COMPARATIVE ANALYSIS OF BACKTRACKING AND GENETIC ALGORITHM IN N QUEEN’S PROBLEM

Anuneet Kumar Dubey², Vijayan Ellappan², Rakesh Paul³, Vishal Chopra⁴

¹²³ School of Information Technology and Engineering (SITE), VIT University, Vellore, Tamil Nadu, India.

Email: evijayan@vit.ac.in

Received on 25-10-2016
Accepted on 02-11-2016

Abstract:

This paper presents a comparative analysis of two algorithms i.e: backtracking and genetic algorithm for the solution of N queen’s problem. In this paper the N queens problem is solved using both the algorithms. Both the algorithms have their own advantages and drawbacks. Both the methods of solving N Queen problem are entirely different. The first one is general method and takes time in days, months and years as N increases. In this paper the time taken by the two methods for a given value of N are compared.

Keywords: Backtracking, Genetic algorithm, N queens.

I. Introduction: The following paper illustrates about the n queens problem along with the 2 algorithms which is compared for the solutions of the problem i.e backtracking and genetic algorithms. The use of the algorithms,benefits of one algorithm over another algorithm. In each algorithm we have provided better explanations along with definitions and pseudo codes. Various papers contents is explained in the literature survey and results is presented before conclusion.

II. N Queens Problem: The problem states that we have to place N no of queens in a N X N matrix in such a way that no other two queens can capture each other . A queen can capture another queen if the other queen is in the same row, same column or diagonally with respect to the first queen. A German chess player named M Bezzel founded this problem in 1848.
III. Backtracking Algorithm

Backtracking algorithm provides a solution for the above problem. In backtracking algorithm we first place the first of the n queens in the first position then we compare the first queen with the another queen to find a suitable place in the board. According to the problem a queen can capture another queen if the queen is positioned in the same row or same column or is positioned diagonally to the first queen so after putting the first queen we check that whether the second queen that we are going to put is being captured by the $1^{st}$ one or not if the position is fine then it is fine otherwise we have to shift the position of the $2^{nd}$ queen Likewise we have to place all the n queens in the board if the queens does not find a suitable place in the table then we have to go back to the previous row, that's where the backtracking algorithm is used when we go back to the previous row we have to shift the position of that queen in the row so that the next queen can be put in the row.

The backtracking algorithm is described as follows

```
Algorithm 1: Backtracking general algorithm
Checks whether solution has been found
If found solution, return it
else for each choice that can be made
take that choice
Recurrence
If recursion gives a positive result, return it
If no choices remain, return failure
```

IV. Genetic Algorithm

Genetic algorithms are mainly used for solving search and optimizations problems by using adaptive methods. Like in biology we use the word genetic to characterize the features of a persons by his genes. By mimicking this process the genetic algorithm is used in the field of mathematics and computers for solving the problems that required informations in hierarchy.

In n queens we can use genetic algorithm for solving the problem. Genetic algorithm uses the concept of survival of the fittest, here the possible combinations of the n elements are represented as ‘chromosomes’ in another word chromosomes can be said as the board positions of the all queens.

The following diagram shows a board representation of the instance of solutions provided by genetic algorithm
as per the above chess board the ‘chromosomes’ of the above board can be written as (3,5,7,1,6,0,2,4). This representation ensures that there is no two queens are on the same column.

First of all we create a random population of chromosomes as per our no of queens. In this only one queen is in the row. The following function is used for implementing this

```python
def random(cls, num_queens=8):
    chromosome = [i for i in range(num_queens)]
    random.shuffle(chromosome)
    return cls(chromosome)
```

In the chromosomes the row number is there so every number is unique in the set.

[1] After each iteration breeding is done between chromosomes, we can select any two random chromosomes and combines them as called as breeding. The following steps are taken in breeding

(i) The common gene of both the parents is transferred to the child.

(ii) For the remaining leftout positions we generate random numbers but that number should not be presented in the row before

```python
def fromParents(cls, parent_one, parent_two):
    if (len(parent_one) != len(parent_two)):
        return "Error"

    if (parent_one == parent_two):
        parent_two = Chromosome.random(len(parent_two))

    possible = [i for i in range(len(parent_one))]
    current = [-1 for i in range(len(parent_one))]

    for i in range(len(parent_one)):
        if (parent_one[i] == parent_two[i]):
            current[i] = parent_one[i]
            possible.remove(current[i])

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

    return cls(current)
```
(iii) The mutation probability is predefined. According to that the genetic [2] mutation is introduced. We check the mutation probability against a random number. If the random number is below the probability number the gene is mutated.

(iv) Then another random chromosomes is choosed and it get replaced using the new child chromosomes.

```python
def mutate(self, mutation_prob = .001):
    if (random.random() < mutation_prob):
        indexOne = random.randint(0,len(self)-1)
        indexTwo = random.randint(0,len(self)-1)
        temp = self.chromosome[indexOne]
        self.chromosome[indexOne] = self.chromosome[indexTwo]
        self.chromosome[indexTwo] = temp
```

After each iteration we arrange the population on the basis of how fit they are which we calculate using the number of conflicts in the board. The following function is used for calculating it.

```python
def getCost(self):
    cost = 0;
    for i in range(len(self.chromosome)):
        cost+=self.chromosome.count(self.chromosome[i]) - 1
    cost+=self.getDiagonalCost()
    return cost
```

The lowest number of conflicts occured in the chromosomes is considered as fittest. The algorithm finishes after [5] finding the fittest chromosomes if not the cycle repeats again untill the fittest chromosomes is found.

V. Literature Survey

(1) Performance Analysis of N Queens Problem Using Backtracking and Genetic Algorithm

In the following paper comparison is done between 2 algorithms i.e backtracking and genetic algorithms for the implementations of n queens problem. The paper deal with the time complexity like the first one takes more time as compared to the second one. The value of N is set upto 50. Implementation is being carried out in MATLAB. Then the final result is in tabulated form.
(2) Solving N Queens Problem Using Various Algorithms

In this paper a survey is done on various algorithms such as ACO, Genetic algorithms, DNA Sticker Algorithm. These algorithms are characterised as Heuristic algorithms. These algorithms are used for optimization by modelling from physical and biological process in nature. This paper is a survey on various algorithms.

(3) Solving N Queens Problem By Prediction

In this paper a new alternative of the backtracking algorithm is used i.e solving by prediction. Backtracking may be considered as brute force in many cases but it is false.

So prediction is used which is more efficient and it predicts the solution more accurate. Backtracking is useful when the problem has more optional path.

(4) Solving 8 Queens Problem By Using Genetic algorithms, Simulated Annealing And Randomization Method

In this paper 3 algorithms are being presented for the solution of 8 queens problem. 8 queens problem contains 92 possible solutions. All 3 algorithms are being executed and time complexity has been measured among all 3 algorithms.

The results concluded that genetic algorithm and simulated annealing algorithm gives better performance than randomization method. And the best among them is simulated annealing.

VI. Results

As per the comparison between the two algorithms population size=500, generations=50 and the value of N=4 to 35 was calculated. Here backtracking was able to calculate the solutions because the value of N is taken upto 35. If the value is greater than 35 then it was very difficult to implement using backtracking. Then the genetic algorithm was used. The results is presented in the tabular form in table 1.

VII. Analysis And Conclusion

From the table we can conclude that backtracking is very much efficient for the values of N less than 10. Actually backtracking can work upto the value of N=35. When comparing to the larger values of N we find that genetic algorithm works better than backtracking algorithm.

And for the values greater than 40 backtracking can take almost 3 days to give results whereas using genetic algorithm can solve it in 20 seconds. So our final conclusion is that for the lower values of N backtracking is efficient whereas for higher value genetic algorithm executes faster in comparison to backtracking method.
Table 1: Execution time comparison of 2 algorithms.

<table>
<thead>
<tr>
<th>N</th>
<th>Backtracking</th>
<th>G A</th>
<th>Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.427</td>
<td>8.3085</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0.508</td>
<td>8.8790</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>0.470</td>
<td>9.1000</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>1.120</td>
<td>9.6120</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>1.330</td>
<td>10.6150</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>225.300</td>
<td>11.2720</td>
<td>5</td>
</tr>
<tr>
<td>35</td>
<td>1040.889</td>
<td>11.8450</td>
<td>2</td>
</tr>
</tbody>
</table>

VIII. References