**Project ID: 1**

**Project Name : N-Queen**

**Name : يوسف محمد سيد محمد (T . A) Id:202001105**

**Name :مروه سعيد اسعد Id:202000867**

**Name :ادهم احمد عيد Id:202000096**

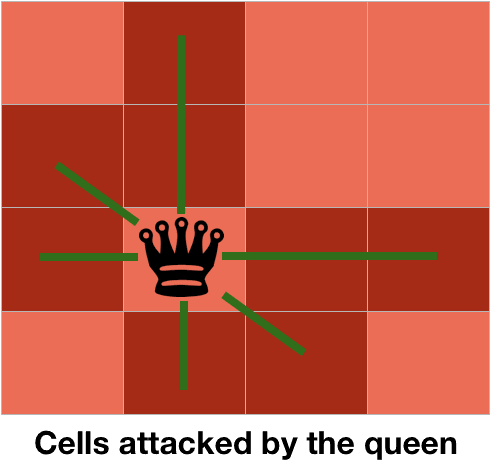
**Name :همسه الحسينى محمد Id:202001036**

**Name :حوريه محمود عبد العليم Id:202000282**

**Name :نورهان عبدالمولى عبدالمولى Id:202001011**

**Project Idea :**

**N Queen In Gineral :** Our goal is to arrange N queens on an NxN chessboard such that no queen can strike down any other queen. A queen can attack horizontally, vertically, or diagonally.

  
So, we start by placing the first queen anywhere arbitrarily and then place the next queen in any of the safe places. We continue this process until the number of unplaced queens becomes zero (a solution is found) or no safe place is left. If no safe place is left, then we change the position of the previously placed queen.

# Our Starting Point in the project :

* In our project the user will type an integer number grater than 3 to show the N\*NQueen chess board
* We have 2 algorithms to solve this problem :

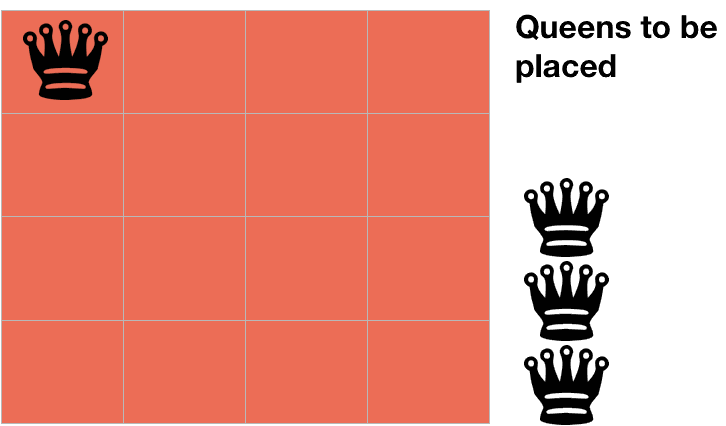
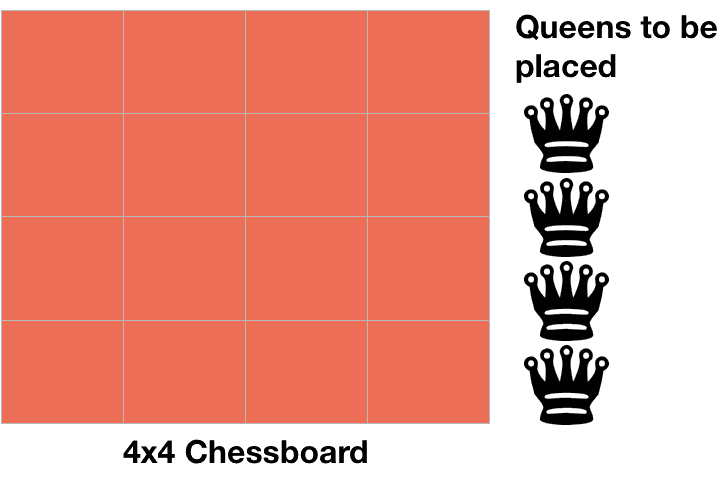
1- Backtracking Algorithm

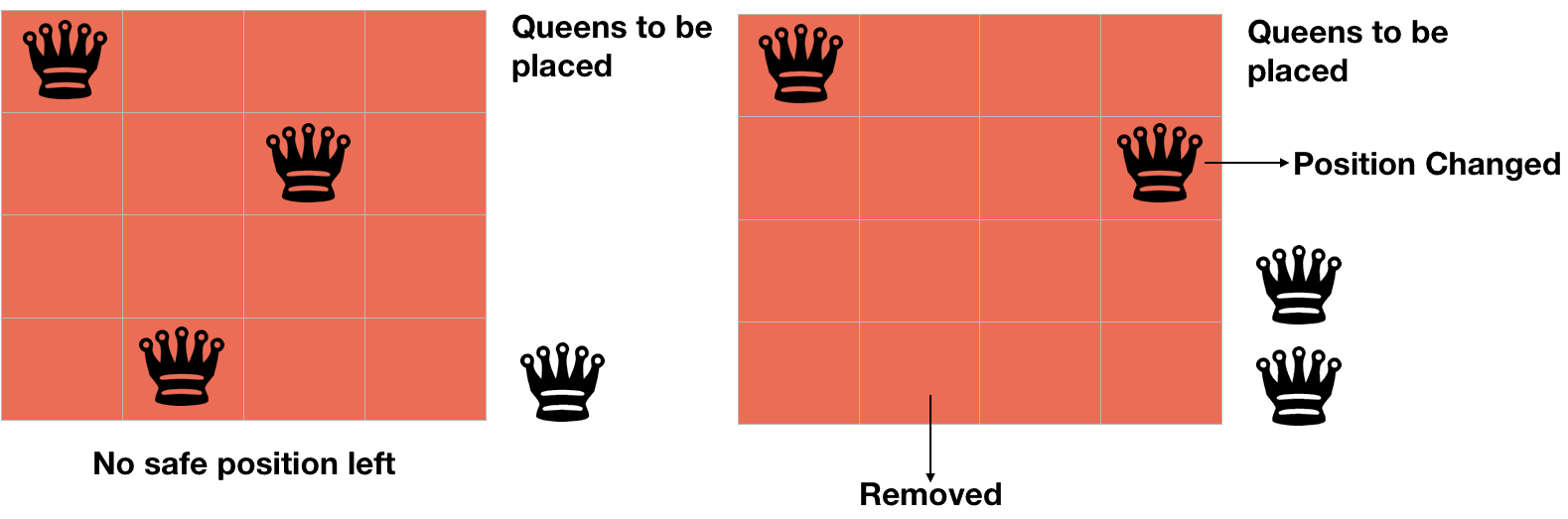
2- Differential Evolution Algorithm

we will explain the two algorithms in details in the next few pages

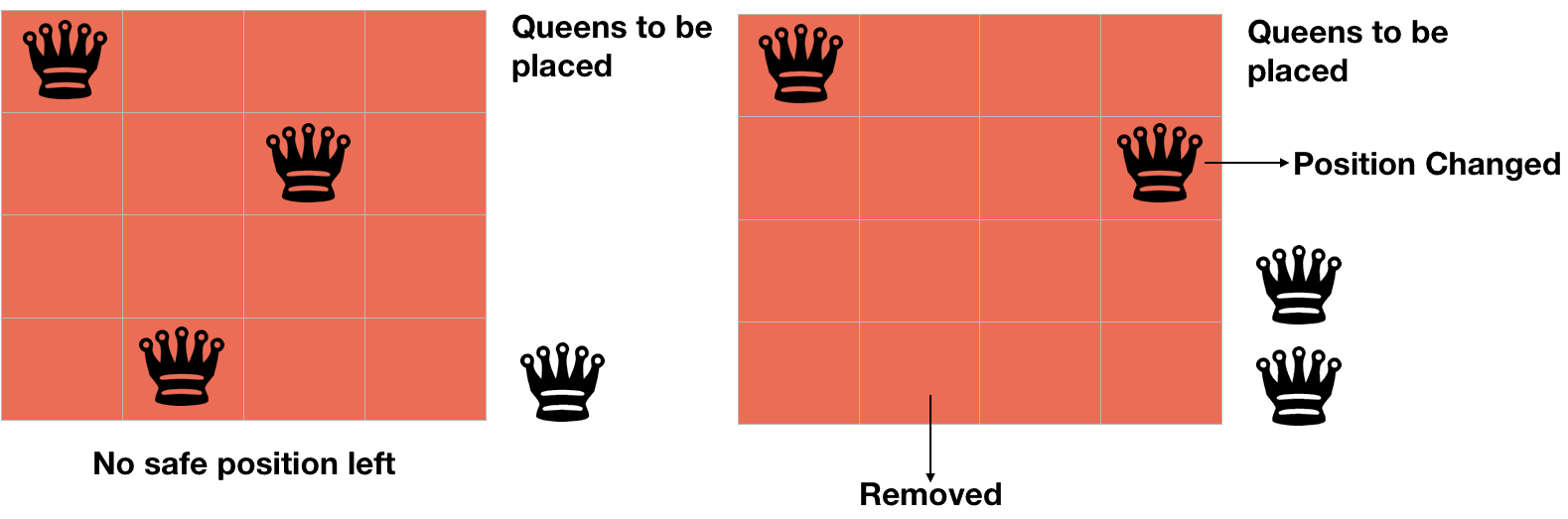
lets start …..

**A) N Queen With Backtracking Algorithm :** let’s explain with 4\*4 chess board so we have 4 queens to place in the board .

we will start by placing the first queen in the first row and place our second queen but we firstly ask if the next place **isSafe** or not , This functhion will be our firs function in our code . It will check if the place attacked from other queens or not however horizontally, vertically, or diagonally.



so we place the queen after asking if the position **isSafe** then place our second queen and so on using the **Solve** function if the number of queens reach **0** the function will return **True** . But if we took a see in this fig we still have one queen does not placed on the board so we use the backtrack and that means that we well back to the last step and choose another position as shown in the next fig



we will repeat this steps until finding the final solution .

**B) N Queen With Differential Evolution Algorithm:**

IN our code We use three Main functions and addition ones Let’s Explane them

**Mutate**: DE generates new individuals in the population by adding weighted difference between two population vectors to a third vector (the mutated vector).

**Crossover:** The mutated vector’s parameters are then mixed with the parameters of another pre-determined vector, the target vector, to yield a trial vector

**Selection(random\_pick):** If the trial vector yields a lower cost function value than the target vector, the trial vector replaces the target vector in the next generation**.**

**The functionalities of n queens…**

1. **N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal so once one queen is put on board, no queens on same row or column can be deployed**
2. **The n-queens problem is about finding how many different ways queens can be placed on a chessboard so that none attack each other**
3. **N-queens problem occurs in chess**
4. **a queen can move any number of steps in any direction**
5. **The only constraint is that it can’t change its direction while it’s moving**.
6. **There are 12 unique solutions to this problem. Two solutions are not unique if you can ”mirror” one solution to find the other, or if you can rotate the board to find**

**the other solution, or a combination of the two moves.**

1. **the number of ways to place a N queens on a NxN board can get very large since we have N^2 choices at first, then N^2 -1 , N^2 -2 and so on... leading to overall time complexity of O(N^2N)**
2. **It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3**  **there's no possible way to fit 3 queens in a 3x3 chess board without creating any conflict.**
3. **n-queens problem can be solved using backtracking. It can also be solved using branch and bound**
4. **the queen may move any number of unoccupied squares in any direction -- horizontally, vertically, or diagonally**
5. **n-queens problem can be solved using backtracking. It can also be solved using branch and bound.**
6. **the exact number of solutions is only known for *n* ≤ 27**

**So it can be played at any level (it depends on the size that the player will select)**

**Ex : 4\*4 8\*8 16\*16 …….**

**We will choose 8\*8 to discuss….**

The **eight queens puzzle** is based on the classic stategy games problem which is in this case putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The color of the queens is meaningless in this puzzle, and any queen is assumed to be able to attack any other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n**queens puzzle** of placing n8 queens on an n×n chessboard

**Similar App In Life**

**CHESS (N-QUEENS) :**

• • N-QUEENS is similar to the game of chess (CHESS) that is found in the market. It is a military movement in the game of chess, but with the difference that the queens will not attack each other, and that will lead to a tie between the white and black queen, and this movement when There are only royals on the playing field, and the chess.

**DESCRIPTION** : is a board game between two players. It is sometimes called Western chess or international chess

• The current form of the game emerged in Spain and the rest of Southern Europe during the second half of the 15th century after evolving from chaturanga, a similar but much older game of Indian origin. Today, chess is one of the world's most popular games

• Chess is an abstract strategy game and involves no hidden information. It is played on a chessboard with 64 squares arranged in an eight-by-eight grid. At the start, each player controls sixteen pieces: one king, one queen, two rooks, two bishops, two knights, and eight pawns. The player controlling 2 the white pieces moves first, followed by the player controlling the black pieces. The object of the game is to checkmate the opponent's king, whereby the king is under immediate attack (in "check") a nd there is no way for it to escape. There are also several ways a game can end in a draw

**A Literature Review**

## -A backtracking algorithm is a problem-solving algorithm that uses a **brute force approach** for finding the desired output.

The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

This approach is used to solve problems that have multiple solutions.

Like N queens problem

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

The prototypical backtracking problem is the classical n Queens Problem, first proposed by German chess enthusiast Max Bezzel in (under his pseudonym “Schachfreund”) for the standard 8 x 8 board and by François-Joseph Eustache Lionnet in for the more general n x n board. The problem is to place n queens on an n x n chessboard, so that no two queens are attacking each other.

This paper introduces the Backtracking Search Optimization Algorithm (BSA), a new evolutionary algorithm (EA) for solving real-valued numerical optimization problems. EAs are popular stochastic search algorithms that are widely used to solve non-linear, non-differentiable and complex numerical optimization problems. Current research aims at mitigating the effects of problems that are frequently encountered in EAs, such as excessive sensitivity to control parameters, premature convergence and slow computation. In this vein, development of BSA was motivated by studies that attempt to develop simpler and more effective search algorithms. Unlike many search algorithms, BSA has a single control parameter. Moreover, BSA’s problem-solving performance is not over sensitive to the initial value of this parameter. BSA has a simple structure that is effective, fast and capable of solving multimodal problems and that enables it to easily adapt to different numerical optimization problems. BSA’s strategy for generating a trial population includes two new crossover and mutation operators. BSA’s strategies for generating trial populations and controlling the amplitude of the search-direction matrix and search-space boundaries give it very powerful exploration and exploitation capabilities. In particular, BSA possesses a memory in which it stores a population from a randomly chosen previous generation for use in generating the search-direction matrix. Thus, BSA’s memory allows it to take advantage of experiences gained from previous generations when it generates a trial preparation. This paper uses the Wilcoxon Signed-Rank Test to statistically compare BSA’s effectiveness in solving numerical optimization problems with the performances of six widely used EA algorithms: PSO, CMAES, ABC, JDE, CLPSO and SADE. The comparison, which uses 75 boundary-constrained benchmark problems and three constrained real-world benchmark problems, shows that in general, BSA can solve the benchmark problems more successfully than the comparison algorithms.

## References

1. J. Derrac et al.

### [A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms](https://www.sciencedirect.com/science/article/pii/S2210650211000034)

### **Swarm Evol. Comput.**

(2011)

1. R. Thangaraj et al.

### [Particle swarm optimization: hybridization perspectives and experimental illustrations](https://www.sciencedirect.com/science/article/pii/S0096300310012555)

### **Appl. Math. Comput**

(2011)

1. I.G. Tsoulos et al.

### [Enhancing PSO methods for global optimization](https://www.sciencedirect.com/science/article/pii/S0096300310004315)

### **Appl. Math. Comput.**

(2010)

1. R. Thangaraj et al.

### [Particle swarm optimization: hybridization perspectives and experimental illustrations](https://www.sciencedirect.com/science/article/pii/S0096300310012555)

### **Appl. Math. Comput.**

(2011)

* <https://codeahoy.com/learn/recursion/ch10/>
* <http://mathcenter.oxford.emory.edu/site/cs171/nQueensProblemAndBacktracking/>

**Review 2**  
 In this paper, a qantum inspired differential evolution algorithm for solving the N-queens problem is presented.The N-queens problem aims at placing N queens on an NxN chessboard, in such a way that no queen could capture any of theothers. The proposed algorithm is a novel hybridization between differential evolution algorit*hms and quantum computing  principles. Accordingly, differential evolution algorithms have been enhanced by the adoption of some quantum concepts suchas quantum bits and states superposition. The use of the quantum interference has allowed this hybrid approach to have aremarkable efficiency and good results.*

Differential evolution (DE) is an efficient and powerful population-based stochastic search technique for solving optimization problems over continuous space, which has been widely applied in many scientific and engineering fields. However, the success of DE in solving a specific problem crucially depends on appropriately choosing trial vector generation strategies and their associated control parameter values. Employing a trial-and-error scheme to search for the most suitable strategy and its associated parameter settings requires high computational costs. Moreover, at different stages of evolution, different strategies coupled with different parameter settings may be required in order to achieve the best performance. In this paper, we propose a self-adaptive DE (SaDE) algorithm, in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Consequently, a more suitable generation strategy along with its parameter settings can be determined adaptively to match different phases of the search process/evolution. The performance of the SaDE algorithm is extensively evaluated (using codes available from P. N. Suganthan) on a suite of 26 bound-constrained numerical optimization problems and compares favorably with the conventional DE and several state-of-the-art parameter adaptive DE variants

In this paper, we have presented a new approach for solving the NQP. A quantum-inspired differentialevolution algorithm has been employed. Thisalgorithm is a hybridization of two well knownalgorithms. The first is the DEA that uses thedifferential evolution operator as alternative of classical GAs' mutation. The second is the QGA, proposed for the first time in the works of Han [4] andwhich was spread in an exponential manner to touchmany fields. A QGA has been used for solving the N-Queens problem [3] and has given promising results.Compared with CGA [4, 5] and QGA, our hybridalgorithm has given better results either in term of computation time or in term of fitness evolution. Toconclude, we can say that the quantum-inspireddifferential evolution algorithm is well situated to beamong the good alternatives to solve combinatorialoptimization problems, especially in term of efficiencyand algorithmic complexity. As future work, we willtry to apply this approach for solving other combinatorial optimization problems

References

1. <https://ieeexplore.ieee.org/abstract/document/4632146>
2. <https://www.academia.edu/4379206/A_Quantum_Inspired_Differential_Evolution_Algorithm_for_Solving_the_N_Queens_Problem>

References

[1]

Cengiz E., Sarkeshik S., and Murat T., “DifferentPerspectives of the N-Queens Problem,”

*in Proceedings of ACM 1992 Computer ScienceConference*

, Kansas City, pp. 12-15, 1992.[2]

Draa A., Talbi H., and Batouche M., “AQuantum Inspired Differential EvolutionAlgorithm for Rigid Image Registration,”

*in Proceedings of the International Conference onComputational Intelligence*

, Istanbul, pp. 408-411, 2004.[3]

Draa A., Talbi H., and Batouche M., “AQuantum Inspired Genetic Algorithm for Solvingthe N-Queens Problem,”

*in Proceedings of the 7*

*th*

*International Symposium on Programming and Systems*

, Algiers, pp. 145-152, 2005.[4]

Han K. and Kim J., “Genetic Quantum Algorithmand its Application to CombinatorialOptimization Problem,”

*in Proceedings of the2000 Congress on Evolutionary Computation*

,USA, pp. 1354-1360, 2000.[5]

Han K. and Kim J., “Quantum InspiredEvolutionary Algorithm for a Class of Combinatorial Optimization,”

*Computer Journal of IEEE Transactions on EvolutionaryComputation*

, vol. 6, no. 6, pp. 55-59, 2000.[6]

Karaboga D. and Okdem S., “A Simple andGlobal Optimization Algorithm for EngineeringProblems: Differential Evolution Algorithm,”

*Computer Journal of Turk Electronic*

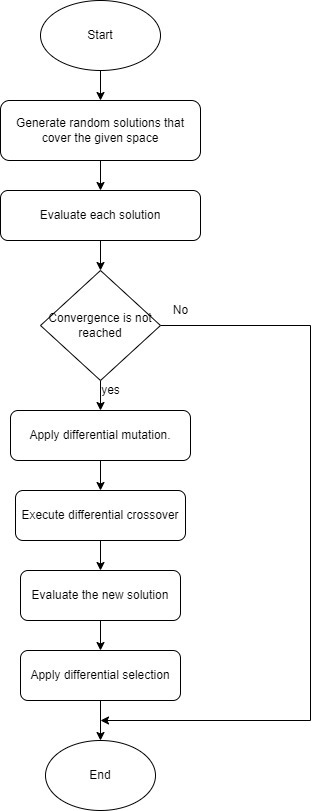
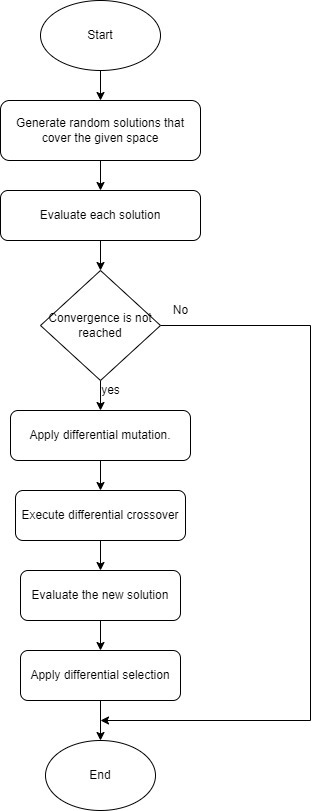
**Review 3**

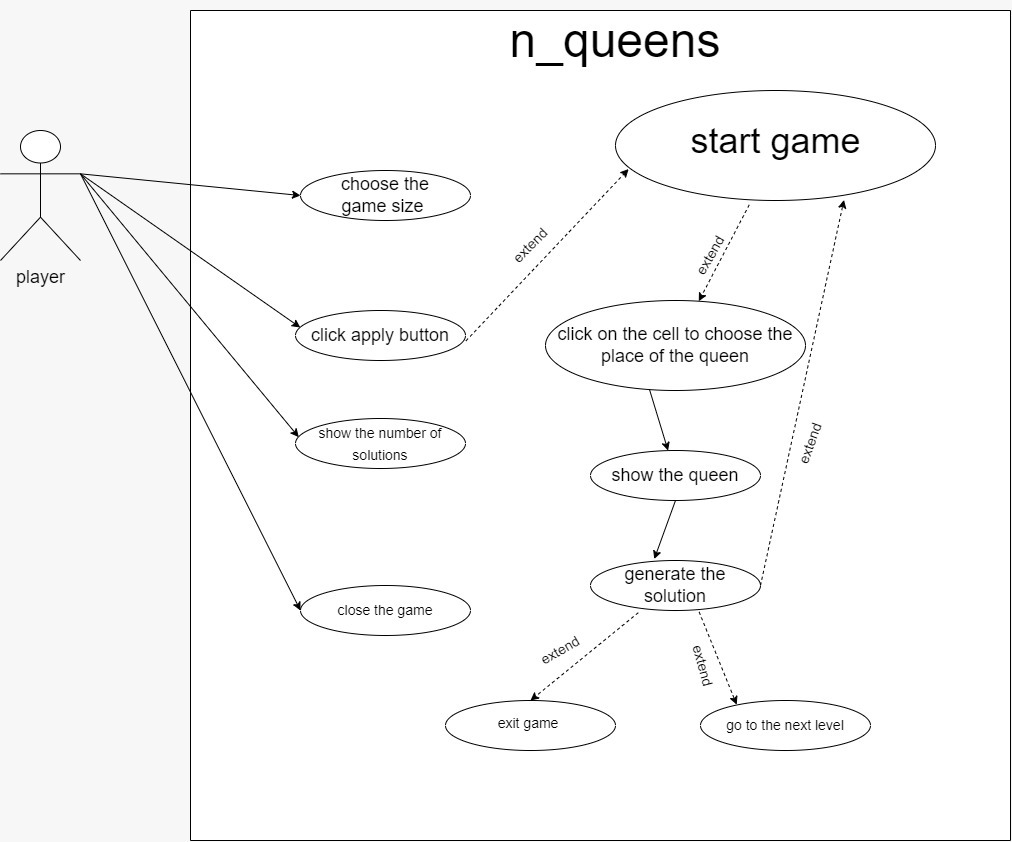
In this paper, a qantum inspired differential evolution algorithm for solving the N-queens problem is presented.The N-queens problem aims at placing N queens on an NxN chessboard, in such a way that no queen could capture any of theothers. The proposed algorithm is a novel hybridization between differential evolution algorithms and quantum computing  principles. Accordingly, differential evolution algorithms have been enhanced by the adoption of some quantum concepts suchas quantum bits and states superposition. The use of the quantum interference has allowed this hybrid approach to have aremarkable efficiency and good results.  
  
  
Differential evolution (DE) is an efficient and powerful population-based stochastic search technique for solving optimization problems over continuous space, which has been widely applied in many scientific and engineering fields. However, the success of DE in solving a specific problem crucially depends on appropriately choosing trial vector generation strategies and their associated control parameter values. Employing a trial-and-error scheme to search for the most suitable strategy and its associated parameter settings requires high computational costs. Moreover, at different stages of evolution, different strategies coupled with different parameter settings may be required in order to achieve the best performance. In this paper, we propose a self-adaptive DE (SaDE) algorithm, in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Consequently, a more suitable generation strategy along with its parameter settings can be determined adaptively to match different phases of the search process/evolution. The performance of the SaDE algorithm is extensively evaluated (using codes available from P. N. Suganthan) on a suite of 26 bound-constrained numerical optimization problems and compares favorably with the conventional DE and several state-of-the-art parameter adaptive DE variants  
  
In this paper, we have presented a new approach for solving the NQP. A quantum-inspired differentialevolution algorithm has been employed. Thisalgorithm is a hybridization of two well knownalgorithms. The first is the DEA that uses thedifferential evolution operator as alternative of classical GAs' mutation. The second is the QGA, proposed for the first time in the works of Han [4] andwhich was spread in an exponential manner to touchmany fields. A QGA has been used for solving the N-Queens problem [3] and has given promising results.Compared with CGA [4, 5] and QGA, our hybridalgorithm has given better results either in term of computation time or in term of fitness evolution. Toconclude, we can say that the quantum-inspireddifferential evolution algorithm is well situated to beamong the good alternatives to solve combinatorialoptimization problems, especially in term of efficiencyand algorithmic complexity. As future work, we willtry to apply this approach for solving other combinatorial optimization problems

References

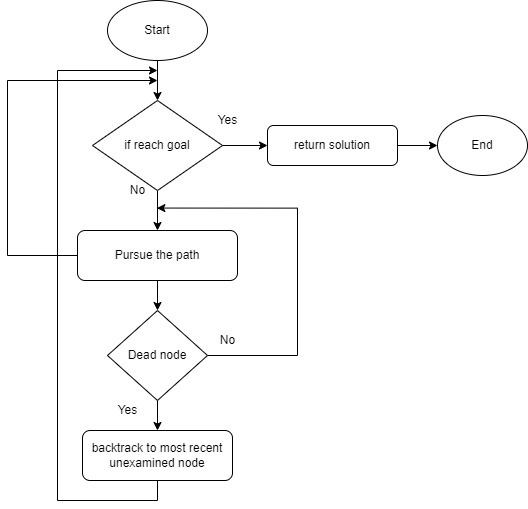
1. <https://ieeexplore.ieee.org/abstract/document/4632146>
2. [**https://www.academia.edu/4379206/A\_Quantum\_Inspired\_Differential\_Evolution\_Algorithm\_for\_Solving\_the\_N\_Queens\_Problem**](https://www.academia.edu/4379206/A_Quantum_Inspired_Differential_Evolution_Algorithm_for_Solving_the_N_Queens_Problem)

**Diagrams**

**Flow chart for the D.E**

**The Use Case Diagram**

**Flow chart of Backtrack Flowchart**



# Backtraching bloking diagram for the code

# 

1. **Block diagram for Backtrack algo**
2. **Backtracking Flowchart for Game**

# 

# Details of the algorithm(s)/approach(es)

The N-queens problem has been coded in several languages. However, the deeper question is the efficiency of the algorithmic strategies employed. The methodology here will be to examine various algorithms to determine the cost of the solution. By comparing the Backtracking Algorithm and Differential Evolution

## **First brief explanation on Backtracking Algorithm:**

Backtracking is a technique for systematically trying all paths through a state space it involves decisions per state.

Backtracking uses the Depth-First search method, when it starts exploring the solutions a bounding function is applied so that the algorithm can check if the so-far built solution satisfies the constrains if it does, it continues searching if it doesn’t the branch would be eliminated, and the algorithm goes back to the level before.

In our problem N-Queens consider 4\*4 chessboard, with the 4-Queens problem we placed a queen at a given point since we cannot have 2 queens in the same column, row, or diagonal (Bounding function),all this spaces were blocked out so it move to the next available space to place the next queen and this continuous until all 4 queens are placed onto the board if there is a problem with trying to find the solution then the backtracking occurs trying to find the next best place to put the queen in.

State-space tree of the 4-queens problem:

Diagram, engineering drawing

Description automatically generated

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.

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

4) If all rows have been tried and nothing worked, return false to trigger backtracking.

The solution of above steps in 4-Queens is (2,4,1,3) there is also another solution for this size chessboard is (3,1,4,2)

## **Second brief explanation on Differential Evolution:**

Differential Evolution is direct search method it satisfies the first requirement since it can be used to minimize any function for which a global minimum exists, uses a population of vectors where the stochastic perturbation of the vectors can be done independently so parallelization is easy.

It includes three important evolutionary operators mutation, crossover and selection to optimize (using a global-search metaheuristic) an objective function over the course of successive generations

Uses N parameters vectors each of dimension D and is primarily suited for numerical optimization problems and also NP doesn’t change during the algorithm’s lifetime.

* **Mutation:** For each vector Xi,G in generation G a mutant vector Vi,G is defined by

Vi,G = Xa,G +F(Xc,G −Xb,G)  
where i = {1,2,...,Np} and a, b, and c are mutually different random integer indices selected from {1,2,..., Np}.

Further, i, a, b, and c are different so that Np ≥4 is required.

F ∈ [0,2] is a real constant which determines the amplification of the added differential variation of (Xc,G−Xb,G). Larger values for F result in higher diversity in the generated population and lower values cause faster convergence.

* **Crossover**: generate new solutions by shuffling competing vectors and also to increase the diversity of the population.

It defines the following trial vector: Ui,G =(U1i,G,U2i,G,...,UDi,G)

Uji,G =Vji,G if randj(0,1) ≤ Cr ∨ j = k, and Uji= Xji,G otherwise

Cr ∈ (0,1) is the predefined crossover rate, and randj(0,1) is the jth evaluation of a uniform random number generator. k ∈{1, 2,...,D} is a random parameter index, chosen once for each i to make sure that at least one parameter is always selected from the mutated vector If randj ≤ CR, then Uij = Vij, else Uij = Xij.

* **Selection:** decide which vector (Ui,G or Xi,G) should be a member of next (new) generation, G +1. For a minimization problem, the vector with the lower value of objective function is chosen (greedy selection).

mutation, crossover, and selection are repeated Np (population size) times to generate a new population. These successive generations are produced until meeting the predefined termination criteria.

# Details of the algorithm(s)/approach(es)

The N-queens problem has been coded in several languages. However, the deeper question is the efficiency of the algorithmic strategies employed. The methodology here will be to examine various algorithms to determine the cost of the solution. By comparing the Backtracking Algorithm and Differential Evolution

## First brief explanation on Backtracking Algorithm:

Backtracking is a technique for systematically trying all paths through a state space it involves decisions per state.

Backtracking uses the Depth-First search method, when it starts exploring the solutions a bounding function is applied so that the algorithm can check if the so-far built solution satisfies the constrains if it does, it continues searching if it doesn’t the branch would be eliminated, and the algorithm goes back to the level before.

In our problem N-Queens consider 4\*4 chessboard, with the 4-Queens problem we placed a queen at a given point since we cannot have 2 queens in the same column, row, or diagonal (Bounding function),all this spaces were blocked out so it move to the next available space to place the next queen and this continuous until all 4 queens are placed onto the board if there is a problem with trying to find the solution then the backtracking occurs trying to find the next best place to put the queen in.

State-space tree of the 4-queens problem:

Diagram, engineering drawing

Description automatically generated

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.

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

4) If all rows have been tried and nothing worked, return false to trigger backtracking.

The solution of above steps in 4-Queens is (2,4,1,3) there is also another solution for this size chessboard is (3,1,4,2)

Testing the Backtracking algorithm

Chart, line chart

Description automatically generated

Table

Description automatically generated

## Second brief explanation on Differential Evolution:

Differential Evolution is direct search method it satisfies the first requirement since it can be used to minimize any function for which a global minimum exists, uses a population of vectors where the stochastic perturbation of the vectors can be done independently so parallelization is easy.

It includes three important evolutionary operators mutation, crossover and selection to optimize (using a global-search metaheuristic) an objective function over the course of successive generations

Uses N parameters vectors each of dimension D and is primarily suited for numerical optimization problems and also NP doesn’t change during the algorithm’s lifetime.

* **Mutation:** For each vector Xi,G in generation G a mutant vector Vi,G is defined by

Vi,G = Xa,G +F(Xc,G −Xb,G)  
where i = {1,2,...,Np} and a, b, and c are mutually different random integer indices selected from {1,2,..., Np}.

Further, i, a, b, and c are different so that Np ≥4 is required.

F ∈ [0,2] is a real constant which determines the amplification of the added differential variation of (Xc,G−Xb,G). Larger values for F result in higher diversity in the generated population and lower values cause faster convergence.

* **Crossover**: generate new solutions by shuffling competing vectors and also to increase the diversity of the population.

It defines the following trial vector: Ui,G =(U1i,G,U2i,G,...,UDi,G)

Uji,G =Vji,G if randj(0,1) ≤ Cr ∨ j = k, and Uji= Xji,G otherwise

Cr ∈ (0,1) is the predefined crossover rate, and randj(0,1) is the jth evaluation of a uniform random number generator. k ∈{1, 2,...,D} is a random parameter index, chosen once for each i to make sure that at least one parameter is always selected from the mutated vector If randj ≤ CR, then Uij = Vij, else Uij = Xij.

* **Selection:** decide which vector (Ui,G or Xi,G) should be a member of next (new) generation, G +1. For a minimization problem, the vector with the lower value of objective function is chosen (greedy selection).

mutation, crossover, and selection are repeated Np (population size) times to generate a new population. These successive generations are produced until meeting the predefined termination criteria.

And in the N-Queens problem the i element in the array represents the i column on the chessboard and the i element’s value, j, represents the j row on the chessboard.

Testing For 8x8 chessboard

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Testing for 16\*16 chessboard

**Development Platforms :**

**1-** Programming language (Python)

**2- IDE** VS

**3- Lyberarys** ( inspect , numpy.random.mtrand ,

operator , random , numpy , pygame , time

**References :**

https://code.energy/8-queens-problem/

https://www.digitalocean.com/community/tutorials/n-queens-problem-java-c-plus-plus

https://www.academia.edu/4379206/A\_Quantum\_Inspired\_Differential\_Evolution\_Algorithm\_for\_Solving\_the\_N\_Queens\_Problem

https://www.researchgate.net/publication/220413939\_A\_Quantum-Inspired\_Differential\_Evolution\_Algorithm\_for\_Solving\_the\_N-Queens\_Problem