Term Project: N-queens

Course: CP468

Group #: 15

Due date: December 1, 2019

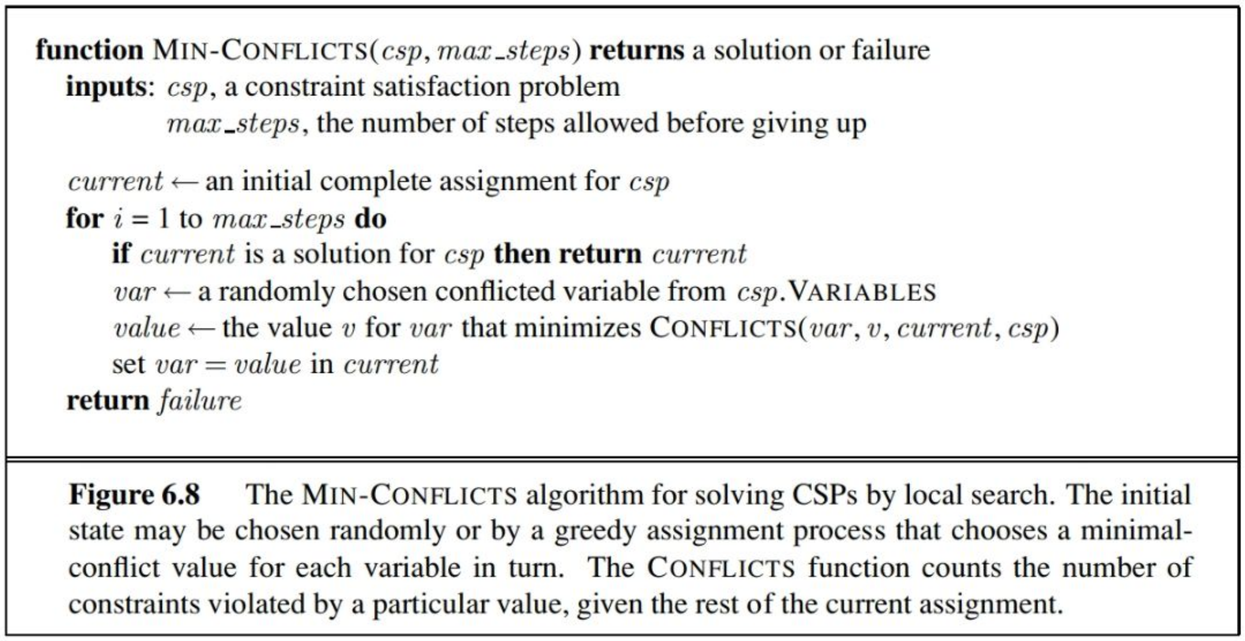
Group members:
Amardeep Sarang - 160112080
Saje Bailey - 160963460
Ryley Davenport - 150470750

Introduction

The n-queens problem is a problem where you try to place N queens on a NxN chess board such that no queen can attack another queen. In this project we will attempt to solve the n-queens problem for large values of N using the min-conflict local search algorithm.

Pseudocode

The following pseudocode was sourced from the course textbook (Artificial Intelligence: A Modern Approach 3rd Edition) and it formed the basis for or min-conflict algorithm.



```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var = value in current
  return failure
```

Figure 6.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

Performance and design choices

For our implementation of the min-conflict algorithm we decided to represent the n-queens as a 1d array. This was done since each queen already had a predetermined column and the only attribute of a queen that would change from one state to another was which row it would be placed in. Therefore, our main data structure was a Java array where the ith element was the queen in the ith column and the value of the ith column was which row the ith queen was placed in. We also chose this data structure because it was fairly lightweight in terms of memory and it was easy to implement the min-conflict algorithm around it.

The main section of the algorithm called for us to choose a random queen and count the number of conflicts/attacks there would be at each square in the queen's column. Counting the conflicts at a square took a complexity of O(n) and this had to be done n times for all the squares in the chosen queen's column, which meant this section had a complexity of O(n^2). The book states that the process of choosing a queen and placing it would only have to be done an average of 50 times however in our testing we found this to be much higher and pushed the overall complexity up closer to O(n^3). Seeing this we decided to switch our code from Python to Java due to Java's inherent speed advantage over Python due to it being a compiled language. We hoped that this would help us overcome the algorithm's complexity and allow us to run the problem with larger value of n.

This was indeed the case, Python took 10 seconds to solve 100-queens and took over 30min to solve 1000-queens whereas Java can solve 100-queens almost instantly and 1000-queens in a few seconds. Despite this improvement 10000-queens problem was not able to be solved even when testing it for 1.5 hour. We felt that this maybe due to the search being stuck in a local minimum. To combat this we added a forbidden list which stopped the algorithm from choosing the queens which had been chosen recently. The queen would remain in this list until a certain number of iterations had passed or until the max length of the list was reach. At which point the queens would exit the list in a FIFO fashion. However, despite these measures our program is still not able to solve the 10000-queens problem.

Code

N-queens solver (min-conflict algorithm)

This java program code solves the n-queens problem for a user specified n. It uses the min-conflicts algorithm.

```
/**
 * This code solves the n-queens problem for a user specified n. It uses the min-conflicts algorithm.
 * type javac NQueen.java to compile
 * type java Nqueen [N] to run
 */

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.io.PrintWriter;
import java.io.File;

public class NQueen {
    public static int[] genState(int N) {
        Random rd = new Random();
        int[] state = new int[N];
        for (int i = 0; i < state.length; i++) {
            state[i] = rd.nextInt(N);
        }
        return state;
    }

    public static boolean allSafe(int[] state) {
        // checks if all queens are safe in given state

        // check all queens for conflict
        for (int q = 0; q < state.length; q++) {
            for (int i = 0; i < state.length; i++) {
                if (q != i) {
                    // check same row conflict
                    if (state[i] == state[q]) {
                        return false;
                    }

                    // check diagonal conflict
                    if (state[i] == state[q] + (i - q)) {
                        return false;
                    }

                    if (state[i] == state[q] - (i - q)) {
                        return false;
                    }
                }
            }
        }
        return true;
    }

    public static int countConflict(int q, int[] state) {
        // counts the number of conflicts that occurs at a given state between
a given
        // queen and all other queens

        int conflict = 0;
        for (int i = 0; i < state.length; i++) {
            if (q != i) {
                // check same row conflict
                if (state[i] == state[q]) {
                    conflict++;
                }

                // check diagonal conflict
                if (state[i] == state[q] + (i - q)) {
                    conflict++;
                }

                if (state[i] == state[q] - (i - q)) {
                    conflict++;
                }
            }
        }

        return conflict;
    }

    public static boolean minConflict(int[] currState, int N, int maxSteps) {
        /*
         * min conflict algorithm implementation input: current_state - a list of the
         * which row each queen is sitting in example: current_state[i]= n the queen in
         * ith column is in the nth row N - n in n-queen
         */
        boolean successful = true;
        Random rd = new Random();
        int forbidCount = 20;// max number of steps that a var should be in the forbidden queue forbidden
        int forbidSize = (int) (N * 0.1);// max size of forbidden queue
        List<Integer> forbiden = new ArrayList<Integer>();
        while (allSafe(currState) == false && maxSteps > 0) {
            int var = rd.nextInt(N);
            int minCon = N + 1;

            // System.out.println(forbidSize);
            List<Integer> minVal = new ArrayList<Integer>();
            if (forbiden.contains(var) == false) {// only select var if not in forbidden queue

                int newVal = currState[var];

                // change var to value that minimizes conflict
                for (int val = 0; val < N; val++) {
                    currState[var] = val;

                    // count how many conflicts this new state will produce
                    int newCon = countConflict(var, currState);

                    if (newCon < minCon) {
                        minVal.clear();
                        newVal = val;
                        minVal.add(val);
                        minCon = newCon;
                    } else if (newCon == minCon) {
                        minVal.add(val);// add all position that tie for min
conflict
                    }
                }

                currState[var] = minVal.get(rd.nextInt(minVal.size()));
                maxSteps--;

                // add var to forbidden queue
                if (minCon == 0) {
                    forbiden.add(var);
                }
            }
            if (forbiden.size() > forbidSize || forbidCount < 1) {
                // release a var from forbidden queue if size if too large or
time expired
                forbiden.remove(0);
                forbidCount = 10;
            }
            forbidCount--;

        }

        if (maxSteps < 1) {
            successful = false;
        }

        return successful;
    }

    public static void printBoard(int[] board) {
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board.length; j++) {
                if (board[j] == i) {
                    System.out.print("Q ");
                } else {
                    System.out.print("_ ");
                }

            }
            System.out.println(" ");
        }
    }

    public static void main(String[] args) {

        int N = 10;

        if (args.length > 0) {
            N = Integer.parseInt(args[0]);
```

```java
                System.out.print("Running n-queens with n = ");
                System.out.println(N);
            } else {
                System.out.println("No n entered using n=10.");
            }

            int[] state;

            boolean successful = false;
            state = genState(N);
            // retry min conflicts with new init state until succes
            while (successful == false) {
                state = genState(N);
                successful = minConflict(state, N, 10000);

                if (successful) {
                    System.out.println("Successful");
                } else {
                    System.out.println("Failed, re-initializing");
                }
            }

            // don't bother printing large board
            if (N < 52) {

                printBoard(state);
            }


            for (int i = 0; i < state.length; i++) {

                String out = "The queen in column " + Integer.toString(i+1)+" is
placed in row "+ Integer.toString(state[i] + 1);
                System.out.println(out);

            }


        }
    }
```

Solution drawer

This python program was used to draw the solution for an n-queens problem, using the output from the min-conflict N-queens solver. It will draw a NxN checkerboard and color the squares with queens on them red.

```
'''''
Requires numpy and openCV to run, use pip to install
To run type python show_queens.py [N] [path to input file]
'''
import cv2
import numpy as np
import sys

f=open(sys.argv[2])
n=int(sys.argv[1])
scale = 700
if len(sys.argv) >3:
    scale= int(sys.argv[3])

img = np.zeros([n,n,3])
black=(0,0,0)
white=(255,255,255)
red=(0,0,255)
col=0
#create checker board
for i in range(n):

    for j in range(n):
        #print(col%2==0)
        if col%2==0:
            img[i][j]=white
        else:
            img[i][j]=black
        col+=1
    col+=1
#color queen cord red
for line in f:
    l=line.strip().split()
    x=int(l[4])-1
    y=int(l[9])-1
    img[x][y]=red

img=rescaled = cv2.resize(img, (scale, scale), interpolation=cv2.INTER_AREA)

cv2.imwrite(str(n)+".jpg",img)
cv2.imshow(str(n), img)
cv2.waitKey(0)
```

Testing

The following is the output from the min-conflict solver for the 10-queens problem. In addition to this output for 100-queens, 50-queens, and 1000-queens is also provided see “100_queens_test.txt”, “1000_queens_test.txt” and “50_queens_test.txt” in project folder.

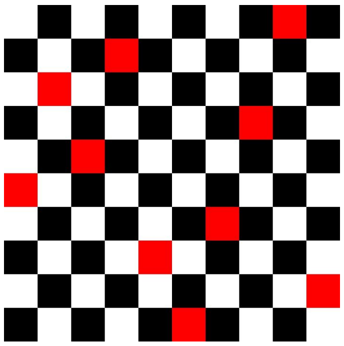
Test results 10-queens:

```
Running n-queens with n = 10
Successful
_ _ _ _ _ Q _ _ _ _
_ _ Q _ _ _ _ _ _ _
_ _ _ _ Q _ _ _ _ _
_ Q _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ Q _ _
_ _ _ _ _ _ _ _ _ Q
_ _ _ _ _ _ Q _ _ _
_ _ _ Q _ _ _ _ _ _
Q _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ Q _
The queen in column 1 is placed in row 9
The queen in column 2 is placed in row 4
The queen in column 3 is placed in row 2
The queen in column 4 is placed in row 8
The queen in column 5 is placed in row 3
The queen in column 6 is placed in row 1
The queen in column 7 is placed in row 7
The queen in column 8 is placed in row 5
The queen in column 9 is placed in row 10
The queen in column 10 is placed in row 6
```

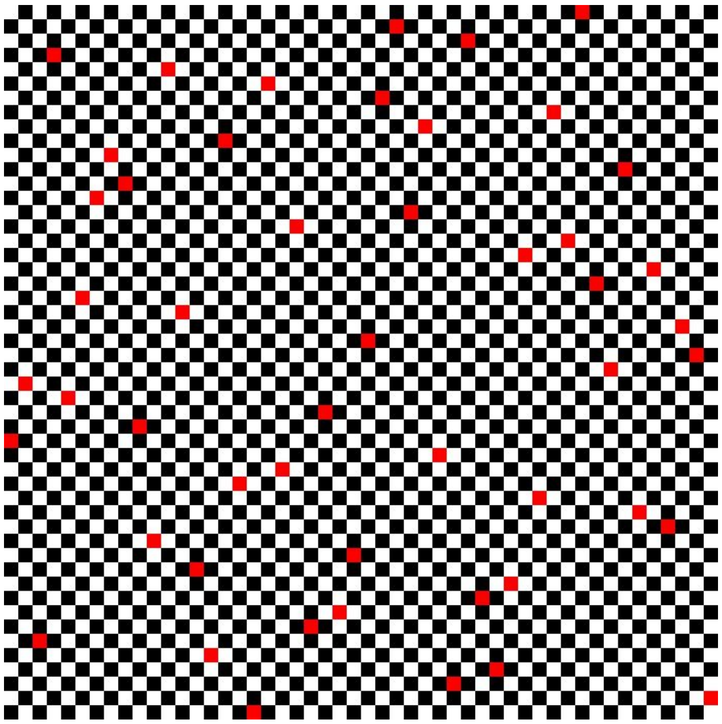
Graphic solution output

The following are solutions for the n-queens problem that have been drawn.

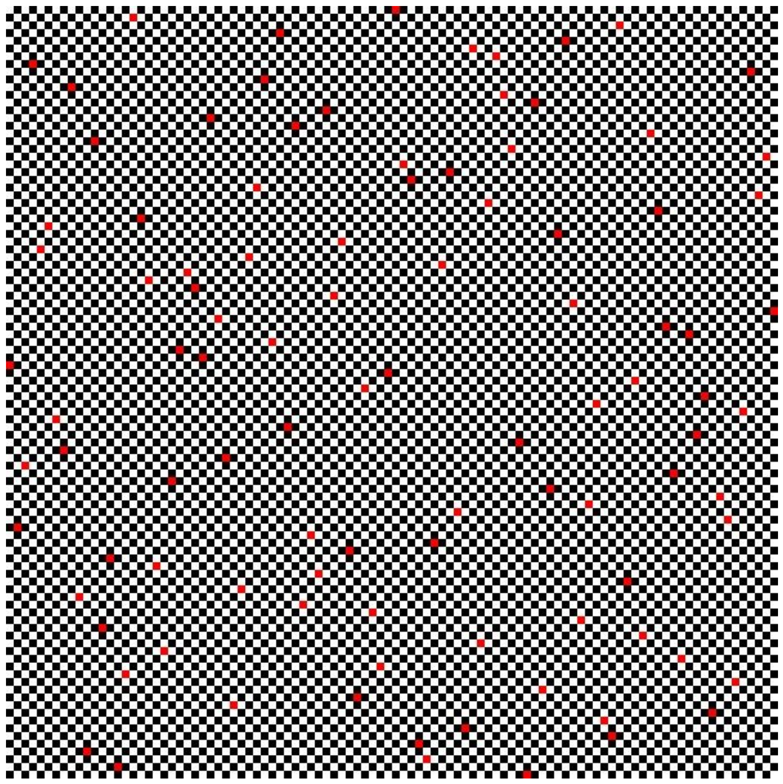
Sample solution for 10-queen:



Sample solution for 50-queen:



Sample solution for 100-queen:



Instructions

To run all programs use a console, you must navigate to the directory where the program is.

Program: NQueen.java

To compile: type javac ‘NQueen.java’

To run: type ‘java Nqueen [N]’

Where:

N – number of queens and size of chess board

Program: show_queen.py

Libraries: Requires numpy and openCV to run, use pip to install

To run: type ‘python show_queens.py [N] [path to input file]’

where:

N – number of queens and size of chess board

path to input file – the path to the text file containing the copied output line from NQueen.java

Each line must have the following format and N must match the number of lines in file:

```
The queen in column 1 is placed in row 9

The queen in column 2 is placed in row 4

The queen in column 3 is placed in row 2

…

The queen in column 10 is placed in row 6
```