Solving 8 Queen Problem By Using Simulated Annealing (SA)

# Abstract—

This paper introduced a metaheuristic algorithm for solving 8-queens problem in addition to find one of the 92 possible solutions for 8\*8 chess board. The Metaheuristics algorithm is Simulated Annealing (SA). The proposed method depends mainly on randomization in both the initialization phase and moving phase that is used to find all the solutions. The obtained results were promising as the SA algorithm is efficient in finding the solutions and also better than the randomization method. Also it has been found that SA was better than the GA as it required less number of steps in finding the solutions

# I. INTRODUCTION -

The aim of N-queens is to place N non attacking queens on an N\*N chess board. This is a generalization of the problem of putting eight non attacking queens on a chessboard, which was first posed in 1848 by M. Bezzel, a German chess player, in the Berliner Schachzeitung [1]. The goal of NQP problem is to arrange N queens on an N\*N chessboard in a way that there is no intersection between them at both vertical or horizontal or diagonal directions. EQP (Eight Queen Problem) is a special case of NQP, it is trying to find a way to place eight queens on a chessboard depending on NQP objective that is discussed above [2,3]. In this paper we propose a Simulated Annealing algorithm to generate a possible solution for eight queens’ problem by using different random initial solutions and calculating the fitness for each solution For each algorithm the intersection of a queen with another queen is calculated to be the fitness for each solution, of course the correct solution is the solution with fitness equal to zero (i.e no intersection).

# II. 8-QUEENS PROBLEM-

The N-queens problem is introduced in 1850 by Carl Gauss and has been studied for many decades by scientists. The N-queens problem is an effort to find a placement of N queens on an N by N chess board so that no two queens attack each other [4]. The number of correct solutions for N queens placed in N\*N chessboard is fixed. For N queens from 1 to 15 the number of possible solutions are shown in Table 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| solution | 1 | 0 | 0 | 2 | 10 | 4 | 40 |
| N | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| solution | 92 | 352 | 724 | 2680 | 14200 | 73712 | 365596 |

TABAL 1: NUMBER OF SOLUTIONS FOR N-QUEENS PROBLEM

# III. SIMULATED ANNEALING (SA)-

The annealing process requires heating and then slowly cooling to obtain a strong crystalline structure. In this paper the main objective is to reach all the 92 possible solutions of 8-queens problem. So the SA repeats cooling process until finding a new solution. The temperature will be set to the current correct solution and decreases until finding new solution. Two loops are applied in the system, the first one for finding new solution and compare it with the current correct solution while the second loop is for generating random neighborhoods, evaluate the object function and compare them with the current solution as shown in algorithm.

Algorithm: Simulated Annealing Algorithm:

Input: Initial random solutions and Starting Temperature (Tmax).

Output: All possible solutions for eight queens problem.

Step1: S = current solution generated randomly.

Step2: Evaluate the fitness function for the current solution F(S).

Step3: T=Tmax. The T value is different from solution to another.

Step4: Generate random neighborhood solution to be a new solution (S'). The neighborhood generation is done by using initial solution and then using full control moving (moving the attacking queens to new non intersected positions.

Step5: Evaluate the objective function of the new solution F(S').

Step6: ∆E = F(S') − F(S); (∆E is the change in the objective function between S and S'). If ∆E value in negative then the new solution is better than the current solution, so replace the new solution with the current solution.

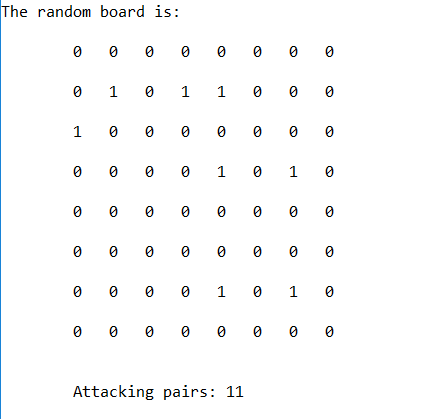
Step8: Update T using the linear updating method; T = T − β, where β is a specified constant value. Step9: Repeat steps 5-8 until the T value equal to zero which means a new solution is found.

# IV. RESULTS AND DISCUSSION-

The table shows the results of applying the simulated annealing algorithm to find the 92 solutions for EQP, with number of iteration that are required to achieve each solution. Also the temperature value for each solution is shown in table.

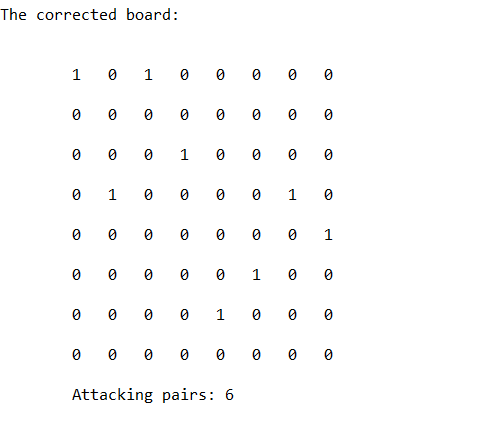
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No | iteration | T | No | iteration | T | No | iteration | T |
| 1 | 2 | 2 | 32 | 152 | 2 | 63 | 289 | 3 |
| 2 | 9 | 7 | 33 | 153 | 1 | 64 | 294 | 5 |
| 3 | 12 | 3 | 34 | 154 | 1 | 65 | 303 | 9 |
| 4 | 18 | 6 | 35 | 157 | 3 | 66 | 303 | 0 |
| 5 | 25 | 7 | 36 | 162 | 5 | 67 | 304 | 1 |
| 6 | 27 | 2 | 37 | 165 | 3 | 68 | 308 | 4 |
| 7 | 35 | 8 | 38 | 166 | 1 | 69 | 317 | 9 |
| 8 | 44 | 5 | 39 | 174 | 8 | 70 | 319 | 2 |
| 9 | 44 | 0 | 40 | 175 | 1 | 71 | 322 | 3 |
| 10 | 50 | 6 | 41 | 175 | 0 | 72 | 327 | 5 |
| 11 | 58 | 8 | 42 | 181 | 6 | 73 | 332 | 6 |
| 12 | 63 | 5 | 43 | 183 | 2 | 74 | 338 | 6 |
| 13 | 66 | 3 | 44 | 188 | 5 | 75 | 346 | 6 |
| 14 | 66 | 0 | 45 | 196 | 8 | 76 | 352 | 6 |
| 15 | 67 | 1 | 46 | 200 | 4 | 75 | 355 | 3 |
| 16 | 75 | 8 | 47 | 208 | 8 | 78 | 357 | 2 |
| 17 | 79 | 4 | 48 | 216 | 8 | 79 | 365 | 8 |
| 18 | 80 | 1 | 49 | 224 | 8 | 80 | 366 | 1 |
| 19 | 81 | 1 | 50 | 230 | 6 | 81 | 371 | 5 |
| 20 | 86 | 5 | 51 | 234 | 4 | 82 | 373 | 2 |
| 21 | 88 | 2 | 52 | 236 | 2 | 83 | 382 | 9 |
| 22 | 92 | 4 | 53 | 241 | 5 | 84 | 387 | 5 |
| 23 | 96 | 4 | 54 | 245 | 4 | 85 | 395 | 8 |
| 24 | 103 | 9 | 55 | 252 | 7 | 86 | 404 | 9 |
| 25 | 109 | 6 | 56 | 254 | 2 | 87 | 412 | 8 |
| 26 | 118 | 9 | 57 | 262 | 8 | 88 | 413 | 1 |
| 27 | 125 | 7 | 58 | 267 | 5 | 89 | 422 | 5 |
| 28 | 130 | 5 | 59 | 271 | 4 | 90 | 433 | 1 |
| 29 | 139 | 9 | 60 | 278 | 7 | 91 | 435 | 2 |
| 30 | 141 | 2 | 61 | 281 | 3 | 92 | 444 | 9 |
| 31 | 150 | 9 | 62 | 286 | 5 |  |  |  |

A random board was generated from our source code:



Where we can see that number of attacking pair is 11

This is the output:



Where number of attacking pairs is 6.

# V. CONCLUSIONS-

This paper showed that the 8-queens problem can be solved by using metaheuristics. The metaheuristic algorithm used here is Simulated Annealing algorithm. SA is applied and gave good results compared with the randomization algorithm. The results of randomization algorithm show that the number of movements (swaps) needed to reach the optimal solution are varied, this due to the randomness in the solutions’ construction. Also it has been found that SA is better than GA as it needed about half the number of iterations that GA needed to find all the possible solutions of the 8-queens problem. This is due to the nature of the SA algorithm, which allows the probabilistic acceptance of a non-improving neighbor solution.

# SOURCE CODE-

#include <bits/stdc++.h>

#include <cstdlib>

#include <cmath>

#include <ctime>

#define size 8

using namespace std;

void simulated\_annealing(int\*\* current); //function to apply the simulated annealing algorithm

void gen\_successor(int\*\* present, int\*\* neighbor); //function to generate successor

int attacking\_pairs(int\*\* board); //function to calculate # of attacking pairs queen

void assign\_board(int \*\*present, int \*\*target); //function to assign a board to another

double rand\_ZeroToOne(); //function to generate random number between 0 and 1

void print\_board(int\*\* board); //function to print a board

int main()

{

int a,b;

int\*\* random\_board= (int\*\*)malloc((size+1)\*sizeof(int\*)); //allocating memory row-wise

for(int i=1; i<=size; ++i)

random\_board[i]= (int\*)malloc((size+1)\*sizeof(int)); //allocating memory column-wise

for(int i=1; i<=size; ++i) //initializing the board to be empty

for(int j=1; j<=size; ++j)

random\_board[i][j]=0;

for(int i=1; i<=size; ++i) //to place 8-queens randomly

{ a= rand() %size +1;

b= rand() %size +1;

if(random\_board[a][b]==1)

i--;

random\_board[a][b]=1;}

cout<<"\nThe random board is: "<<endl;

print\_board(random\_board);

cout<<endl<<"\tAttacking pairs: "<<attacking\_pairs(random\_board)<<endl;

simulated\_annealing(random\_board); //applying simulated annealing algorithm to the random board

return 0;}

void simulated\_annealing(int\*\* current)

{ double T= 100.0;

double dE;

double pro;

double ran;

int currAttPairs;

int nextAttPairs;

int\*\* next;

next= (int\*\*)malloc((size+1)\*sizeof(int\*)); //allocating memory row-wise

for(int i=1; i<=size; ++i)

next[i]= (int\*)malloc((size+1)\*sizeof(int)); //allocating memory column-wise

while(true)

{

if(T<0) //--------------------change was made here

break;

gen\_successor(current,next); //generating a successor

currAttPairs= attacking\_pairs(current);

nextAttPairs= attacking\_pairs(next);

if(currAttPairs==0)

{cout<<"Iteration Needed:"<<100-T<<endl;

break;}

dE= currAttPairs-nextAttPairs; //difference between current and next

if(dE>=0)

assign\_board(next,current);

else

{

pro= exp(dE/T);

//ran= rand\_ZeroToOne(); //generating a random number between 0 and 1

if(0.80<pro) //if less than 50% do nothing

assign\_board(next,current);

}

T-=0.5; // decrement by 0.5

}

cout<<endl<<"The corrected board: "<<endl<<endl;

print\_board(current);

cout<<"\tAttacking pairs: "<<currAttPairs<<endl;

}

void gen\_successor(int\*\* present, int\*\* neighbor)

{

assign\_board(present,neighbor); //assigning present board to neighbor

int x= rand() %size +1; //random generation of position

int y= rand() %size +1;

if(neighbor[x][y]==0) //condition to identify if the position contains a queen

{

gen\_successor(present, neighbor); //recursive call of the function

return;

}

else

{

int x1= (x%8) +1;

int y1= (y%8) +1; //calculating position to find successor

if(neighbor[x1][y1]==1)

{

gen\_successor(present, neighbor); //recursive call of the function

return;

}

else //condition to generate a successor

{

neighbor[x1][y1]=1;

neighbor[x][y]=0;

}

}

}

int attacking\_pairs(int\*\* board)

{

int count=0;

for(int j=1; j<=size; ++j)

{

for(int i=1; i<=size; ++i)

{

if(board[i][j]==1) //condition to find queen in a position

{

for(int k=i+1; k<=size; ++k) //to check pairs row-wise

{

if(board[k][j]==1)

{

count++;

break;

}

}

for(int k=j+1; k<=size; ++k) //to check pairs column-wise

{

if(board[i][k]==1)

{

count++;

break;

}

}

for(int m=i+1,n=j+1; m<=size && n<=size; ++m,++n) //to check pairs upper diagonally

{

if(board[m][n]==1)

{

count++;

break;

}

}

for(int m=i-1,n=j+1; m>=1 && n<=size; --m,++n) //to check pairs lower diagonally

{

if(board[m][n]==1)

{

count++;

break;

}

}

}

}

}

return count;

}

void assign\_board(int\*\* present,int\*\* target)

{

for(int i=1; i<=size; ++i)

for(int j=1; j<=size; ++j)

target[i][j]= present[i][j];

}

double rand\_ZeroToOne()

{

srand((unsigned)time(NULL)); //seeding a random number using computer's internal clock

return ((double)rand()/RAND\_MAX); //generating a random number and transferring it in between 0 and 1

}

void print\_board(int\*\* board)

{

cout<<"\n";

for(int i=1;i<=size;++i)

{

cout<<"\t";

for(int j=1;j<=size;++j)

cout<<board[i][j]<<" ";

cout<<endl<<endl;

}

}