والدین، دو راه حل برای فرزندان ایجاد می کند و سپس موقعیت های خالی باقی مانده ژن را با ژن های والد دیگر پر می کند و ترتیب آنها را حفظ می کند. والدهای حاصل به‌صورت یک آرایه دوبعدی برگردانده می‌شوند.

```
def selection(pop, p_sel):
    sel_pool =
np.random.permutation(pop[:,1].size)[0:int(round(pop[:,1].size*p_sel))]
    bestSol = pop[sel_pool[0],:]
    for sol in sel_pool[1:len(sel_pool)]:
```

```
        if pop[sol,len(bestSol)-1] < bestSol[len(bestSol)-1]:
            bestSol = pop[sol,:]

    return bestSol
```

کد یک فرآیند انتخاب را با انتخاب تصادفی مجموعه ای از راه حل ها از جمعیت انجام می دهد و سپس به طور مکرر مقادیر برازش آنها را برای یافتن راه حلی با کمترین ارزش تناسب مقایسه می کند (با فرض اینکه مقدار تناسب کمتر نشان دهنده راه حل بهتری باشد). این بهترین راه حل انتخاب شده به عنوان خروجی تابع برگردانده می شود.

```
def swap_mutation(child,numberOfSwaps):

    for  i in range(numberOfSwaps):
        swapGenesPairs = np.random.choice(len(child), 2, replace = False)
        a = child[swapGenesPairs[0]]
        b = child[swapGenesPairs[1]]

        child[swapGenesPairs[0]] = b
        child[swapGenesPairs[1]] = a

    return child
```

این کد عملگر جهش مبادله را با انتخاب تصادفی جفت‌های ژن و جابجایی موقعیت‌های آنها به یک راه‌حل فردی (فرزند) اعمال می‌کند. این فرآیند برای تعداد مشخصی از مبادله ها (numberOfSwaps) تکرار می شود. سپس تابع فرد جهش یافته را برمی گرداند.

```
def plotCheckBoard(sol):

    def checkerboard(shape):
        return np.indices(shape).sum(axis=0) % 2

    sol = sol -1
    size = len(sol)
    color = 0.5
    board = checkerboard((size,size)).astype('float64')
    # board = board.astype('float64')
    for i in range(size):
        board[i, int(sol[i])] = color

    fig, ax = plt.subplots()
    ax.imshow(board, cmap=plt.cm.CMRmap, interpolation='nearest')
    plt.show()
```

کد یک راه حل (sol) می گیرد که نمایانگر ستون های موقعیت های ملکه در یک صفحه شطرنج است و به صورت بصری موقعیت ها را به عنوان یک الگوی شطرنجی با مربع های رنگی نشان دهنده وزیر ها نشان می دهد.

```
npop = 100        # Number of solutions
size = 8          # Size of board and queens
ox_size = 2  # variables changed during order crossover
generation = 100  # Number of generations
p_sel = 0.95  # Probability of Selection
p_m = 0.1  #  Probability of Mutation
numberOfSwaps = 2 # Number of swaps during mutation
```

این کد پارامترهایی را برای اجرای یک الگوریتم ژنتیک برای حل مشکل N-Queens تنظیم می کند. الگوریتم ژنتیک جمعیتی از راه‌حل‌ها را ایجاد می‌کند، عملگرهای ژنتیکی مانند متقاطع و جهش را اعمال می‌کند و در تعداد مشخصی از نسل‌ها برای یافتن راه‌حل بهینه یا نزدیک به بهینه تکرار می‌شود.

```
pop = np.zeros((npop,size))
for i in range(npop):
   pop[i,:] = np.random.permutation(size)+1

fit = checkFitness(pop)
pop = np.hstack((pop, fit))
```

```
meanFit = np.zeros(generation)
```

این کد پارامترهایی را برای اجرای یک الگوریتم ژنتیک برای حل مشکل N-Queens تنظیم می کند. الگوریتم ژنتیک جمعیتی از راه‌حل‌ها را ایجاد می‌کند، عملگرهای ژنتیکی مانند متقاطع و جهش را اعمال می‌کند و در تعداد مشخصی از نسل‌ها برای یافتن راه‌حل بهینه یا نزدیک به بهینه تکرار می‌شود.

```
for gen in range(generation):

    print(f"Generation: {gen} / {generation}")

    parents = [selection(pop,p_sel),selection(pop,p_sel)]

    offsprings = order_crossover(parents[0][0:size], parents[1][0:size], ox_size)

    for child in range(len(offsprings)):
        r_m = round(random.random(),2)
        if r_m <= p_m:
            offsprings[child] = swap_mutation(offsprings[child], numberOfSwaps)

    fitOff = checkFitness(offsprings)
    offsprings = np.hstack((offsprings, fitOff))

    pop = np.vstack([pop,offsprings])
```

```
pop = pop[pop[:,size].argsort()][0:npop, :]

meanFit[gen] = (pop[:,size]).mean()
```

این کد حلقه الگوریتم ژنتیک اصلی را نشان می دهد. والدین را انتخاب می کند، فرزندان را از طریق عملیات متقاطع و جهش تولید می کند، اعتبار آنها را ارزیابی می کند، جمعیت را به روز می کند و میانگین اعتبار و ارزیابی را برای هر نسل محاسبه می کند. این فرآیند برای تعداد مشخصی از نسل ها تکرار می شود.

```
bestSol = pop[np.argmin(pop[:, size]), :]

#Plot Graphic
plt.plot(meanFit)
plt.grid()
plt.title("Evolution of Fit (Mean)")
plt.ylabel("Fit Mean")
plt.xlabel("Generation")
plt.show()


print(f"Best Solution have: { bestSol[size]} Conflict(s)")
plotCheckBoard(bestSol[0:size])
```

، این کد بهترین راه حل به دست آمده از الگوریتم ژنتیک را پیدا می کند، تکامل مقادیر میانگین تناسب اندام را ترسیم می کند، تعداد تضادهای بهترین راه حل را چاپ می کند و صفحه شطرنج را با موقعیت های ملکه بهترین راه حل تجسم می کند.

کد دوم:

```
from operator import indexOf
import random
```

این کد بهترین راه حل به دست آمده از الگوریتم ژنتیک را پیدا می کند، تکامل مقادیر میانگین ارزیابی را ترسیم می کند، تعداد تضادهای بهترین راه حل را چاپ می کند و صفحه شطرنج را با موقعیت های وزیر بهترین راه حل تجسم می کند.

```
def random_chromosome(size):
    return [random.randint(0, size - 1) for _ in range(size)]
```

تابع random_chromosome با ایجاد لیستی از اعداد صحیح تصادفی یک کروموزوم تصادفی تولید می کند. اندازه کروموزوم با پارامتر اندازه ارسال شده به تابع تعیین می شود. این کد بسته به مقدار اندازه، اجازه می دهد تا کروموزوم هایی با طول های مختلف تولید کند.

```
def fitness(chromosome, maxFitness):
    horizontal_collisions = (
        sum([chromosome.count(queen) - 1 for queen in chromosome]) / 2
```

```
)
diagonal_collisions = 0


n = len(chromosome)
left_diagonal = [0] * (2 * n - 1)
right_diagonal = [0] * (2 * n - 1)
for i in range(n):
    left_diagonal[i + chromosome[i] - 1] += 1
    right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1


diagonal_collisions = 0
for i in range(2 * n - 1):
    counter = 0
    if left_diagonal[i] > 1:
        counter += left_diagonal[i] - 1
    if right_diagonal[i] > 1:
        counter += right_diagonal[i] - 1
    diagonal_collisions += counter


# 28-(2+3)=23
return int(maxFitness - (horizontal_collisions + diagonal_collisions))
```

تابع تناسب تناسب یک کروموزوم را در مسئله N-Queens محاسبه می کند. هم برخوردهای افقی و هم مورب را در نظر می گیرد و یک مقدار تناسب را برمی گرداند که تعداد درگیری ها یا برخوردهای کروموزوم را با توجه به حداکثر مقدار تناسب نشان می دهد.

```
def crossover(x, y):
    n = len(x)
    child = [0] * n
    for i in range(n):
        c = random.randint(0, 1)
        if c < 0.5:
            child[i] = x[i]
        else:
            child[i] = y[i]
    return child
```

تابع متقاطع یک تقاطع باینری بین دو کروموزوم والد x و y انجام می دهد. با انتخاب تصادفی ژن‌های هر یک از والدین در هر موقعیت، کروموزوم فرزند جدیدی ایجاد می‌کند. کروموزوم فرزند حاصل توسط تابع برگردانده می شود.

```
def mutate(x):
    n = len(x)
    c = random.randint(0, n - 1)
    m = random.randint(0, n - 1)
    x[c] = m
    return x
```

تابع جهش با انتخاب تصادفی یک ژن و تغییر مقدار آن به یک مقدار تصادفی جدید در محدوده مقادیر معتبر برای آن ژن، جهش را روی کروموزوم انجام می دهد. سپس کروموزوم جهش یافته توسط تابع برگردانده می شود.

```
def probability(chromosome, maxFitness):
    return fitness(chromosome, maxFitness) / maxFitness
```

تابع احتمال، احتمال انتخاب یک کروموزوم را بر اساس مقدار تناسب آن نسبت به حداکثر مقدار تناسب محاسبه می کند. احتمال به دست آمده یک مقدار نرمال شده بین ۰ و ۱ است که نشان دهنده احتمال انتخاب کروموزوم در طول فرآیند انتخاب است.

```
def random_pick(population, probabilities):
    populationWithProbabilty = zip(population, probabilities)
    total = sum(w for c, w in populationWithProbabilty)
    r = random.uniform(0, total)
    upto = 0
    for c, w in zip(population, probabilities):
        if upto + w >= r:
            return c
        upto += w
    assert False, "Shouldn't get here"
```

تابع random_pick یک انتخاب تصادفی از یک جمعیت را بر اساس احتمالات مرتبط انجام می دهد. در میان جمعیت تکرار می شود و احتمال تجمعی را تا زمانی که به یک عدد تصادفی تولید شده برسد یا از آن فراتر رود بررسی می کند. فرد مربوط به آن احتمال تجمعی انتخاب شده و برگردانده می شود.

```
def genetic_queen(population, maxFitness):
    mutation_probability = 0.1
    new_population = []
    sorted_population = []
    probabilities = []
    for n in population:
        f = fitness(n, maxFitness)
        probabilities.append(f / maxFitness)
        sorted_population.append([f, n])

    sorted_population.sort(reverse=True)
```

تابع genetic_queen مقادیر تناسب و احتمالات را برای هر کروموزوم در جمعیت محاسبه می کند. همچنین جمعیت را بر اساس مقادیر تناسب اندام به ترتیب نزولی مرتب می کند. این مراحل معمولاً به عنوان آماده سازی اولیه قبل از انجام عملیات ژنتیکی مانند انتخاب، متقاطع و جهش روی جمعیت انجام می شود.

```
new_population.append(sorted_population[0][1])  # the best gen
    new_population.append(sorted_population[-1][1])  # the worst gen
```

```
    for i in range(len(population) - 2):

        chromosome_1 = random_pick(population, probabilities)
        chromosome_2 = random_pick(population, probabilities)

        # Creating two new chromosomes from 2 chromosomes
        child = crossover(chromosome_1, chromosome_2)

        # Mutation
        if random.random() < mutation_probability:
            child = mutate(child)

        new_population.append(child)
        if fitness(child, maxFitness) == maxFitness:
            break
    return new_population
```

این بخش از تابع genetic_queen کروموزوم های والد را بر اساس احتمالات آنها انتخاب می کند، برای ایجاد کروموزوم های فرزند جدید، جهش را برای برخی از کروموزوم های فرزند اعمال می کند و آنها را به جمعیت جدید اضافه می کند. این روند تا زمانی ادامه می یابد که راه حلی با حداکثر تناسب پیدا شود یا تعداد فرزندان مورد نظر تولید شود.

```
def print_chromosome(chrom, maxFitness):
```

```
    print(
        "Chromosome = {},  Fitness = {}".format(str(chrom), fitness(chrom,
maxFitness))
    )
```

تابع print_chromosome برای نمایش نمایش ژنتیکی و ارزش تناسب یک کروموزوم استفاده می شود. این یک راه راحت برای تجسم و بازرسی کروموزوم های فردی در طول اجرای یک الگوریتم ژنتیک فراهم می کند.

```
def print_board(chrom):
    board = []

    for x in range(nq):
        board.append(["x"] * nq)

    for i in range(nq):
        board[chrom[i]][i] = "Q"

    def print_board(board):
        for row in board:
            print(" ".join(row))

    print()
    print_board(board)
```

```
if __name__ == "__main__":
    POPULATION_SIZE = 500


    while True:
        # say N = 8
        nq = int(input("Please enter your desired number of queens (0 for exit): "))
        if nq == 0:
            break


        maxFitness = (nq * (nq - 1)) / 2  # 8*7/2 = 28
        population = [random_chromosome(nq) for _ in
range(POPULATION_SIZE)]


        generation = 1
        while (
            not maxFitness in [fitness(chrom, maxFitness) for chrom in population]
            and generation < 200
        ):


            population = genetic_queen(population, maxFitness)
            if generation % 10 == 0:
                print("=== Generation {} ===".format(generation))
                print(
                    "Maximum Fitness = {}".format(
                        max([fitness(n, maxFitness) for n in population])
```

```
            )
        )
        generation += 1

    fitnessOfChromosomes = [fitness(chrom, maxFitness) for chrom in
population]

    bestChromosomes = population[
        indexOf(fitnessOfChromosomes, max(fitnessOfChromosomes))
    ]

    if maxFitness in fitnessOfChromosomes:
        print("\nSolved in Generation {}!".format(generation - 1))

        print_chromosome(bestChromosomes, maxFitness)

        print_board(bestChromosomes)

    else:
        print(
            "\nUnfortunately, we could't find the answer until generation {}. The best
answer that the algorithm found was:".format(
                generation - 1
            )
        )
        print_board(bestChromosomes)
```

این برنامه به کاربر اجازه می دهد تا تعداد وزیرهای مورد نظر را وارد کند و با استفاده از الگوریتم ژنتیک سعی در حل مسئله N-Queens دارد. این به طور مکرر جمعیتی از کروموزوم ها را تکامل می دهد، عملیات انتخاب، متقاطع و جهش را انجام می دهد و مقادیر تناسب را برای یافتن راه حل ردیابی می کند. این برنامه به‌روزرسانی‌هایی را در مورد پیشرفت ارائه می‌کند و بهترین راه‌حل پیدا شده را همراه با نمایش بصری صفحه شطرنج نمایش می‌دهد.

# فصل ششم

# جمع‌بندی و نتیجه‌گیری

# جمع‌بندی و نتیجه‌گیری

در پایان، هدف این پروژه حل مشکل n-Queens با استفاده از زبان برنامه نویسی پایتون بود. این پروژه یک رویکرد الگوریتم ژنتیک را پیاده‌سازی کرد که شامل تولید یک جمعیت اولیه از راه‌حل‌های تصادفی به‌عنوان کروموزوم، ارزیابی تناسب آنها با استفاده از یک تابع مناسب و استفاده از عملگرهای ژنتیکی مانند انتخاب، متقاطع و جهش برای بهبود مکرر راه‌حل‌ها بود. این الگوریتم برای یافتن پیکربندی وزیرها در n x n صفحه شطرنج طراحی شده است که در آن هیچ دو ملکه نمی‌توانند به یکدیگر حمله کنند. (گه ما از ۸ و ۱۶ استفاده می‌کنیم.)

این برنامه یک رابط کاربر پسند برای وارد کردن تعداد وزیر ها ارائه کرد و بهترین راه حل یافت شده را به همراه نمایش تصویری صفحه شطرنج نمایش داد. با استفاده از قدرت الگوریتم‌های ژنتیک و انعطاف‌پذیری پایتون، این پروژه با موفقیت به مشکل n-Queens پرداخت و کاربرد تکنیک‌های محاسباتی تکاملی را در سناریوهای حل مسئله به نمایش گذاشت.

# منابع و مراجع

[1] https://github.com/paulojunqueira/N-Queen-Problem-Evolutionary-Algorithm/blob/master

[2] https://github.com/mahdihassanzade/N-Queen-Problem-using-Genetic-Algorithm/blob/main

[3]

# Abstract

In this project, we aim to write a program using the Python programming language that solves the n-Queens problem using a gradual evolution algorithm.

.


**Amirkabir University of Technology**
**(Tehran Polytechnic)**

**Department of Mathematics and Computer science**

**AI priject**

# Report of the 6th project

**By**
**Hasti Barghararіyan**

**Supervisor**
**Dr. Mahdi Ghatee**

**Advisor**
**Behnam Yousefimehr**

**May 2023**