NAME: SREE LAKSHMI HIRANMAYEE KADALI

20010920-T244

Problem Statement:

N queens problem describes a graph search problem which finds the target configuration/goal state while finding the solution. The problem mainly focuses on placing 8-Queens on an 8*8 chessboard such that no queens attack each other.

Problem Formulation:

States: Place 0-7 queens on the chess board.

Initial State: No goal configuration.

Operators: Add a queen to the leftmost column that doesn't contain a queen yet, considering only the rows that are not attacked by a queen that is previously on board.

Goal State: A configuration of no attacking of 8-queens on a chessboard 8*8.

N- Queens Placement Idea:

We start placing the first queen on the 0^{th} column of the chessboard. For every next column, we place the queen from the 0^{th} row of the column so that it does not attack any already placed queens. For every possible valid placement of the queen, the next queen and the next column is selected.

Thus,

- We check if there is a queen placed on the same row.
- We check if there is a queen placed on the upper diagonal.
- We check if there is a queen placed on the lower diagonal.

Solving N-Queens through Backtracking:

The main idea of Backtracking, is tracing back to the previous step and abandoning the current state if the candidate cannot possibly complete a valid solution. Any partial solution that contains two queens attacking each other is abandoned and backtracked to change the position of the previously placed queen.

When we place a new queen in a column, we check for clashes with already placed queens. If we find a row for which there is no clash/attack with the other placed queens, we mark this row and column as a part of solution. Else, we backtrack and try other combinations of previously placed queens.

Backtracking can be effectively done with Depth-First-Search (DFS). While Breadth-First Search (BFS) approaches to search every node at the present depth level to satisfy the given property and then moving prior to next level.

Comparing Depth-first Search and Breadth-First Search Solutions:

Depth-first Search traces each solution from the placement of root queen to final queen. It stores the first possible solution and then backtraces to test the next possible solution. Meanwhile, Breadth-first Search takes the approach of creating a list of all possible solutions for placing the queen in first row. Then it creates a list of all possible solutions for placing second row queens with the first-row possibilities. Then it checks the combinations of placing third row queen with the first two rows and so on. Once the final queen is placed to check all the possible combinations, the actual solutions of configuration of 8 queens' problem is retrieved.

After careful comparison, I feel backtracking using Depth-First Search is better compared to Breadth-first Search because of the following reasons:

Time Complexity: DFS with Backtracking is significantly faster compared to BFS because it immediately returns the first solution, while BFS calculates all possible combinations of solutions. BFS is more suitable for finding the shortest path.

Space Complexity: BFS requires more memory and is more complex because it requires to visit every node and remember all the possible solutions. While DFS has less space complexity and needs to store only a single path from the root to leaf node at once.

Execution Time: While there is no great difference in execution time, BFS is slow compared DFS.

Optimality: Since, BFS considers all the Neighbors first, it is not optimal for puzzle problems. While DFS produces a solution while exploring all paths through decision making and produces most optimal solutions in puzzle problems.

Analysing time complexity of search techniques:

Time Complexity Analysis of Depth First Search:

While considering the time complexity of N queens using Backtracking algorithm,

Let N be total number queens, n the number of left queens and (N-n) be already placed queens.

For, arranging n Queens on n*n chessboard, without satisfying the special conditions, we have to go through every position of n, which results in N^N arrangements. However, Backtracking reduces time complexity because it eliminates the dead ends when the approach doesn't lead to a solution. The recursive function, which checks for already placed queens, runs n times and for each iteration it calls function(row) because it will run only in safe cells, which turns out to nT(n-1) times and O(N^2). Overall, the time complexity for worst case is O(N!)

Time Complexity Analysis of Breadth-First Search:

If we consider the time complexity of BFS in N queens' problem. Since, the BFS approaches N queens problem row by row i.e., with one queen at a time. We can visualize a tree which give n

arrangements for the first queen. Next, the second queen is arranged with each arrangement of the 1st queen giving n*n arrangements for second queen. This leads us to derive total nodes of O(b^d+1). Therefore, the time complexity of BFS is:

1+b+b^2+……b^d=O(b^d). [exponential in the depth of solution d].

Comparing both cases, time complexity is worse in both the cases, but DFS is optimal than BFS in terms of space complexity.

Does Backtracking provide better solutions? If so, why?

The "Brute-force" algorithm approaches N queens' problem by looking through every position on an N board with N number of queens. If the queens cannot capture each other then it has found a solution else it checks all the possible solutions. This leads to a time complexity of $O(N^N)$ which leads to waste enormous number of resources and calculation. Brute-force can be considered pointless and impractical because it doesn't have a bounding function which prevents placing two queens in same row, column or diagonal.

Checking almost 62! ways to place remaining queens can be avoided if two queens are placed in same row/ column. All these unnecessary calculations can lead us to conclude Brute-force is not optimal for solving N-queens problem.

Optimizing this approach, if we prevent all the possible attack positions of queens, by backtracking to try another safe position, it helps us derive all solutions in a time complexity of O(N!) which is 100 times faster than brute force. In a way, Backtracking is more efficient and optimal compared to Brute-Force approach because it examines only 15,720 possible queen placements, by removing placements of queen in same row and column. It can be further optimized with pruning method.

Considering the case of genetic algorithm, the approach involves recombination of fittest individuals to produce successor states/offspring that populate the next generation. To apply genetic algorithm framework, we randomly select population states which involves a sequence of numbers where each index denotes the row number from left to right of each column queens. Applying the fitness function can help us select parents with highest fitness score. The first part of the first parent is then crossed with second part of second parent to produce a offspring. The fitness score of the off springs are checked and so on, to produce generations. The solution can be obtained in any generation.

While Backtracking is an effective and efficient solution to provide exact results in ample amount of time, genetic algorithm can provide good solutions to higher valued Queens

So, I feel backtracking is an efficient algorithm for providing solution for 8 Queens problem