An approach to solve n-Queens problem using Genetic Algorithm

Harshada Talnikar[1] , Shivaji Pansare[2]

[1]*Asst. Professor, Dept. Of Computer Science, SN Arts, DJM Commerce, BNS Sci College, Sangamner.Maharashtra, India*

[2]*Asst. Professor, Dept. Of Computer Science, SN Arts, DJM Commerce, BNS Sci College, Sangamner.Maharashtra, India*

ABSTRACT

Genetic algorithm (GAs) is powerful method which uses heuristic approach. It is capable of searching large spaces of possible solutions in an efficient manner. This paper approaches an implementation of Genetic Algorithm for solving n-Queens problem. It provides an efficient way to solve the problem than traditional backtracking method. In proposed genetic algorithm, members of the population are represented by number of chromosomes. Each chromosome is a possible board position in the n queens game. Instead of traditional approach the proposed algorithm gives efficient solution in terms of reduced time and efforts.

KEYWORDS : *Genetic Algorithm, n-Queens problem, chromosome, population, crossover.*

I. INTRODUCTION

The N-QUEENS PROBLEM is to place eight queens on a chessboard so that no two queens attack each other. It is combinatorial problem. This problem can be stated as placing *n* no attacking queens on an nxn chessboard. Since each queen must be on a different row and column, we can assume that queen *i* is placed in *i*-th column. All solutions to the *n*-queens problem can be represented as *n*-tuples (*q1*, *q2*, …, *qn*) that are permutations of an *n*-tuple (1, 2, 3, …, *n*). Position of a number in the tuple represents queen's column position, and its value represents queen's row position. This representation of the solution space where two of the constraints (row and column conflicts) are already satisfied should be searched in order to eliminate the diagonal conflicts also. Complexity of this problem is *O(n!)*.

A general approach to find solution for N-QUEENS PROBLEM:

- Create a solution matrix of the same structure as chess board.
- Whenever place a queen in the chess board, mark that particular cell in solution matrix.
- At the end print the solution matrix, the marked cells will show the positions of the queens in the chess board.

Algorithm:

```
1. Place the queens column wise, start from the left most column
2. If all queens are placed.
    1. return true and print the solution matrix.
3. Else
    1. Try all the rows in the current column.
```

2. Check if queen can be placed here safely if yes mark the current cell in solution matrix as 1 and try to solve the rest of the problem recursively.
3. If placing the queen in above step leads to the solution return true.
4. If placing the queen in above step does not lead to the solution , BACKTRACK, mark the current cell in solution matrix as 0 and return false.

4. If all the rows are tried and nothing worked, return false and print NO SOLUTION.

Fig 1 shows Unique Solutions 8-Queens Problem. The solution is represented as (3,6,2,7,14,8,5) means- queen1 is placed on row1 and column 3, queen2 is placed on row2 and column 6, queen3 is placed on row3 and column 2 and so on.

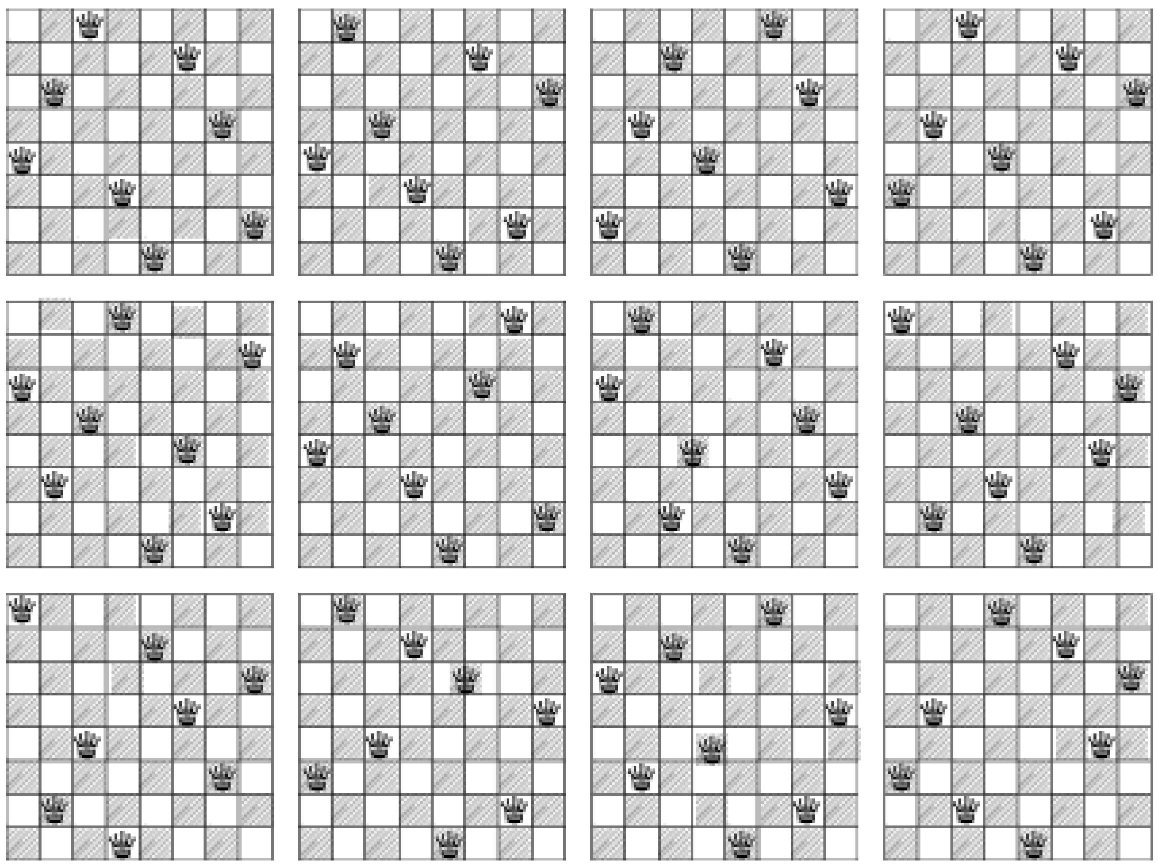

Fig 1 Solutions 8-Queens Problem

To determining a good fitness function for *n*-Queen problem is important. A fitness function determines how close a wrong solution is to a correct one. Since *n*-tuple representation eliminates row and column conflicts, wrong solutions have queens attacking each other diagonally. A fitness function can be designed to count diagonal conflicts. For a correct solution, the function will return zero.
For a simple method of finding conflicts [1],
consider an *n* tuple (*q1,..., qi,..., qj, ..., qn*). *i*-th and *j*-th queen share a diagonal if:

i- qi=j- qj or *i+ qi=j+ qj*

which reduces to: mod(*qi- qj)=mod(i-j)*

This simple approach results in fitness function with complexity of *O(n2)*. It is possible to reduce complexity to *O(n)* by observing diagonals on the board.

II. GENETIC ALGORITHMS

Genetic algorithms uses search and optimization procedures based on 3 principles

1 Selection
2. Crossover
3. Mutation.

GAs simulate those processes in natural populations which are essential to evolution. Exactly which biological processes are essential for evolution? In nature, individuals in a population compete with each other for resources such as food, water and shelter. Also, members of the same species often compete to attract a mate. Those individuals which are most successful in surviving and attracting mates will have relatively larger numbers of offspring. Poorly performing individuals will produce few of even no offspring at all. This means that the genes from the highly adapted, or "fit" individuals will spread to an increasing number of individuals in each successive generation. The combination of good characteristics from different ancestors can sometimes produce "super fit" offspring, whose fitness is greater than that of either parent. In this way, species evolve to become better suited to their environment. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. Potential solutions are obtained as individuals that are evaluated using a fitness function representing a problem being optimized.

Basic structure of a genetic algorithm is shown in the following list:

1. A random population of individuals (potential solutions) is created. All individuals are evaluated using a fitness function.
2. Certain number of individuals that will survive into next generation is selected using selection operator. Selection is somewhat biased, favoring "better" individuals.
3. Selected individuals act as parents that are combined using crossover operator to create children.
4. A mutation operator is applied on new individuals. It randomly changes few individuals (mutation probability is usually low)
5. Children are also evaluated. Together with parents they form the next generation.

Steps 2.-5 are repeated until a given number of iterations have been run, solution improvement rate falls below some threshold, or some other stop condition has been satisfied.

III. PROPOSED ALGORITHM

As genetic algorithms work by applying “Natural Selection” to a population so that only the fittest in the population can survive. In our implementation of the genetic algorithm members of the population are represented by number of chromosomes. Each chromosome is a possible board position in the n queens game.

All such possible generated chromosomes in the population mutate until only the “fittest”, which in our case is the solution with least conflicts among the queens remain.

At each iteration, the chromosomes are generated. The breed function given below select two random chromosomes from the population and combines. This breed function act as follows

- If both parents have same board positions, the child is assigned it.
- For the remaining position random numbers are generated to form chromosomes following the constraint not to have more than one queen in the same row.

```
CompareParents(cls, parent1, parent2)
   {
     if (len(parent1) != len(parent2))
    return "Error"
     if (parent1 == parent2)
           parent2 = Chromosome.random(len(parent2))
     possible = [i for i in range(len(parent1))]
     current = [-1 for i in range(len(parent1))]

     for i in range(len(parent1))
         if(parent1 == parent2[i])
             current[i] = parent1[i]
             possible.remove(current[i])

     for i in range(len(parent1))
         if(current[i] == -1)
           current[i] = random.choice(possible)
           possible.remove(current[i])

     return cls(current)

                 }
```

- Then using genetic mutation is applied to the chromosome according to probability. The mutation probability is compared against a random number. If the random number falls below the probability number the gene is mutated by switching two random positions in the chromosome.
- Then a new third random chromosome is selected and replaced using the newly created child chromosome.

```
  mutate(self, mutation_prob = .001)
     {
         if (random.random() < mutation_prob )
              index1 = random.randint(0,len(self)-1)
              index2 = random.randint(0,len(self)-1)
              temp = self.chromosome[index1]
              self.chromosome[index1] = self.chromosome[index2]
        self.chromosome[index2] = temp
}
```

At each iteration the population is sorted according to fitness it gets. The fitness of the chromosome is calculated using the number of conflicts in the board. This is calculated using the cost function.

```
Cost(self)
 {
        cost = 0;
        for i in range(len(self.chromosome))
            cost+=self.chromosome.count(self.chromosome[i]) - 1
            cost+=self.DiagonalCost(i)
        return cost
 }

 DiagonalCost(self,index)
  {
     cost = 0
     for i in range(len(self.chromosome))
        if(i != index)
           dx = abs(index - i)
           dy = abs(self.chromosome[index] - self.chromosome[i])
        if(dx == dy)
           cost= cost+1
```

```
    return cost
  }
```

The chromosome is fit if the number of conflicts is the lowest. If the fittest chromosome is a solution the algorithm ends. If not the iteration of n-queen continues until a solution is found or the limit to iterations is reached.

IV. RESULT

The following solutions can be constructed using the following table.

Solution #	**Elements show which column to use per chessboard row**							
	Row 0	**Row 1**	**Row 2**	**Row 3**	**Row 4**	**Row 5**	**Row 6**	**Row 7**
1	0	4	7	5	2	6	1	3
2	0	5	7	2	6	3	1	4
3	1	3	5	7	2	0	6	4
4	1	4	6	0	2	7	5	3
5	1	4	6	3	0	7	5	2
6	1	5	0	6	3	7	2	4
7	1	5	7	2	0	3	6	4
8	1	6	2	5	7	4	0	3
9	1	6	4	7	0	3	5	2
10	2	4	1	7	0	6	3	5
11	2	4	7	3	0	6	1	5
12	2	5	1	4	7	0	6	3

V. CONCLUSION

This paper attempted to solve *n*-Queen problem using genetic algorithms. *n*-Queen problem represents a large class of NP problems which cannot be solved in a reasonable amount of time using deterministic methods. Although using heuristic methods for solving problems genetic algorithms proved able to solve combinatory problems with simple "yes" and "no" answers. Furthermore, tests showed that GA is able to find different solutions for a given number of queens.

REFERENCES

[1] Ellis Horowitz and Sartaj Sahni, *Fundamentals of computer algorithms*, Computer Science Press Inc., Rockville, MD, 1978.

[1] I. Martinjak and M. Golub, “Com-parison of Heuristic Algorithms for the N-Queen Problem”, Proceedings of the ITI 2007 29th Int. Conf. on Information Technology Interfaces, June 25, 2007.

[2] K. D. Crawford, “Solving the N-Queens Problem Using GA”, In Proceedings ACM/SIGAPP Symposium on Applied Computing, Kansas City, 1992, pages 1039-1047.

[3] Božiković, Marko, G. "paralleling genetic algorithm", Faculty of Electrical Engineering and Computing, Zagreb, 22.05.2006.

[4] Sloane, Neil J. A., Number of ways of placing n non attacking queens on n x n board, The On-Line Encyclopedia of Integer Sequences id:A000170, http://www.research.att.com/~njas/sequence,s/A000170, (30.01.2007.)