گزارش پروژه ژنتیک

آوا میرمحمدمهدی 810199501

تعریف ژن و کروموزوم:

در این پروژه هر ژن را یک operator را operand در نظر می‌گیریم و برای ساختن کروموزوم به تعداد طول equation از این ژن‌ها را در کنار هم قرار می‌دهیم (به این نکته توجه شود که در اندیس‌های زوج یک operand و در اندیس‌های فرد یک operator قرار می‌گیرند).

تولید جمعیت اولیه:

برای تولید جمعیت اولیه به تعداد populationSize کروموزوم تولید می‌کنیم. برای تولید کروموزوم برای هر اندیس زوج یک operand تصادفی از مجموعه operandها و برای هر اندیس فرد یک operator تصادفی از مجموعه operatorها انتخاب می‌کنیم.

```
def makeFirstPopulation(self):
    population = []
    for i in range(populationSize):
        chromosome = []
        for j in range(self.equationLength):
            if j % 2 == 0:
                chromosome.append(random.choice(self.operands))
            else:
                chromosome.append(random.choice(self.operators))
        population.append(chromosome)
    return population
```

پیاده‌سازی و مشخص کردن fitness function:

به ازای هر کروموزوم موجود در جمعیت اولیه، حاصل آن عبارت ریاضیاتی را محاسبه کرده و قدرمطلق اختلاف این حاصل با goalNumber را به عنوان fitness آن کروموزوم در نظر می‌گیریم؛ با تعریف ارائه شده هرچه مقدار fitness یک کروموزوم کمتر باشد، آن کروموزوم بهتر است.

```
def calcFitness(self, chromosome):
        chromosomeStr = ''.join(str(chrom) for chrom in chromosome)
        chromRes = eval(chromosomeStr)
        fitness = abs(self.goalNumber - chromRes)
        return fitness
```

تشکیل matingPool:

ابتدا کروموزوم‌ها را بر اساس fitnessشان به ترتیب نزولی مرتب می‌کنیم (اولین عضو دارای بیشترین مقدار fitness است و درواقع بدترین کروموزوم از نظر نزدیک بودن به جواب)؛ حال در واقع یک رتبه به هر کروموزوم نسبت می‌دهیم، اولین عضو آراهی مرتب شده دارای رتبه 1 است و به همین ترتیب رتبه‌ی هر کروموزوم نسبت به کروموزوم قبلی خود یک واحد اضافه می‌شود. هر کروموزوم را به تعداد رتبه‌اش در mating pool کپی می‌کنیم؛ با اینکار نزدیک ترین کروموزوم به جواب اصلی به تعداد بیشتری نسبت به بقیه کروموزوم‌ها تکرار می‌شود. حال mating pool بدست آمده را شافل می‌کنیم تا عضوهایش به صورت رندوم در کنار یکدیگر قرار گیرند و در انتها به تعداد populationSize از آن برمیداریم.

```
def createMatingPool(self, fitnesses):
        matingPool = []
        for i in range(populationSize):
            fitness, chromosome = fitnesses[i]
            for j in range(i+1):
                matingPool.append(chromosome)
        random.shuffle(matingPool)
        return matingPool[0:populationSize]
```

تشکیل carriedChromosomes:

برای اینکار تعدادی از کروموزوم‌هایی که fitness بهتری از بقیه دارند را به طور مستقیم وارد پاسخی که از مرحله فعلی به مرحله بعد می‌فرستیم می‌کنیم؛ مقدار این تعداد بستگی به carryPercentage دارد.

```
carriedChromosomes = []
            for i in range(0, int(populationSize*carryPercentage)):
                fit , chromosome = fitnesses[populationSize - 1 -i]
                carriedChromosomes.append(chromosome)
```

تشکیل crossoverPool:

در این مرحله باید از ترکیب کردن ژن‌های دو کروموزوم والد، دو کروموزوم فرزند بسازیم؛ برای اینکار دو عضو متوالی از matingPool به عنوان دو کروموزوم والد برمی‌داریم و همچنین یک عدد رندوم بین 0 تا 1 نیز تولید می‌کنیم، درصورتی که عدد رندوم تولید شده از crossoverProbability بیشتر باشد، دو فرزند ما عینا همان دو والد خواهند بود و آنها را به crossoverPool منتقل می‌کنیم و در غیر این صورت یک عدد رندوم بین اعدادی که در بازه‌ی طول equation وجود دارد انتخاب می‌کنیم و از ابتدای کروموزوم

والد اول تا آن ژن و از ابتدای آن ژن تا انتها در کروموزوم والد دوم را به کروموزوم فرزنداول می‌دهیم و برای فرزند دوم نیز برعکس این عمل می‌کنیم.

```
def createCrossoverPool(self, matingPool):
        crossoverPool = []
        i = 0
        while i < len(matingPool)-1:
            firstParent, secondParent = matingPool[i], matingPool[i+1]
            if (random.random() > crossoverProbability):
                crossoverPool.append(firstParent)
                crossoverPool.append(secondParent)
            else:
                crossoverPoint = random.randrange(1, self.equationLength - 1)
                firstChild = firstParent[:crossoverPoint] + secondParent[crossoverPoint:]
                secondChild = secondParent[:crossoverPoint] + firstParent[crossoverPoint:]
                crossoverPool.append(firstChild)
                crossoverPool.append(secondChild)
            i += 2
        return crossoverPool[0:populationSize]
```

در انتها به تعداد carriedChromosomes به population ای که به مرحله بعدی می‌رود اضافه می‌کنیم و بقیه‌ی آرایه population را از crossoverPoolپر می‌کنیم؛ در میان اینکار به ازای هر عضو crossovePool یک عدد رندوم تولید می‌کنیم و درصورتی که این عدد رندوم از mutationProbability کوچکتر بود، آن کروموزوم را به طور مستقیم از crossoverPool به population اضافه می‌کنیم و در غیر این صورت دو ژن رندوم (در واقع دو اندیس از ژن‌ها) از کروموزوم را انتخاب می‌کنیم (یکی متغلق به اندیس زوج و دیگری متعلق به اندیس فرد) و یک عضو از operandها و یک عضو از operatorها به طور رندوم انتخاب می‌کنیم و دو ژن موجود در آن اندیس‌ها را با آنها جابجا می‌کنیم. یکی از دلایل این امر این است که اگر در جمعیت اولیه operator یا operand ای وجود نداشت، در جمعیت جدید تولید شده احتمال وجود آن باشد.

```
def mutate(self, chromosome):
      if random.random() > mutateProbability:
          return chromosome
      else:
          evenIndex , oddIndex = random.randrange(0, self.equationLength -1, 2), random.randrange(1, self.equationLength -1, 2)
          chromosome[evenIndex] = random.choice(self.operands)
          chromosome[oddIndex] = random.choice(self.operators)
      #TODO mutate the input chromosome
      return chromosome
```

سوالات:

1) جمعیت اولیه ی بسیار کم یا بسیار زیاد چه مشکلاتی را به وجود می آورند؟

اگر جمعیت اولیه بسیار کم باشد، کروموزوم‌های اولیه که به صورت رندوم ساخته می‌شوند ممکن است برخی از operand ها یا operator ها را نداشته باشند و درواقع تنوع کروموزوم‌های اولیه کم باشد؛ در مقایسه، جمعیت خیلی زیاد باعث می‌شود زمان انجام عملیات محاسباتی و پیمایش بیش از حد حلقه‌ها مانند بدست آوردن fitness و crossoverPool بسیار زیاد شود.

2) اگر تعداد جمعیت در هر دوره افزایش یابد، چه تاثیری روی دقت و سرعت الگوریتم می گذارد؟

در اینصورت دقت با احتمال خوبی افزایش می‌یابد ولی سرعت الگوریتم کاهش پیدا می‌کند؛ همچنین اگر مراحل تولید نسل زیاد باشد و در هر مرحله تعداد جمعیت بیشتر شود، ممکن است این عدد بسیار زیاد شود و حافظه کافی برای ذخیره آن نداشته باشیم.

3) تاثیر هر یک از عملیات crossover و mutation را بیان و مقایسه کنید. ایا می‌توان فقط یکی از آنها را استفاده کرد؟ چرا؟

عملیات crossover از هر دو کروموزوم والد، دو کروموزوم فرزند تولید می‌کند که هر فرزند ویژگی‌هایی ترکیبی از دو والد خود را دارا است و درواقع با استفاده از دو راه حل خوب، دو راه حل جدید ساخته می‌شود که ممکن است از دو راه حل قبلی بهتر باشند و اینکار را با احتمال حدود 70 درصد به بالا انجام می‌دهیم. در عملیات mutation که به احتمال کم (یعنی حدود 0.1 درصد یا کمتر) انجام می‌شود، ما برخی از ژن‌های یک کروموزوم را تعویض یا جابجا می‌کنیم، با انجام اینکار احتمال خروج از مینیمم محلی یا ماکسیمم محلی بیشتر می‌شود.

4) به نظر شما چه راهکارهایی برای سریعتر به جواب رسیدن در این مسئله ی خاص وجود دارد؟

برای سریعتر به جواب رسیدن می‌توان از همان مرحله ابتدایی یعنی تولید جمعیت اولیه، عملگرها و عملوندها را در جایگاه صحیح قرار دهیم (یعنی در اندیس‌های زوج عملوندها و در اندیس‌های فرد عملگرها قرار گیرند). همچنین در عملیات crossover نیز دقت شود که ژن‌هایی که از هر والد در فرزند قرار می‌گیرند، در جایگاه درستی از نظر عملگر و عملوند باشند و در mutation نیز به جای ژن‌های تعویضی باید متناسب با جایگاه، عملوند یا عملگر قرار گیرد.

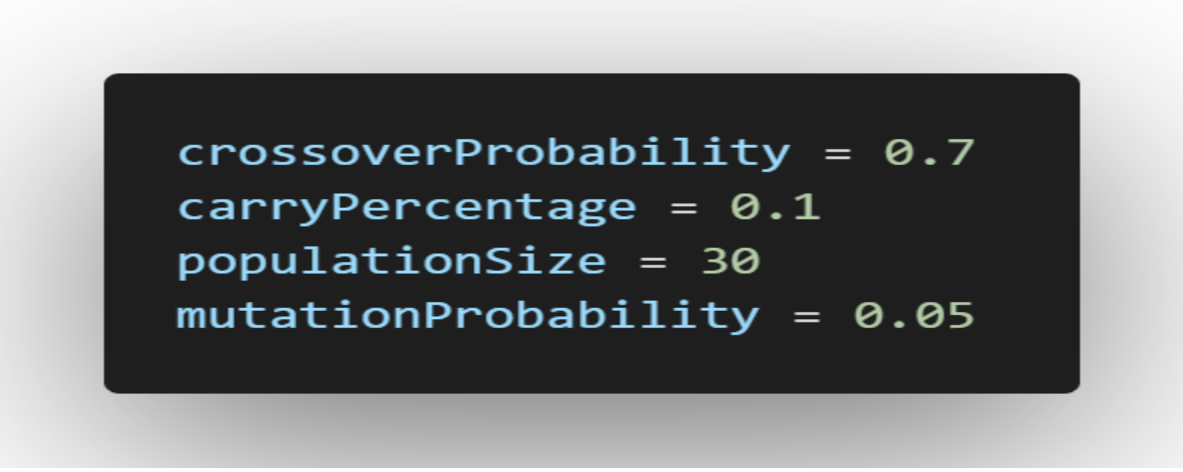
5) با وجود استفاده از این روش ها، باز هم ممکن است که کروموزوم ها پس از چند مرحله دیگر تغییر نکنند. دلیل این اتفاق و مشکلاتی که به وجود می آورد را شرح دهید. برای حل آن چه پیشنهادی می دهید؟ (راه حلهای خود را امتحان کنید و بهترین آن ها را روی پروژه خود پیاده سازی کنید)

اگر عملیات mutation و عملیات‌هایی مشابه آن انجام نشود، احتمال اینکه کروموزوم‌ها بعد از چند مرحله دیگر تغییر نکنند بیشتر می‌شود، در واقع در این شرایط در یک اکسترمم محلی گیر می‌کند و خارج شدن از آن مشکل می‌شود؛ در اینجا برای حل این مشکل از mutation استفاده می‌کنیم.

6) چه راه حلی برای تمام شدن برنامه در صورتی که مسئله جواب نداشته باشد پیشنهاد می دهید؟

می‌توان برای حل این مشکل تعداد نسل‌های محاسبه شده از ابتدای جمعیت اولیه را محاسبه کنیم و درصورتی که این عدد از مقداری بیشتر شد، عملیات را متوقف کنیم و اعلام کنیم که پاسخی برای مساله وجود ندارد؛ بدیهی است تعداد نسل‌هایی که مورد بررسی قرار می‌گیرند باید متناسب با تعداد جمعیت و در این مساله طول معادله داده شده باشد.

در این پروژه مقادیر احتمالات داده شده با توجه به توضیحات به صورت زیر انتخاب شده است.



زمان اجرای الگوریتم با توجه رندوم بودن بسیاری از بخش‌ها متفاوت است ولی یک نمونه از زمان اجرا در زیر قرار داده شده است.