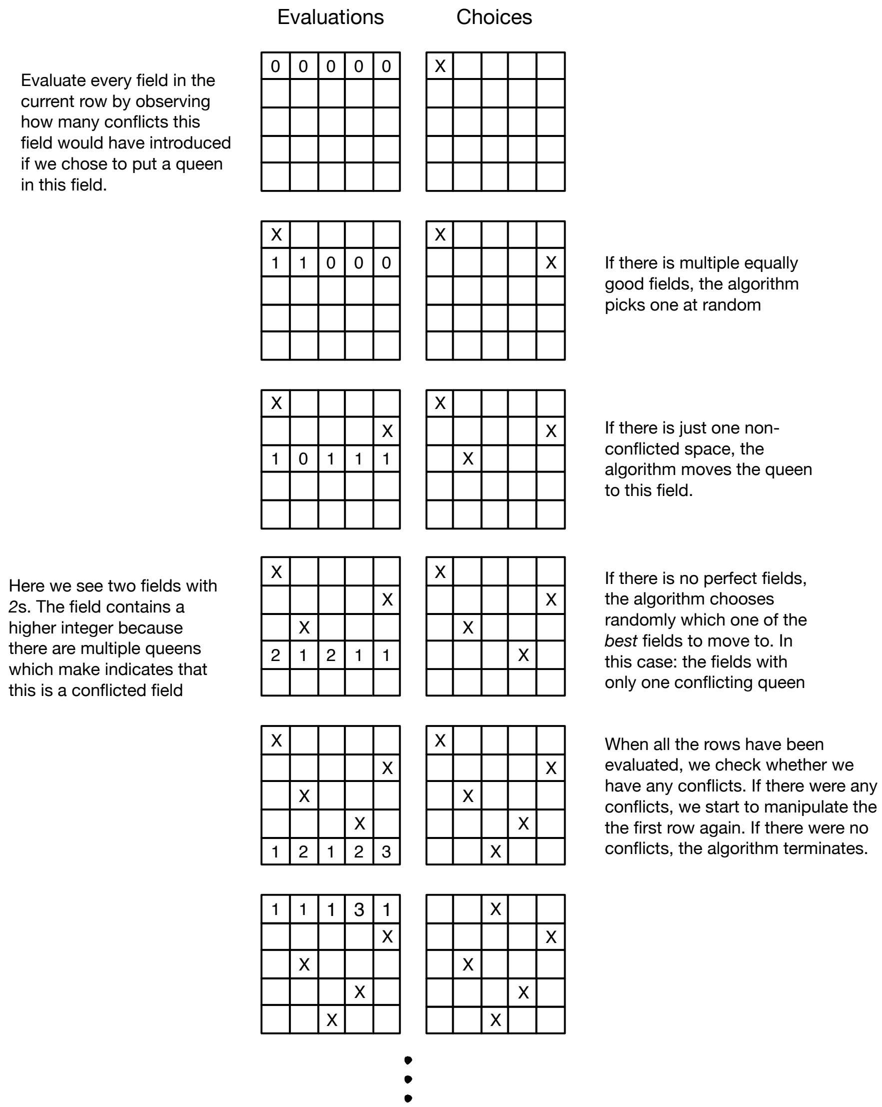
TDT4136 Logic and Reasoning Systems

Jørgen Grimnes
Assignment 5

Fall 2013

1 How the algorithm works



2 The Python implementation

The MIN CONFLICT algorithm

```
def min_conflict( row,queens,k_queens ):
    """
    This method is an implementation of the CSP min_conflict strategy.
    It checks every column against every queen, thus achieving a time
    complexity of k^2 (k-queens) which is reasonably low.
    """
    encountered_min = ( ) # fewest possible conflict
    minimum_conflicts = [] # the columns which would yield the fewest

    for column in xrange(k_queens):
        """
        Loop over each column/field in a single row
        """
        out = 0
        for queen in queens:
            """
            Evaluate how many conflicts this current field would
            introduce in comparison to the other queens.
            """
            if queen.column != -1  and queen.row!=row:
                if queen.column == column:
                    out += 1
                elif (queen.row - row) == (queen.column - column):
                    out += 1
                elif (queen.row - row) == -(queen.column - column):
                    out += 1

        """
        Keep track of the best results
        """
        if out<encountered_min:
            encountered_min = out
            minimum_conflicts = [column]
        elif out==encountered_min:
            minimum_conflicts.append( column )

    return random.choice( minimum_conflicts ), encountered_min
#end min_conflict
```

The algorithm will loop through the different fields (columns) in the given row, while it calculates how many conflicts would have been introduced if we placed the current queen in this column. The method always keeps track of the best solutions.

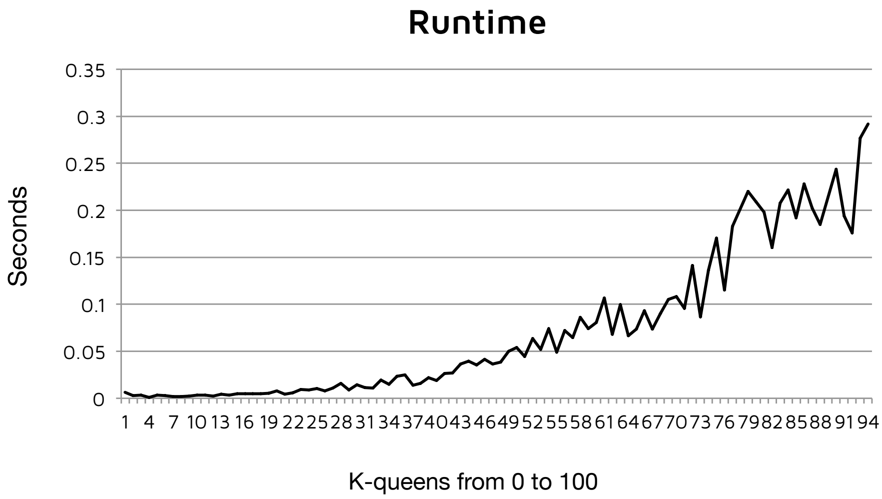
The idea is to check the (row, column) coordinate of every queen and see if they are aligned with currently evaluating (row, column) either vertically or diagonally. Since there are only a few arithmetic operations, this runs fast. See the following page for the complete code.

This script is made to be run by the [PYPY interpreter]

```python
try: import numpypy as np
except ImportError: import numpy as np
import random

class Queen:
    def __init__(self, row_index):
        self.row = row_index
        self.column = -1 # not yet on the board
#end class

def min_conflict( row,queens,k_queens ):
    """
    This method is an implementation of the CSP min_conflict strategy.
    It checks every column against every queen, thus achieving a time
    complexity of k^2 (k-queens) which is reasonably low.
    """
    encountered_min = ( ) # fewest possible conflict
    minimum_conflicts = [] # the columns which would yield the fewest

    for column in xrange(k_queens):
        """
        Loop over each column/field in a single row
        """
        out = 0
        for queen in queens:
            """
            Evaluate how many conflicts this current field would
            introduce in comparison to the other queens.
            """
            if queen.column != -1  and queen.row!=row:
                if queen.column == column:
                    out += 1
                elif (queen.row - row) == (queen.column - column):
                    out += 1
                elif (queen.row - row) == -(queen.column - column):
                    out += 1

        """
        Keep track of the best results
        """
        if out<encountered_min:
            encountered_min = out
            minimum_conflicts = [column]
        elif out==encountered_min:
            minimum_conflicts.append( column )

    return random.choice( minimum_conflicts ), encountered_min
#end min_conflict

def local_search( k_queens ):
    queens = [  Queen(i) # The i represents its row, which is constant.
                for i in xrange(k_queens)]  # Our game pieces
    system_conflicts = None                 # Number of conflicts

    while system_conflicts!=0:
        # Keep running until there are no conflicts.
        system_conflicts = 0

        for queen in queens:
            """
            Loop over the queens and relocate them to a column which causes fewer
            conflicts, then update the total number of conflicts on our chess board.
            """
            queen.column, n_conflicts = min_conflict( queen.row, queens, k_queens )
            system_conflicts = system_conflicts + n_conflicts
    #endwhile

    return queens
# end local search
```

```
k = 100  # bord size
results = local_search( k ) # Returns k queen objects

display_matrix = np.zeros(   shape=( k,k ),  # used to display the result
                             dtype=np.uint8 )
for queen in results: display_matrix[queen.row,queen.column] = 1


"""
Perform a numpy compatible (non-truncated) print-out of the results.
"""
print '\n'.join([' '.join(map(str,lines)) for lines in display_matrix.tolist()])
```

3 Results



This curve is somewhat better than the n^2 time complexity I expected. The algorithm has proven itself usable up to the 2000-queens problem. You may find a solution for the 8, 16 and 35-queens problem on the following pages.

8 Queens

```
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
```

16 Queens

```
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
```

35 Queens

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

4 Addendum

I had a hard time understanding how I should have copy-pasted some of the domain changes here, since the algorithm only uses a “meta understanding” of the queens domain. I have chosen this implementation so that the algorithm can escape “awful” situations where the every element in the domain is conflicted.