Ai-project -> 1

1) project definition :

* N - Queens problem solver :

-> The N-Queens problem is to place n chess queens on an n by n chessboard so that no two queens are on the same row, column or diagonal. This puzzle dates to 1848, and only two years later a variant was introduced by Nauck (1850) in which some number of queens are pre-placed and the solver is asked to place the rest, if possible

It's the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. chessboard that no queens attack each other by being in the same row, column or diagonal.

It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

We aim to develop a program using the given algorithms that implements this problem and solve it.

2) Algorithms:

first algorithm -> Backtracking Algorithm:

the general definition:

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree). the algorithm tries to find a sequence path to the solution which has some small checkpoints from

where the problem can backtrack if no feasible solution is found for the problem.

N-queens case:

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

1) Start in the leftmost column.

2) If all queens are placed return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row
then mark this [row, column] as part of the
solution and recursively check if placing
queen here leads to a solution.

b) If placing the queen in [row, column] leads to
a solution then returns true.

c) If placing queen doesn't lead to a solution then
unmark this [row, column] (Backtrack) and go to
step (a) to try other rows.

3) If all queens are placed return TRUE.

4) If all rows have been tried and nothing worked,

return false to trigger backtracking.

Second algorithm -> Differential Evolution:

the general definition:

In evolutionary computation, differential evolution (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution about a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE is originally due to Storn and Price. Books have been published on theoretical and practical aspects of using DE in parallel computing, multi

objective optimization, constrained optimization, and the books also contain surveys of application areas. Excellent surveys on the multi-faceted research aspects of DE can be found in journal articles like.

N-queens case:

Every queen on a checker square can reach the other squares that are located on the same horizontal, vertical, and diagonal line. So there can be at most one queen at each horizontal line, at most one queen at each vertical line, and at most one queen at each of the 4n-2 diagonal lines. Furthermore, since we want to place as many queens as possible, namely exactly n queens, there must be exactly one queen at each horizontal line and at each vertical line.

Generate random solutions that cover the given space.
Evaluate each solution.
g=1
While (convergence is not reached)
For i=1 to Population size
Apply differential mutation.
Execute differential crossover
Clip the solutions if necessary
Evaluate the new solution.
Apply differential selection.
End
g=g+1;
End

3) Input explanation:

* In our project the user would feed the program with number of queens that wanted to implement or it can be predefined data.

4) Output explanation :

For solving n queens problem, we will try placing queen into different positions of one row. And checks if it clashes with other queens. If current positioning of queens if there are any two queens attacking each other. If they are attacking, we will backtrack to previous location of the queen and change its positions. And check clash of queen again.

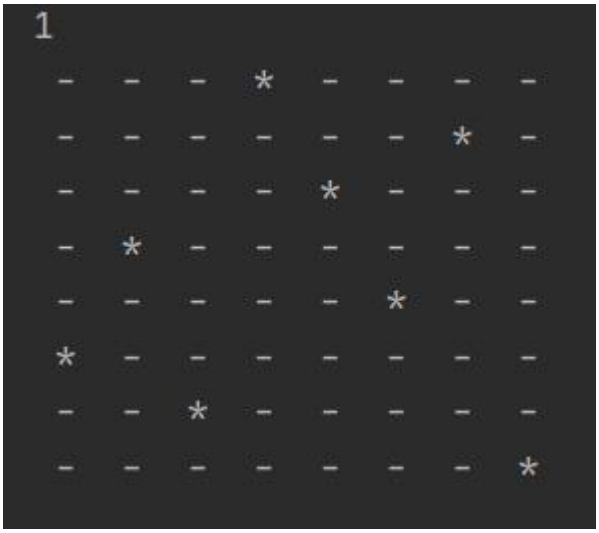
5) Screenshot of the output:

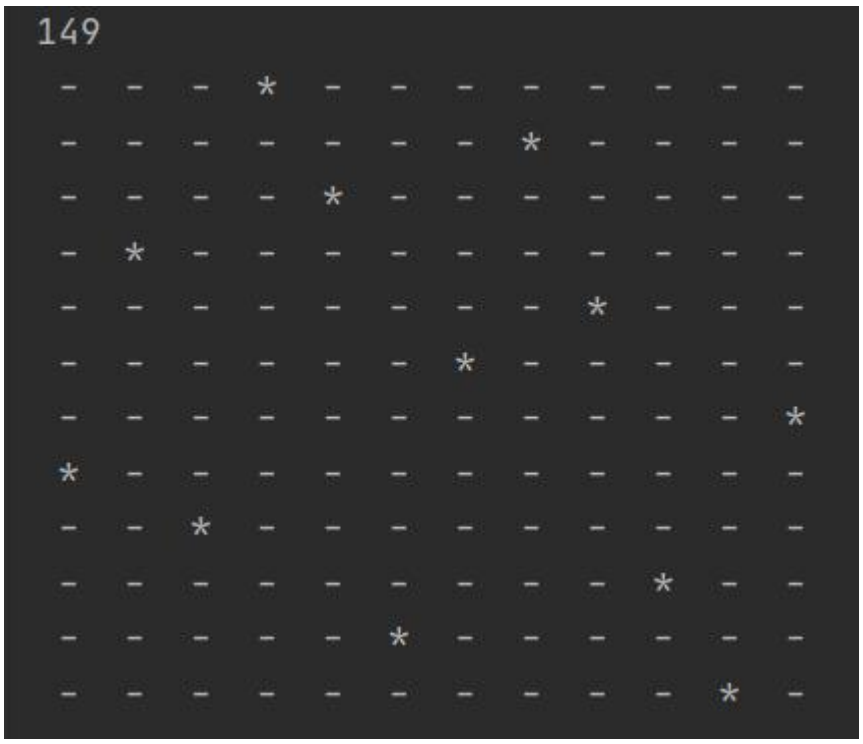
#Backtracking Algorithm:

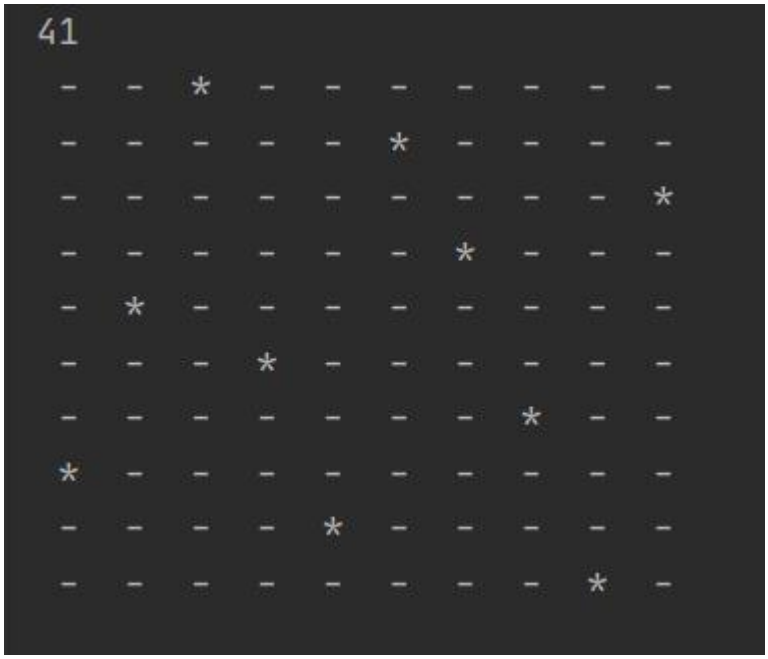
```
~ ~ ~ ~ ~ ~ ~ Q

~ ~ Q ~ ~ ~ ~ ~

Q ~ ~ ~ ~ ~ ~ ~

~ ~ ~ ~ ~ ~ Q ~

~ ~ ~ ~ Q ~ ~ ~
```

```
the solution   1
~ ~ Q ~

Q ~ ~ ~

~ ~ ~ Q

~ Q ~ ~

the solution   2
~ Q ~ ~

~ ~ ~ Q

Q ~ ~ ~

~ ~ Q ~

The Number Of Solutions of 8 Queen is 2
```

#Differential Evolution:


1


149



6) Source code URL:

https://github.com/Ahmedhesham9275/QueensGeneticAlgorithm

7) References

1- Data Structure Using C++ 2nd Edition

ISBN-13: 978-0-324-78201-1

ISBN-10: 0-324-78201-2

2- Introduction to Algorithms
Third Edition

ISBN 978-0-262-03384-8

ISBN-10:978-0-262-53305-8