A PROJECT REPORT ON

OPTIMIZATION N QUEEN PROBLEM USING GENETIC ALGORITHM

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

IN THE PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)

SUBMITTED BY

VISHAL TANAWADE	**18CS055**
SARTHAK THORAT	**18CS056**
HITESH SASWADKAR	**18CS048**
SURAJ SHIRUDE	**18CS051**
PRATIK KARHEKAR	**18CS027**

DEPARTMENT OF COMPUTER ENGINEERING

AISSMS COLLEGE OF ENGINEERING

PUNE 411001

SAVITRIBAI PHULE PUNE UNIVERSITY
2021 -2022

CERTIFICATE

This is to certify that the project report entitles

"**OPTIMIZATION N QUEEN PROBLEM USING GENETIC ALGORITHM**"

Submitted by

VISHAL TANAWADE	**18CS055**
SARTHAK THORAT	**18CS056**
HITESH SASWADKAR	**18CS048**
SURAJ SHIRUDE	**18CS051**
PRATIK KARHEKAR	**18CS027**

is a bonafide student of this institute and the work has been carried out by him/her under the supervision of **Prof. Dr. D. M. Ujalambkar** and it is approved for the partial fulfillment of the requirement of Savitribai Phule Pune University, for the award of the degree of **Bachelor of Engineering** (Computer Engineering).

Prof. Dr. D. M. Ujalambkar
Guide
Department of Computer Engineering

Dr. D. P. Gaikwad
Head,
Department of Computer Engineering

Place : Pune

Date :

ACKNOWLEDGEMENT

We take this opportunity to thank Head Of The Department for giving us an opportunity to work on a mini-project that helped hone our technical skills to a great extent. We are immensely grateful to **Prof. Dr. D. M. Ujalambkar** for their guidance, support and suggestions that helped us in making this project a success.

It has been an honor to be guided by **Prof. Dr. D. M. Ujalambkar** for her terrific mentorship, and we are very thankful for all of her constructive criticism that greatly helped us in our journey towards the successful completion of the project.

We would also like to acknowledge the efforts and appreciate the departmental faculty for their efforts.

INDEX

Contents:

Abstract

N-Queen is a well-known NP-Hard problem. In N-Queen problem, the challenge is to place n queens on n*n chess board such that no two queens can attack each other. The problem can-not be solved using traditional algorithms. Genetic algorithm is a well-known optimization technique. Genetic Algorithm is used to solve many computational problems of science and engineering. GA applied its genetic operators to solve a problem. The performance of genetic algorithm depends upon its genetic operators and parameter values set for population size, cross over rate and mutation rate the problem has to be represented in genetic form to solve it using genetic algorithm. The main objective of this project is to solve n queens problem using Genetic Algorithm.

Keywords- *Genetic algorithm, N-Queens Problem, NP-Hard Problem, Optimization*

1. INTRODUCTION

The concept of GA is taken from natural science. The concept is applied on some computational problems to solve. Genetic Algorithm have its operations such as encoding, initial population generation, selection, cross over, and mutation. The selection, cross over and mutation are carried out repeatedly until a satisfactory solution is not found. GA never guarantees to produce the best possible solution but it generates many solutions with different fitness values. The fitness of a solution determines its quality. Terminating criteria is set to stop the iterations of the genetic algorithm. The terminating criteria may be number of iterations to perform or a required fitness value of the best solution etc. In genetic algorithm different parameters such as size of initial population, cross over rate, mutation rate etc. are to be set. The performance of the genetic algorithm depends upon the value of genetic parameters. Many authors implemented GA to solve various problems using different values of genetic parameters.

N-Queens have to be placed on a check board of size N*N in this problem. The constraint is that none of the queen should attack on any other queen. As the number of queens increases, the size of the chess board also increases and the complexity of the algorithm to solve this problem also increases. NQueen problem can be solved using backtracking algorithm. The backtracking algorithm also take so much take time when the size of N is very large i.e. in hundreds or in thousands. Genetic algorithm is also solved using heuristic algorithm such as Ant Colony Optimization ACO, Genetic Algorithm GA etc. In this paper, the objective is to find best values of the genetic parameters to solve N-Queen NP-Hard problem. The next section illustrates the survey of the recent work done by other authors in this field.

1.1 Problem statement

To apply Genetic Algorithm for optimization of N queens Problem

1.2 Objective

- To apply Genetic Algorithm for various real-world problems
- To learn the N-Queen’s Problem
- To apply Genetic Algorithm for the optimization of N-Queen’s Problem
- To learn the Soft Computing Concepts

1.3 Software and Hardware Requirement

Software

- Operating System: Windows/Linux
- Programming Language: Python 3.8 and above
- Software (IDE): PyCharm or VSCode

Hardware

- Speed: 233MHz and above
- Hard disk: 10GB
- RAM: 512 MB

2.TERMINOLOGY

2.1 Population:

- Collection of All States that we consider for solving our problem.

2.2 Fitness

- It is solutions which is closest to the final solution.

2.3 Selection

- Selecting states that are closest to the solution (Fittest).

2.4 Crossover

- We interchange the values between selected state.

2.5 Mutation

- In this values are changed randomly like DNA in Biology.
- We can change any number of states to any value.

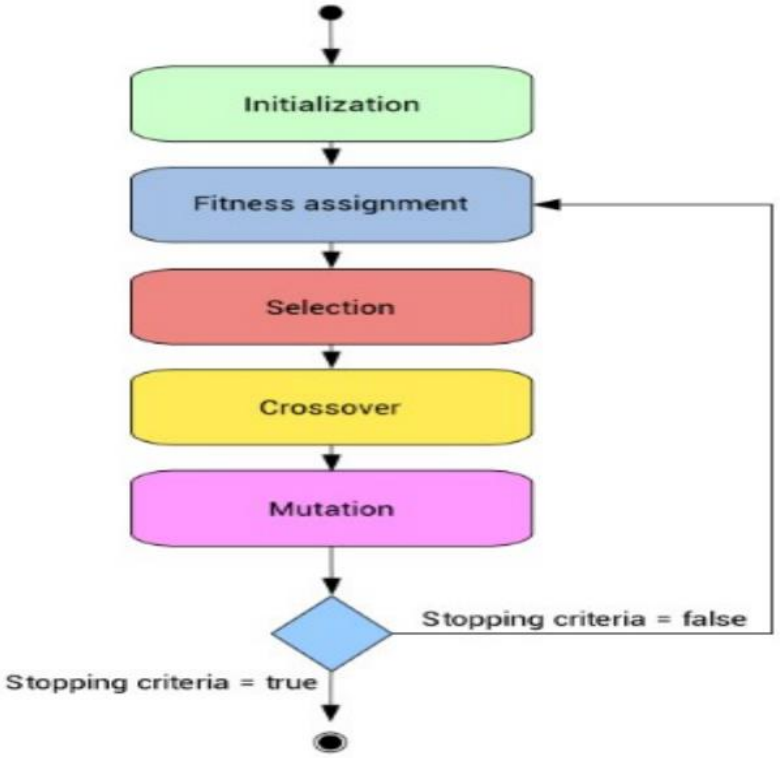
2.6 Gene

- A **gene** is a number between 0 to n-1.
- Is a position of any queen in the board

2.7 Chromosome

- A **chromosome** is an array of these genes. It could be the solution.

3. FLOWCHART:


Initialization
Fitness assignment
Selection
Crossover
Mutation
Stopping criteria = false
Stopping criteria = true

4. IMPLEMENTATION AND EVALUATION:

4.1 Genetic Algorithm :

Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as Evolutionary Computation.GA works on a population consisting of some solutions where the population size (pop size) is the number of solutions. Each solution is called individual. Also, each individual has a fitness value. To select the best individuals, a fitness function is used. The result of the fitness function is the fitness value representing the quality of the solution. The higher the fitness value the higher the quality the solution. The individuals in the mating pool are called parents. Every two parents selected from the mating pool will generate two offspring (children). A general outline how Genetic Algorithm (GA) works is given below:

1. A random population of candidate solutions is created and the fitness scores of the individuals are calculated and the chromosomes are sorted in the population and ranked according to the fitness values.
2. Certain number of chromosomes will pass onto the next generation depending on a selection operator, favoring the better individuals based on their ranking in the population.
3. Selected chromosomes acting as parents will take part in crossover operation to create children whose fitness values are to be calculated simultaneously. Crossover probability is generally kept high, because it is seen to give better children in terms of fitness values.
4. Then based on a mutation probability, a mutation operator is applied on new individuals which randomly changes few chromosomes. Mutation probability is generally kept low.
5. Evaluated off-springs, together with their parents form the population for the next generation

4.2 Steps to Solve N Queen Problem

1. Representing individuals .
2. Generating an initial Population.
3. Apply Fitness Function
4. Selecting parents for mating in accordance to their fitness.

5. Crossover of parents to produce new generation

6. Mutation of new generation to bring diversity.

4.3 Implementation

In this work, Genetic Algorithm is applied to solve N-Queen problem. The performance of N-Queen problem is analyzed on different values of three genetic parameters population size, cross over rate and mutation rate.

1. Population Size : A population is a group of candidate solutions of the problem. The quality of these solution varies by their fitness value. Population size determine the number of chromosomes in the population. Larger population size takes more time in performing genetic operations. In this work population size varies from 20 to 200 in steps of 20 and the performance of GA to solve N-Queen problem is analyzed. Initial population size must be optimum, as large population size may lead for genetic algorithm to take more time whereas less population size may not lead to good mating pools

2. Cross Over – Here, parents are selected and off springs are produced, using them. This is used for exploitation of the Search space. Here, crossover probability is entered by user. This probability is generally kept high. The rate of cross over determines how many chromosomes of the population will participate in the cross over. If the cross over rate is low, then the changes in the population are very less. But it takes very less time to perform cross over. If it taken high, then the changes in the population are very high and chances to get the solution in early generations also increases.

3. Mutation Rate – Here, a random change in chromosome to make a new solution is known as mutation. This is used for exploration of the Search space. This probability is generally kept less, as High mutation probability leads to random search. Mutation introduces some changes in the population. After mutation, the fitness value of the chromosome may increase or decrease. Generally, mutation rate is kept very low.

5.CODE AND OUTPUT

```
import random
import copy
import math
import numpy as np
import time


cross_prob = 0
mutation_prob = 0


def print_board(individual, n):
    print()
    print('--------------N queens Board--------------')
    print()
    solution = []
    for x in range(n): # empty board!
        solution.append(['*'] * n)
    for z in range(n):
        solution[n-individual[z]][z] = 'Q'
    for z in solution:
        brackets_removed = str(z)[1:-1]
        brackets_removed = brackets_removed.replace("'", " ")
        brackets_removed = brackets_removed.replace(",", "")
        print(brackets_removed)
        print()


def fitness(individual):
    row_clashes = abs(len(individual) - len(np.unique(individual)))
    diagonal_collisions = 0
    n = len(individual)
```

```
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + individual[i] - 1] += 1
        right_diagonal[len(individual) - i + individual[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 / (n-abs(i-n+1))
    attack_pairs = row_clashes + diagonal_collisions
    return attack_pairs

def crossover(individual1, individual2, prob1):
    if prob1 < cross_prob:
        n = len(individual1)
        c = random.randint(0, n - 1)
        return individual1[0:c] + individual2[c:n]
    else:
        return individual1

def mutation(individual, prob2):
    if prob2 < mutation_prob:
        n = len(individual)
        c = random.randint(0, n - 1)
        m = random.randint(1, n)
        individual[c] = m
```

```
        return individual
    else:
        return individual

def generate_individual(n):
    result = list(range(1, n + 1))
    np.random.shuffle(result)
    return result



class Genetic(object):

    def __init__(self, n, pop_size):
        self.queens = []
        for i in range(pop_size):
            self.queens.append(generate_individual(n))

    def generate_population(self, random_selections=5):
        candid_parents = []
        candid_fitness = []
        for i in range(random_selections):
            candid_parents.append(self.queens[random.randint(0, len(self.queens) - 1)])
            candid_fitness.append(fitness(candid_parents[i]))
        test_list = []
        for i in range(0, len(candid_parents)):
            test_list.append([candid_fitness[i], candid_parents[i]])
        test_list.sort()
        x = test_list[0]
        y = test_list[1]
```

```
    temp1 = crossover(x[1], y[1], random.random())
    temp2 = crossover(y[1], x[1], random.random())
    p = mutation(temp1, random.random())
    q = mutation(temp2, random.random())
    if fitness(p) < x[0]:
        if not any(list == p for list in self.queens):
            self.queens.append(p)
            test_list.append([fitness(p), p])
    if fitness(q) < x[0]:
        if not any(list == p for list in self.queens):
            self.queens.append(q)
            test_list.append([fitness(q), q])

def finished(self):
    for i in self.queens:
        if(fitness(i) == 0):
            return [1, i]
    return [0, self.queens[0]]

def start(self, random_selections=5):
    count = 0
    while self.finished()[0] == 0:
        count = count+1
        self.generate_population(random_selections)
    final_state = self.finished()
    print()
    print('populations generated:', count)
    print()
    print(('Solution : ' + str(final_state[1])))
    print_board(final_state[1], n)
```

```
# ********** N-Queen Problem With GA Algorithm **********

n = (int)(input('Enter the value of N : '))
max_pairs = (n*(n-1))/2
initial_population = (int)(input('Enter initial population size : '))
cross_prob = (float)(input('Enter crossover probablity:'))
mutation_prob = (float)(input('Enter mutation probablity:'))
begin_timer = time.time()
algorithm = Genetic(n=n, pop_size=initial_population)
algorithm.start()
print('Time Taken in seconds: ', time.time() - begin_timer)
```

********** Output **********

```
Enter the value of N : 8
Enter initial population size : 10
Enter crossover probablity:0.8
Enter mutation probablity:0.2

populations generated: 12206

Solution : [6, 1, 5, 2, 8, 3, 7, 4]

--------------N queens Board--------------

 *   *   *   *   Q   *   *   *

 *   *   *   *   *   *   Q   *

 Q   *   *   *   *   *   *   *

 *   *   Q   *   *   *   *   *

 *   *   *   *   *   *   *   Q

 *   *   *   *   *   Q   *   *

 *   *   *   Q   *   *   *   *

 *   Q   *   *   *   *   *   *

Time Taken in seconds:  45.61882424354553
```

6. ADVANTAGES AND DISADVANTAGES

6.1 Advantages

- Is faster and more efficient as compared to the traditional methods.
- GA is good for noisy environment.
- GA uses probabilistic transition rule, not deterministic rules.
- GA is easily parallelised.

6.2 Disadvantages

- GA is computationally expensive some time.
- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- If not implemented properly, the GA may not converge to the optimal solution.

7.CONCLUSION

In this project, Genetic Algorithm is applied to solve N-Queen problem. The performance of GA improves for high value of population size, cross over rate and mutation rate. However, the algorithm takes more time for high value of these parameters. In future the performance of GA can be analysed for larger instances of N-Queen problem.