Table of Contents

Note:

- Follow step 2 if Python 3.7.3 is not installed on the local machine.
- Follow step 3 if MiniSat version 1.14 is not installed on the Linux machine.
- Follow step 4 and 5 if project has not been downloaded to the local machine.
- More information about GUI: Click link.

N-queens Problem

The n-queens puzzle is the problem of placing N queens on an N×N chessboard such that no two queens attack each other.

To encode the n-queens (n is a given number) problem into the DIMACS CNF format and solve it by the SAT solver, we need to encode the following constraints

1) no two queens on the same row,
2) no two queens on the same column, and
3) no two queens on the same diagonal.

Concept

Let P_{ij} denote the propositional letter for the position (i, j) of the board, which has size $N \times N$: the i^{th} row, and the j^{th} column of the board where i can be $1, 2, 3, \ldots, N$ and j can be $1, 2, 3, \ldots, N$. If P_{ij} is true; there is a queen on (i, j); otherwise, there is no queen

P_{11}	P_{12}	...	P_{1j}	...	P_{1N}
P_{21}	P_{22}	...	P_{2j}	...	P_{2N}
...	...	...	...	...	...
P_{i1}	P_{i2}	...	P_{ij}	...	P_{iN}
...	...	...	...	...	...
P_{N1}	P_{N2}	...	P_{Nj}	...	P_{NN}

ROW

Let us first consider rows. For such a row, there should be at least one true, and at most one true. At least one, since in total, there should be N queens. So on every row, there should be at least one queen. Moreover, on every row, there should be at most one queen; otherwise, they can hit each other. That means P_{ij} and $P_{i'j'}$ are in the same row if and only if $i = i'$.

P_{11}	P_{12}	...	P_{1j}	...	P_{1N}
P_{21}	P_{22}	...	P_{2j}	...	P_{2N}
...	...	...	...	...	...
P_{i1}	P_{i2}	...	P_{ij}	...	P_{iN}
...	...	...	...	...	...
P_{N1}	P_{N2}	...	P_{Nj}	...	P_{NN}

Let us start with at least one queen. To express the observation that there is a queen for i^{th} row, that is easily expressed in the following formula.

$$P_{i1} \vee P_{i2} \vee \ldots \vee P_{ij} \vee \ldots \vee P_{iN}$$

At least one of these Ps should be true. It can be expressed by simply taking the or of all of these propositional letters. Moreover, in shorthand notation, we can write it in this way.

$$\bigvee_{j=1}^{N} P_{ij}$$

The next is, at most, one of them is true. If there is a sequence of propositional letters, shown below, and at most one may be true, that means if we take two of them, then it is not allowed that both are true.

$$P_{i1}, P_{i2}, \dots, P_{ij}, \dots, P_{iN}$$

So if we take two of them, if one is true, then the other one is false; that's expressed in the following formula, for i^{th} row.

$$\begin{gathered}
P_{i1} \rightarrow \neg P_{i2}, \\
P_{i1} \rightarrow \neg P_{i3}, \\
\dots, \\
P_{i1} \rightarrow \neg P_{iN}, \\
P_{i2} \rightarrow \neg P_{i1}, \\
P_{i2} \rightarrow \neg P_{i3}, \\
P_{i2} \rightarrow \neg P_{i4}, \\
\dots, \\
P_{i2} \rightarrow \neg P_{iN} \\
\dots, \\
P_{iN} \rightarrow \neg P_{i(N-1)}
\end{gathered}$$

In other words, if we take two of them, at least one should be false. It can be expressed in the following formula, for i^{th} row.

$$\begin{gathered}
\neg P_{i1} \vee \neg P_{i2}, \\
\neg P_{i1} \vee \neg P_{i3}, \\
\dots, \\
\neg P_{i1} \vee \neg P_{iN}, \\
\neg P_{i2} \vee \neg P_{i1}, \\
\neg P_{i2} \vee \neg P_{i3}, \\
\neg P_{i2} \vee \neg P_{i4}, \\
\dots, \\
\neg P_{i2} \vee \neg P_{iN} \\
\dots, \\
\neg P_{iN} \vee \neg P_{i(N-1)}
\end{gathered}$$

If we remove the duplicate cases, it will be expressed in the formula not P_{ij} or not P_{ik} for every j less than k or expressed in the following formula, for i^{th} row.

$$\neg P_{ij} \vee \neg P_{ik}, \forall j < k$$

Moreover, in shorthand notation, we can write it in this way.

$$\bigwedge_{0<j<k\leq N} (\neg P_{ij} \vee \neg P_{ik})$$

COLUMN

P_{11}	P_{12}	...	P_{1j}	...	P_{1N}
P_{21}	P_{22}	...	P_{2j}	...	P_{2N}
...	...	...	...	...	...
P_{i1}	P_{i2}	...	P_{ij}	...	P_{iN}
...	...	...	...	...	...
P_{N1}	P_{N2}	...	P_{Nj}	...	P_{NN}

Next, we consider columns that have exactly the same requirements. The only difference is that the i and the j are swapped. So, that's expressed in the following formula.

For at least one queen on j^{th} column,

$$P_{1j} \vee P_{2j} \vee ... \vee P_{ij} \vee ... \vee P_{Nj}$$

In shorthand notation,

$$\bigvee_{i=1}^{N} P_{ij}$$

For at most one queen on j^{th} column,

$$\neg P_{ij} \vee \neg P_{kj}, \forall i < k$$

In shorthand notation,

$$\bigwedge_{0<i<k\leq N} (\neg P_{ij} \vee \neg P_{kj})$$

So until now, we have requirements for at least one queen on every row, at most one queen on every row, at least one queen on every column, at most one queen on every column. That's expressed in the following formula.

For at least one queen on every row,

$$\bigwedge_{i=1}^{N} \bigvee_{j=1}^{N} P_{ij}$$

For at most one queen on every row,

$$\bigwedge_{i=1}^{N} \bigwedge_{0<j<k\leq N} (\neg P_{ij} \vee \neg P_{ik})$$

For at least one queen on every column,

$$\bigwedge_{j=1}^{N} \bigvee_{i=1}^{N} P_{ij}$$

For at most one queen on every column,

$$\bigwedge_{j=1}^{N} \bigwedge_{0<i<k\leq N} (\neg P_{ij} \vee \neg P_{kj})$$

DIAGONAL

Furthermore, now it remains to express the requirements on diagonals. On every diagonal, there should be at most one queen. Let us consider such a diagonal.

P_{11}	P_{12}	...	P_{1j}	...	P_{1N}
P_{21}	P_{22}	...	P_{2j}	...	P_{2N}
...	...	...	...	...	...
P_{i1}	P_{i2}	...	P_{ij}	...	P_{iN}
...	...	...	...	...	...
P_{N1}	P_{N2}	...	P_{Nj}	...	P_{NN}

We can see that two elements are on the same diagonal P_{ij} and $P_{i'j'}$ if they have the same sum. In this case, the Orange color, the sum is $N+1$. By $1+N, 2+(N-1), \dots, (N-1)+2, N+1$. Moreover, it is not allowed that two of these variables are both true by following constraints that no two queens on the same diagonal. So, that's expressed in the following formula.

$$\neg P_{ij} \vee \neg P_{i'j'} \leftrightarrow i + j = i' + j'$$

In the other direction, we have something similar. Nevertheless, now the requirement is not the sum but expresses the difference.

P_{11}	P_{12}	...	P_{1j}	...	P_{1N}
P_{21}	P_{22}	...	P_{2j}	...	P_{2N}
...	...	...	...	...	...
P_{i1}	P_{i2}	...	P_{ij}	...	P_{iN}
...	...	...	...	...	...
P_{N1}	P_{N2}	...	P_{Nj}	...	P_{NN}

P_{ij} and $P_{i'j'}$ are on such a diagonal if and only if $i - j = i' - j'$. So, that's expressed in the following formula.

$$\neg P_{ij} \vee \neg P_{i'j'} \leftrightarrow i - j = i' - j'$$

So for all i, j, i', j' in such a way that (i, j) is different from (i', j'). However, they are on the same diagonal, expressed by $i + j = i' + j'$ or $i - j = i' - j'$. We have the requirement that it's not allowed that both are true. So not P_{ij} or not $P_{i'j'}$ should hold. More, we can do this for all i, j, i', j'. Nevertheless, then we have many things double. We may restrict to the cases where i is less than i'. Here, we have a big formula.

$$\bigwedge_{0<i<i'\leq N} \left(\bigwedge_{j,j':i+j=i'+j' \vee i-j=i'-j'} \neg P_{ij} \vee \neg P_{i'j'} \right)$$

We let i and i' run over all the values from 1 to N and j and j' run on all the values in such a way that they corresponding two positions are on the same diagonal.

OVERALL

Now we have collected all ingredients of the total formula, and here is the total formula presented.

$$\left(\bigwedge_{i=1}^{N} \bigvee_{j=1}^{N} P_{ij} \right) \wedge \left(\bigwedge_{i=1}^{N} \bigwedge_{0<j<k\leq N} (\neg P_{ij} \vee \neg P_{ik}) \right) \wedge \left(\bigwedge_{j=1}^{N} \bigvee_{i=1}^{N} P_{ij} \right) \wedge$$
$$\left(\bigwedge_{j=1}^{N} \bigwedge_{0<i<k\leq N} (\neg P_{ij} \vee \neg P_{kj}) \right) \wedge \left(\bigwedge_{0<i<i'\leq N} \left(\bigwedge_{j,j':i+j=i'+j' \vee i-j=i'-j'} \neg P_{ij} \vee \neg P_{i'j'} \right) \right)$$

Pseudocode

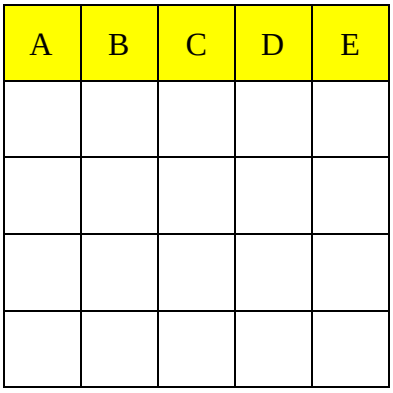
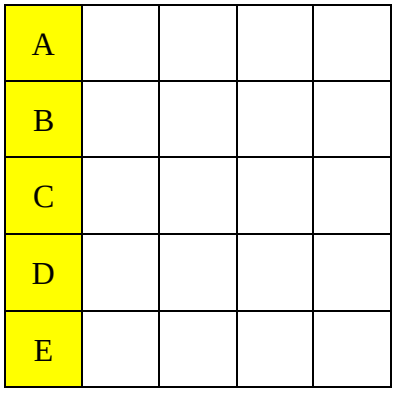
To encode the problem into the CNF form, first, we create two beside functions to deal with, at least one queen and at most one queen requirements. Both are shown below.

Function 1

Function 1 will produce a clauses of conjunction of all element in the *propositions* then add to the *cnf* which is the list of clauses.

```
Function 1: atLeastOneQueen
    input  :  propositions = A list of proposition letters.
              cnf  =  A list of clauses.
    output :  cnf  =  A list of clauses.
1   Append cnf  with propositions
2   return cnf
```

The *propositions* is the list of proposition letters that receive from *cnf function*. It cloud be the list of proposition letters in any row or any column. For example,

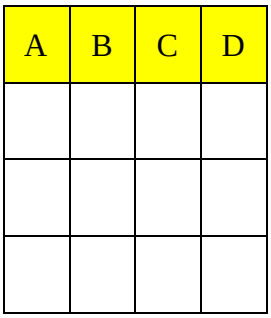
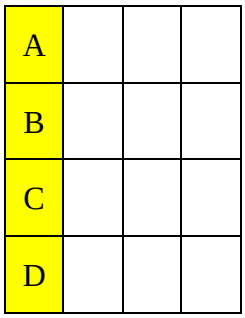
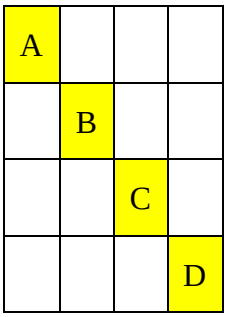
	Any row	Any column
Example Table		
propositions	$[A, B, C, D, E]$	
clauses appended to *cnf*	$[A, B, C, D, E]$	
Meaning	$A \vee B \vee C \vee D \vee E$	

Function 2

Function 2 will produce a clauses of each pair conjunction of negation in the propositions then add to the *cnf* which is the list of clauses.

```
Function 2: atMostOneQueen
      input  :  propositions = A list of proposition letters.
                cnf  =  A list of clauses.
      output :  cnf  =  A list of clauses.
1     for k := 0 to propositions length do
2        for j := 0 to k do
3           Create an empty list temp
4           Append temp with -propositions[j]
5           Append temp with -propositions[k]
6           Append cnf  with temp
7        end for
8     end for
9     return cnf
```

The *propositions* is the list of proposition letters that receive from *cnf function*. It cloud be the list of proposition letters in any row, any column or any diagonal. For example,

	Any row	Any column	Any diagonal
Example Table			
propositions	$[A, B, C, D, E]$		
clauses appended to *cnf*	$[-A, -B], [-A, -C], [-A, -D], [-B, -C], [-B, -D], [-C, -D]$		
Meaning	$(\neg A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (\neg A \vee \neg D) \wedge (\neg B \vee \neg C) \wedge (\neg B \vee \neg D) \wedge (\neg C \vee \neg D)$		

Function 3

Function 3 will create the *input.cnf* file and send it to Linux machine to run SAT Solver which is MiniSat version 1.14. Then download the result which is *output.txt* from the Linux machine to the local machine. And return the solution in the *output.txt*.

Function 3: cnf

input : *N* = The size of the problem
show_all = Boolean to show all solutions, if true the function will return all possible solutions, otherwise will return only one solution.
output : *solutions* = A list of solutions.

```
1   Create an empty 2D list board
2   Assign the initial value of the board with running number from 1 to N*N
3   Create an empty list to collect solutions. solutions
4   Create an empty list to collect clauses. cnf
5   for i := 0 to N do
6      Create an empty list to collect all proposition letters in the i^th row. propositions_row
7      Create an empty list to collect all proposition letters in the j^th column. propositions_col
8      for j := 0 to N do
9         Append propositions_row with board[i][j]
10        Append propositions_col with board[j][i]
11     end for
12     Call atLeastOneQueen(propositions_row, cnf)
13     Call atMostOneQueen(propositions_row, cnf)
14     Call atLeastOneQueen(propositions_col, cnf)
15     Call atMostOneQueen(propositions_col, cnf)
16  end for
17  for i := 0 and N-1 do
18     for j := 0 to N do
19        Create an empty list to collect all proposition letters in the first direction of each diagonal. propositions1
20        Create an empty list to collect all proposition letters in the second direction of each diagonal. propositions2
21        for i' := 0 to N do
22           for j' := 0 to N do
23              if i equal to N-1 and j equal to 0 then
24                 Do nothing
25              else if i equal to N-1 and j equal to N-1 then
26                 Do nothing
27              else then
28                 if i equal to i' and j equal to j' then
29                    Append propositions1 with board[i'][j']
30                    Append propositions2 with board[i'][j']
31                 else if i-j equal to i'-j' then
32                    Append propositions1 with board[i'][j']
33                 else if i+j equal to i'+j' then
34                    Append propositions2 with board[i'][j']
35                 end if
36              end if
37           end for
38        end for
39        Call atMostOneQueen(propositions1, cnf)
40        Call atMostOneQueen(propositions2, cnf)
41     end for
42  end for
43  Convert a list cnf to a input.cnf file
44  Send the input.cnf file to Linux Machine to run SAT Solver which is MiniSat version 1.14
45  Download output.txt which is the solution of the problem to the local machine
46  Append solutions with the solution from output.txt file
47  if show_all equal to True then
48     repeat
49        Append cnf with the negated form of the solution from output.txt file
50        Convert a list cnf to a input.cnf file
51        Send the input.cnf file to Linux Machine to run SAT Solver which is MiniSat version 1.14
52        Download output.txt which is the solution of the problem to the local machine
53        Append solutions with the solution from output.txt file
54     until the solution from output.txt file is unsatisfiable
55  end if
56  return solutions
```

There are 3 main sections in this function, **The first section**: line 5-16

This part will create the list of proposition letter for each row and each column and send it to Function 1 and Function 2 to make clauses and append it to *cnf* variable which is a list of clauses. This part will math with the formula below.

For at least one queen on every row, line: 5-12 and 16

$$\bigwedge_{i=1}^{N} \bigvee_{j=1}^{N} P_{ij}$$

For at most one queen on every row, line: 5-11, 13 and 16

$$\bigwedge_{i=1}^{N} \bigwedge_{0<j<k\leq N} (\neg P_{ij} \vee \neg P_{ik})$$

For at least one queen on every column, line: 5-11, 14 and 16

$$\bigwedge_{j=1}^{N} \bigvee_{i=1}^{N} P_{ij}$$

For at most one queen on every column, line: 5-11 and 15-16

$$\bigwedge_{j=1}^{N} \bigwedge_{0<i<k\leq N} (\neg P_{ij} \vee \neg P_{kj})$$

The second section: line 17-42

This part will create the list of proposition letter for each diagonal which have 2 directions and send it to Function 1 and Function 2. This part will math with the formula below.

For at most one queen on every diagonal,

$$\bigwedge_{0<i<i'\leq N} \left(\bigwedge_{j,j':i+j=i'+j' \vee i-j=i'-j'} \neg P_{ij} \vee \neg P_{i'j'} \right)$$

The last section: line 43-56

Line 43-46: This part will convert *cnf* to the *input.cnf* file and send it to Linux machine to run SAT Solver which is MiniSat_version 1.14. Then download the result which is *output.txt* from the Linux machine to the local machine. Append *solutions* with the solution from *output.txt* file

Line 47-55: If the *show_all* condition is *True*, this program will find all the possible solutions by the first step, append *cnf* with the negated form of the solution from *output.txt* file, then do the same as *line 43-46* did and repeated all step until the solution from the *output.txt* file is unsatisfiable. More info to getting more solutions. But if the *show_all* condition is *False* it will go to *line 56*.

Line 56: Finally, return the *solutions*.

Install Python to Local machine

1. Go to this link.
2. Download Python 3.7.3 for your Operating System (e.g., Mac OS X, Windows)
3. Install Python 3.7.3 by following this link.

Install SAT Solver to Linux machine

1. Connect to TTUnet VPN by following this link.
2. Connect to the Linux server which name is *willow.cs.ttu.edu* by following this link. Note: *willow.cs.ttu.edu* is not a link. It is a machine name.
3. Download and install SAT Solver which is MiniSat version 1.14 to this Linux machine, type:

   $ wget http://minisat.se/downloads/MiniSat_v1.14_linux
   $ chmod u+x MiniSat_v1.14_linux

More about MiniSat version 1.14

- Carefully read this link for instructions on how to invoke MiniSat, how to save a solution, and how to get all solutions. You will not need to use any of MiniSat's optional flags.
- To run the MiniSat
  - Require input file in a simplified "DIMACS CNF" format, follow steps below to create the simple one to test, More info.
    - To create input.cnf file, type:
      $ nano input.cnf
    - Paste the text below to the terminal,
      p cnf 5 4
      1 -5 4 0
      -1 5 3 4 0
      -3 -4 0
      -1 -2 3 -4 -5 0
    - To save the file, press *Ctrl+o*.
    - To exit nano press *Ctrl+x*.
  - To get the solution, type:
    $ ./MiniSat_v1.14_linux input.cnf output.txt
  - To read the solution, type:
    $ cat output.txt

    The output.txt should show as below,
    SAT
    1 -2 -3 4 5 0

Download project to Local machine

First method (clone repository)

1. Open Command Prompt or Terminal. More information.
2. Change the current working directory to the location where you want to download the project.
3. Clone repository:

    $ git clone https://github.com/rinriko/N-Queens.git

Note: If Git is not installed, please complete the following instructions. Or follow the second method instructions.

Second method (download project)

1. Go to https://github.com/rinriko/N-Queens.
2. Above the list of files, click ↓ Code.
3. Click "Download ZIP".
4. Save the zip file into a convenient location on your PC and start working on it.
5. Extract the ZIP file.

Set up project

From the first method

1. Continue use Command Prompt or Terminal, type:

    $ cd N-Queens
2. In order to install the project requirements, type:

    $ make setup

From the second method

1. Open Command Prompt or Terminal. More info.
2. Change the current working directory to the location where you have the cloned directory. More info.
3. In order to install the project requirements, type:

    $ make setup

Run project

Inside the N-Queens directory, use the Command Prompt or Terminal and type:

$ make run

Help

Inside the N-Queens directory, use the Command Prompt or Terminal and type:

$ make help

How to use this program

1. Connect to TTUnet VPN by following this link.
2. Enter your eraider username.
3. Enter your eraider password.
4. Select your options.
