Assignment #1

Genetic Algorithm을 통한 N-Queens Problem해결

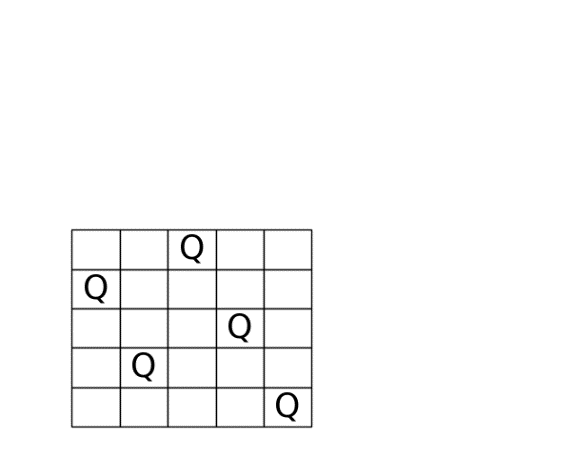
부산대학교 전기컴퓨터공학부 정보컴퓨터공학전공

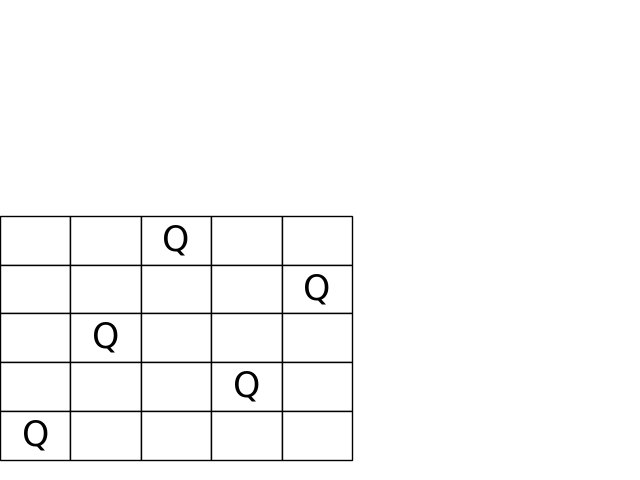
201524582 정희석

1. n-Queens Problem?

- NxN 체스판이 있을 때 n개의 Queen이 서로 공격을 못하게 놓는 방법을 찾는 문제이다.

Ex) 5-Queens Problem:

5x5 체스판에서 5개의 Queen이 서로 간섭이 안되도록 위치시키는 법을 찾는 문제이다.



2. Genetic Algorithm?

- Genetic Algorithm은 유전자의 진화에서 착안한 알고리즘으로 Chromosome이라는 유전배열을 디자인하여(Chromosome Design) 데이터를 넣어서 생성(Initialization), 원하는 값을 가지는 지 확인단계(Fitness Evaluation), 부모 유전자 선택(Selection), 유전자 교차(Crossover), 유전자 돌연변이 발생(Mutation)를 몇 세대에 걸쳐서 진행시켜서 원하는 데이터를 얻는 알고리즘이다.

3. 각 단계별 진행

1) Chromosome Design

Chromosome은 5개의 element를 가지고 있는 Array.

각 element는 1~5의 숫자로 구성

ex)Chromosome = [1,2,3,4,5]

=>

   퀸ㅁㅁㅁㅁ

   ㅁ퀸ㅁㅁㅁ

   ㅁㅁ퀸ㅁㅁ

   ㅁㅁㅁ퀸ㅁ

   ㅁㅁㅁㅁ퀸

chromosome을 총 10개를 만들어서 초기화. 각각의 chromosome(list)를 새로운 List에 넣어서 2차원 list를 표현, 이를 chromosomeList로 명명하여 사용한다.

2) Initialization

Random 모듈에서 shuffle을 사용하여 초기상태에서 5개의 숫자가 모두 다른 상태로 일직선상에 위치하는 것을 피하도록 초기화 한다.

3) Fitness Evaluation(가장 중요한 부분)

이 단계에서 확인해야 하는 부분은 같은 행에 퀸이 위치하고 있는가, 대각선 상에 위치하고 있는가이다.

5-Queens Problem을 예시로 들어보면 List에서 첫 번째 chromosome부터 검사 진행

현재의 chromosome가 n개 중에서 x번째이며 이 chromosome내의 현재index를 i라고 할 때

i = 0 -> 4(=n-1)이고 먼저chromosomeList[x][i]와 chromosomeList[x][j](j=i+a or j=i-a)가 같은지를 확인하고 같은 경우 check값을 추가하여 fitness Evaluation의 weight를 부여한다.

다음은 현재 위치에서 대각선 방향에 위치하고 있는지를 확인한다.

chromosomeList[x][i] chromosomeList[x][i-j]-j or +j 또는

chromosomeList[x][i+j]-j or +j가 chromosomeList[x][i]와 같은지를 확인하고 같은 경우 check값을 추가하여 fitness Evaluation의 weight를 부여한다.

숫자로 예시를 들어보면 1,4,3,5,2에서는 (1.1), (2,4), (3,3), (4,5), (5,2)에 각각 퀸이 위치하는데 (1,1)과 (3,3)에서 대각선 상에 있는 것을 확인할 수 있다. 이를 수식으로 옮겨보면 현재의 chromosome이 x번째 chromosome이라고 할 때 (1,1)은 chromosomeList[x][0], (3,3)은 chromosomeList[x][2]이다. 이 둘이 대각선 관계에 있음은 둘의 index차이인 2를 첫 번째 element인 1에 더했을 때 (chromosomeList[x][0]+2==chromosomeList[x][2]) 또는 세 번째 element인 3에서 뺐을 때 (chromosomeList[x][0]==chromosomeList[x][2] - 2) 값이 일치하기 때문에 충돌이 일어난다고 표현할 수 있다. 이를 표현한 것이 위의 표현이다. 실제 구현은 아래와 비슷한 구조로 구현한다.

for x in chromosomeList

for i in range(0,5)

for j in range(1,4)

x[i]를 기준으로 i-j > 0 or i+j < 5 이면 check

i-j > 0 && i+j < 5

그리고 selection을 진행하기 위하여 각 chromosome의 check값을 checkList를 만들어서 관리한다.

위의 과정에서 check가 0인 경우에는 우리가 찾는 정답이므로 이를 출력하고 기존의 chromosome들을 삭제하고 chromosomeList에 정답 chromosome을 복사하며 이 함수는 check값(충돌 weight)을 출력하게 한다.

4) Selection

3단계에서 나온 check들 중 가장 낮은 check를 selection개수 만큼 찾아서 해당 index의 chromosome을 선택, 이를 기존의 chromosome들을 삭제하고 추가하여 남기는 과정을 진행한다. 이 때 Selection의 개수는 n\_of\_select를 인자로 전달받는다.

5) Crossover

4단계에서 나온 chromosome들을 기반으로 하며 해당 chromosome의 데이터의 개수의 절반을 기준으로 앞의 데이터들과 뒤의 데이터들을 crossover하여 리스트에 추가, 앞의 데이터들은 parent chromosome들의 낮은 index부터, 뒤의 데이터들은 parent chromosome들의 높은 index부터 진행하여 crossover한다.

Ex) [1.3.4.5.2] & [2,5,3,1,4]가 selected chromosome이라 하면 parent chromosome들은 남겨놓고 추가로 [1,3,3,1,4]&[2,5,4,5,2]를 만들어 낸다.

6) Mutation

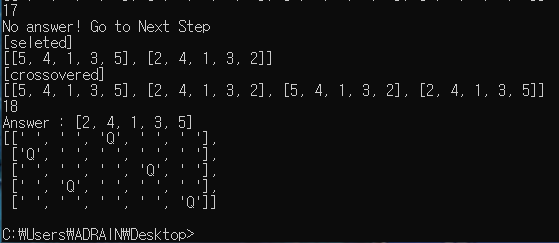
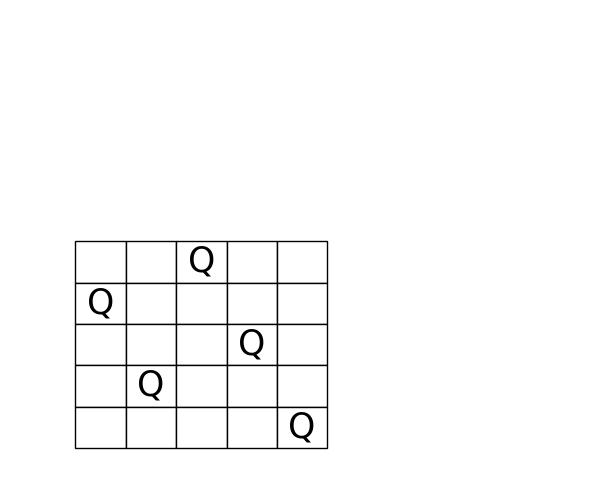
4,5단계에서 선택된 chromosome들을 기반으로 각각의 chromosome마다 3개의 random seed를 사용, 첫 번째 seed는 mutation을 할 element의 개수를, 두 번째 seed는 첫 번째 seed를 반복하는 동안 mutation을 시행할 chromosome의 element의 위치를, 세 번째 seed는 어떤 수로 바꿀지를 결정하여 이를 chromosome 리스트에 추가하여 원래의 chromosome의 개수가 될 때까지 진행한다.

7) Go to 3) Fitness Evaluation

3단계인 fitness evaluation부터 mutation까지를 답이 나올 때까지 세대가 지나면서 계속 진행한다.

4. 결과

결과는 할 때마다 다르게 나온다. shuffle하여 입력하는 initialization단계에서 답이 나오는 경우도 있고 10세대 이상 넘어가서 답이 나오는 경우도 있다. 아래는 나오는 결과 중 일부를 첨부한다.



해당 결과는 matplot module을 사용하여 체스판을 표현한 그림과 18세대 만에 해당 답이 나온 결과를 출력한 것이다. 17세대에서 보면 원하는 답이 Fitness Evaluation에서 나오지 않아서 Selection으로 진행, Selection에서 고른 2개의 가장 정답에 근접한 두 chromosome을 출력, Crossover으로 진행하여 그 결과인 4개의 chromosome를 출력한다. 위의 경우에는 crossover한 결과에서 정답이 나온 경우이다. 그리고 n=5일 때 chromosome을 10개로 하면 initialization과정에서 답이 나올 확률이 높다. 이유는 initialization 과정에서 완전한 random이 아닌 shuffle을 사용하였기 때문이다. 이는 chromosome을 6개 정도로 낮추면 여러 세대를 거쳐갈 확률이 높아진다.

5. 개선할 점 및 아쉬운 점

- Crossover에서 다양성 추가!

현재는 element의 개수에서 1/2를 해서 앞과 뒤를 섞는 방식이며 1-4,2-3 처럼 연결하는 방식으로 다양성이 부족 => 답을 Mutation에서 나오게 하는 방식으로 구현되어 있다. 이는 generation의 증가를 초래하며 개선해야할 과제.

- Mutation의 완전 무작위 발생

현재는 3번의 random seed를 사용하여 첫 번째 seed에서는 mutation의 개수를, 두 번째 seed부터는 첫 번째 seed의 수만큼 반복하면서 mutation이 될 위치, 세 번째 seed는 mutation으로 대체될 수를 정해서 확실한 확률을 알 수가 없다. => 이 역시 Generation의 증가를 초래하며 개선해야할 과제이다.

- crossover와 mutation의 간섭 문제

현재 코드는 selection과 crossover진행 후에도 chromosome의 개수가 기존에 설정한 chromosome의 개수보다 작은 경우에는 정상적으로 mutation이 진행되는데 만약 chromosome의 개수를 4개로 설정하고 selection을 2개로 설정한다면 mutation이 진행이 안되도록 되어있는 부분을 어떻게 하면 좋을지가 개선 과제이다.

- 선택권을 콘솔에서 받지 않고 값을 저장하여 사용

현재 코드는 n-Queens 문제의 n값을 n\_of\_element에 저장, chromosome의 개수를 n\_of\_chromosome에, selection할 개수를 n\_of\_selection에 저장하여 소스코드 단계에서 지정하도록 하였는데 이번 과제가 5-queens에 국한되어 있고 동작여부를 확인하는 것에 초점이 맞춰져 있기 때문이었다. 이를 프로그램 사용자로부터 입력 받아서 지정할 수 있도록 할 수 있으면 더 좋을 것 같다.

6. 느낀 점

이번 과제는 모델 디자인은 생각보다 어렵지 않았지만 이를 구현하는데 있어서 쉽지는 않았다. 각 과정을 function으로 나누어 작성하고 이를 확인하는 과정으로 진행하였으며 tensorflow용 colab환경에서 정리하면서 진행하였다. 생각을 실제로 구현화 하는 작업은 항상 어렵다고 생각한다. 하지만 이번 과제를 통해서 python 사용법을 remind할 수 있었고 genetic algorithm을 이해할 수 있었다. 그리고 여러 추후 해결해야할 과제들을 찾을 수 있었던 좋은 기회였다.