N-queens variant using genetic algorithm

Badral Khurelbaatar 101166852
COMP 3106 Project Report

Carleton University
Ottawa, ON, Canada

Statement of contribution: This is a solo project

1. Introduction

1.1 Background

N-queens problem is a problem of placing n queens on nxn sized board, where no queens attack each other.

In a regular chess game, a queen can attack in a straight line vertically, horizontally, or diagonally. Therefore, there can be no more than one queen in each row, column, and diagonal on the board.

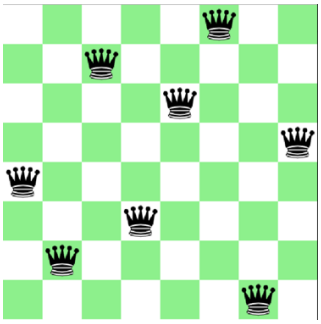

Fig.1: (This is one of the solutions for n-queens with n = 8)

An example of n-queens problem, eight queens puzzle was first published by chess composer Max Bezzel in 1848. Franz Nauck published the first solution in 1850, and he developed the problem into N-queen problem.

N-queen problem can be solved using backtracking method, but it is computationally very expensive as the N gets higher O(n!). I am using genetic algorithm to solve the n-queen algorithm faster than backtracking method.

1.2 Objective

For this project, I solved the n-queens variant using genetic algorithm where there are a random number of walls on the board from 1 to n * (n-1).

The purpose of the walls is to allow 2 queens to stay in same horizontal, diagonal, or diagonal line on the board. If there is a wall between two queens that are in the same line, the two queens won't be able to attack each other, because the wall blocks them from attacking each other.

2. Methods

The type of method used from artificial intelligence is genetic algorithm (GA).
Genetic algorithm is a heuristic search that computes the solution to a program.
It is inspired from Charles Darwin's Theory of Evolution where a genotypes with characteristic best fit to their environment are more likely to survive.

2.1 String representation

To begin with, we should represent the board with n queens using a structure.
For simplicity, this problem has one queen on each column even though I can put two or more queens on same column with walls between them.

To represent a genotype, I used an integer list of size n. Each integer in the list represents the row number of a column with the list index.

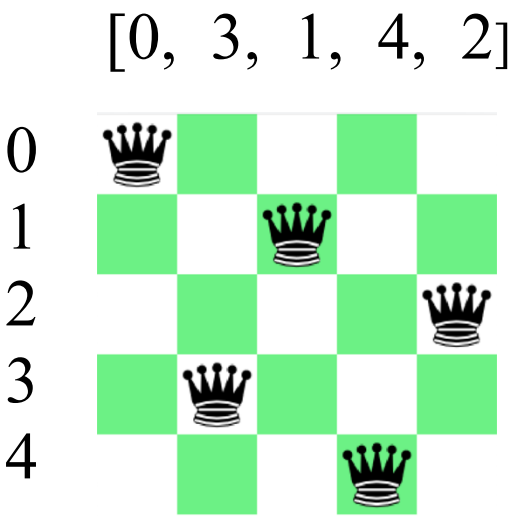

Fig.2:

In **Fig.2**, element at index 0 has value 0, which represents the queen at column 0 and row 0.
Element at index 1 has value 3 which represents the queen at column 1 and row 3.
Element at index 2 has value 1 which represents the queen at column 2 and row 1.
Element at index 3 has value 4 which represents the queen at column 3 and row 4.
Element at index 4 has value 2 which represents the queen at column 4 and row 2.

2.2 Initial population

Initially, some number of populations of the species is generated randomly and the whole population is tested for the solution and bred until the solution is found.

In the algorithm, user input defines the population size and for each genotype and fitness of the genotype is computed. If the fitness of a genotype is found to be infinite, the solution is found during the initialization of the population. Tuple of genotype fitness and the genotype itself is added to a list of population in ascending order of genotype fitness.

Example: [(1/3, [1,0,3,2]), (1/2, [2,3,0,1])]

2.3 Fitness function

In nature, ones with the highest chance of survival rate stays alive and reproduces.
To calculate the survival chance of the candidate solutions, we need to compute a fitness function which calculates the survival rate of a genotype.

Survival rate depends on the number of conflicts there are on the board. There are two types of conflicts.

1. There are two queens attacking each other.
2. A queen and a wall occupy the same square on the board.

We want the genotypes with the least number of conflicts to survive and reproduce, so we calculate fitness functions as:

$$f(x) = \frac{1}{\#\ conflict}$$

To find the number of conflicts, I used nested for loop to iterate through each pair of columns in the genotype's list, and increments the number of conflicts when

1. Two queens on same column attack each other and there is no wall between them.
2. Two queens on same diagonal attack each other and there is no wall between them.

Since I put one queen on one column, I did not have to compute if there are two or more queens on same row, because our data structure to store the genotype does not support multiple queens on the same row.

2.4 Genetic operator

In nature, the offspring of two animals/plants copies the DNA of its parents. This is called genetic operator for the algorithm.

For a parent x and y, the offspring takes the first i genes of x parent and last n-i genes of y parent. i is chosen randomly from 1 to n – 1.

$$g(x, y) = x_1 x_2 \dots x_i y_{i-1} \dots y_n \qquad 1 \leq i \leq n$$

Example:
n = 5
x = [1,0,3,4,2]
y = [4,1,3,0,2]
i = 3

g(x, y) = [1,0,3,0,2]

2.5 Mutation

In some offspring, the DNA is slightly changed due to some random mistake when copying the parents DNA.

In the algorithm, after the genetic operation, gene of the offspring is mutated with 10% chance.

Random gene is chosen randomly and set to random integer between 0 and n-1.

$$m(x) = x_1 x_2 \dots x_{j-1}\, z\, x_{j+1} \dots x_n \qquad 1 \le j \le n \qquad 0 \le z \le n-1$$

Mutation can be both bad and good to the species. For the beneficial way, as the environment changes constantly, some species need to adapt fast into the new environment to not go extinct.

2.6 Selection

In nature, individuals with the highest chance of survival stays alive and reproduces. Other individuals with low chance of survival dies, and so their genes.

In genetic algorithm, we select the ones with the highest chance of survival to breed with each other and add their offspring to the population, while we eliminate the ones with the lowest chance of survival from the population.

In my algorithm, I randomly chose two different genotypes with fitness value higher than the half of the population and produced one offspring from them and added the offspring into the population array. Then I eliminated a genotype with the lowest fitness value from the population. This step is executed n/2 types so the half of the population with lowest fitness value are eliminated and replaced with offspring off genotypes with higher fitness value.

2.7 Dataset

For the dataset for this problem, I am using an array of randomly generated genes. A user enters the number of population, and the algorithm randomly generates that number of candidate solutions.

User also inputs the number of walls they want to place on the board and the number of walls can be in range from 1 to n*(n-1) inclusive. The reason why upper limit is n*(n-1) is that there

are n*n squares and n queens on the board. Therefore, the maximum number of walls can be the difference between the total number of squares and the total number of queens

$$n^2 - n = n\,(n-1)$$

The walls' positions are randomly generated by selecting unique random numbers from 0 to n^2-1 as we can number the squares on the board starting with 0 from the top left corner.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Fig.3: (This is an example of numbering the squares on an 8x8 board)

3. Results

A user inputs N (number of queens), population size, and number of walls.
I used python's pygame module to show the first solution it finds:

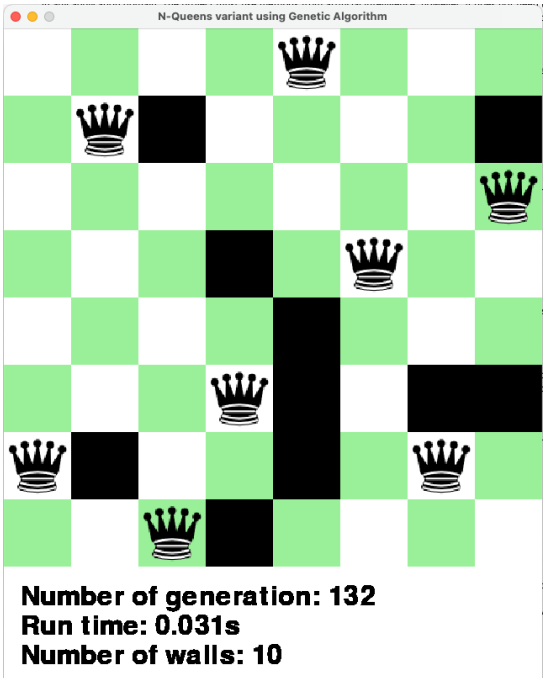

Fig.4: N=8, number of populations = 40, number of walls = 10

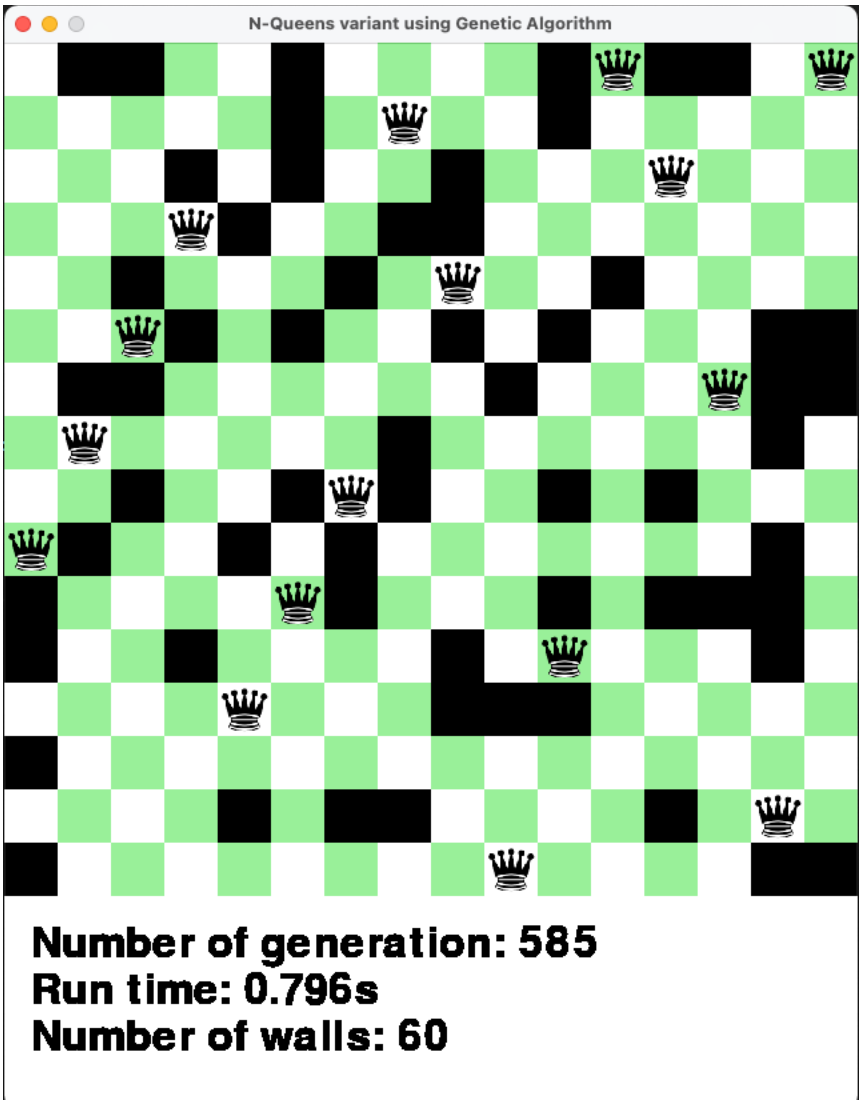

Fig.5: N = 16, wall count = 60

(Number of generations is the number of iterations it took to find the solution)
(Black squares are walls)

Fig.4 shows one of the solutions to N-queens variant problem where N is 8, number of populations is 40, and number of walls is 10.

There are no queens that attack each other, and walls between queens on same column, row, diagonal prevents two queens from attacking each other.

My algorithm also finds the time taken for finding the solution and number of generations it took.

To see the average time it took to find the solution to different number of walls, I calculated the average time for 100 runs.

The first value is average time
The second value is average iteration/generation

N	Wall count					
	0	3	6	9	12	15
6	0.0307s 983.46	0.0089s 195.72	0.0063s 106.46	0.0089s 180.48	0.0195s 479.71	-

N	Wall count							
	0	4	8	12	16	20	24	28
8	0.0258s 423.17	0.0085s 79.32	0.0071s 61.01	0.0062s 51.03	0.0075s 56.68	0.0089s 73.07	0.0094s 70.96	0.0108s 83.1

N	Wall count					
	0	10	20	30	40	50
10	0.1953s 1922.12	0.0307s 125.28	0.0301s 106.57	0.034s 103.74	0.0389s 115.47	0.0493s 146.17

N	Wall count							
	0	12	24	36	48	60	72	84
12	0.3032s 1745.42	0.0701 182.27	0.0663s 138.72	0.0652s 122.0	0.0722s 117.36	0.0895s 139.86	0.1031s 163.73	0.1288s 203.94

N	Wall count							
	0	14	28	42	56	70	84	98
14	0.5311s 1991.41	0.1058s 216.96	0.1008s 168.31	0.1029s 147.75	0.1132s 147.81	0.126s 143.62	0.1466s 165.2	0.165s 180.09

N	Wall count							
	0	16	32	48	64	80	96	112
16	0.6247s 1800.38	0.1969s 279.44	0.1871s 209.23	0.1841s 174.9	0.1935s 170.06	0.221s 178.14	0.2381s 178.27	0.2566s 182.68

From the results, having walls on the chessboard finds the solution faster than having no walls on the board. But as the wall count increases, the time taken to find the solution becomes longer.

Using genetic algorithm is way faster than using other kinds of algorithms including backtracking algorithm.

Concluding the from results above in the table, the fastest time was when the number of walls is two times the number of queens on the board.

4. Discussion

One thing I realized was that the average number of generations it took to find the solution was very high. Using better genetic operator and mutation might improve the algorithm's runtime.

If I had more time, I would implement a better selection. When selecting pairs from the population for reproduction, I would choose the genotypes probabilistically based on the fitness value instead of randomly selecting from the 2nd half of the population with higher fitness value.

Also, the reason it took long to find solution with high wall count is because when generating a random genotype, it did not exclude the walls from the possible queen placement, so it gave the genotype high fitness value for each queen placement on a wall. I would implement that when generating a genotype, it would choose only from squares with no wall.

5. References

1. https://arxiv.org/pdf/1802.02006.pdf
2. https://en.wikipedia.org/wiki/Genetic_algorithm
3. https://towardsdatascience.com/genetic-algorithm-vs-backtracking-n-queen-problem-cdf38e15d73f
4. https://en.wikipedia.org/wiki/Eight_queens_puzzle