Rene Ismael Vazquez

Equally Attacking Knights Problem

Guiding Question and Variables:

Let's consider a square chessboard with the dimensions n x n.
What is the maximum number of knights that can be placed so that each knight is attacking exactly m knights?
Where should the knights be placed?
Another way to phrase m is to be, the number of knights that each other knight is attacked by.

We can label each chess board position as a coordinate pair (i,j) such that $1 \le i, j \le n$.
Here is an example of a labeling where n = 4:

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

x_{ij} will be a binary variable, meaning $x_{ij} \in \{0, 1\}$
1 will correspond to a knight placed on (i,j), and 0 will correspond to no piece there.

Objective Function

Since we want to maximize the number of knights placed on the board, we get an objective function of this:

$$\text{Maximize: } \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

Constraints

The constraint states that every knight needs to be attacking exactly m other knights.
This will be accomplished with a set of inequalities for each board position (i,j).
For any (i,j) we will need to find all the possible positions that can reach that position (i,j).
Since the knight moves down 2 and over 1 in any direction, there are 8 possible knight moves.
We also have to ensure we only work with knights from legal spaces (on the board).
This is accomplished by summing all $x_{i \pm 1,\ 2;\ j \pm 1,\ 2}$ such that $1 \le i \pm 1,\ 2;\ j \pm 1,\ 2 \le n$

Additionally, if a knight moves 1 in the first direction, it must move 2 in the second.
Thus, if i increases by 1, j can only be adjusted by ± 2
If j decreases by 2, i can only be adjusted by ± 1
(Assuming the new squares are legal board positions)
If a knight moves 1 in one direction, it must move 2 in the other, and vice versa:
$\Delta i + \Delta j = 3$
Where $i = \pm 1, \pm 2$ *and* $j = \pm 1, \pm 2$

This makes up the LHS of our constraint.

Each LHS will be repeated twice, being upper and lower bounded by 2 inequalities.
The RHS will describe the upper and lower bounds such that the LHS equates to m. (if $x_{ij} = 1$)
It also needs to be considered that if x_{ij} is 0, meaning there is no knight on that position, that the square doesn't need to be attacked from any of the possible attacker positions.
Thus, every constraint is going to depend on x_{ij} to determine if a knight is even there to consider.
Each LHS will be lower bounded by $m*x_{ij}$, meaning that

- If $x_{ij}=1$, meaning a knight is there to consider, the LHS $\geq$ m
- If $x_{ij}=0$, meaning there is no knight to consider, the LHS ≥ 0

Each LHS will be upper bounded by $(m-8)*x_{ij}+8$

- If $x_{ij}=1$, meaning a knight is there to consider, the LHS $\leq$ m
- If $x_{ij}=0$, meaning there is no knight to consider, the LHS ≤ 8
    - We choose 8, because that is the upper limit for possible attacking knights.

Thus, if $x_{ij}=1$, the LHS will be upper and lower bounded by m, forcing the LHS = m.
This covers the cases where a knight is and isn't at position x_{ij}
If $x_{ij} = 0$, the LHS will be upper bounded by 8, because there are 8 possible attacking squares, and lower bounded by 0, because no squares need to be attacking (i,j) (as there's no knight there)

These constraints can be formulated as such:

$$\sum_{1}^{8} x_{i \pm 1,\ 2;\ j \pm 1,\ 2} \leq (m-8)x_{ij} + 8$$
$$\text{such } i = \pm 1, 2 \text{ and } j = \pm 1, 2,$$
$$1 \leq i \pm 1,\ 2;\ j \pm 1,\ 2 \leq n\ ,$$
$$\text{and } \Delta i + \Delta j = 3$$
$$\text{for all } 1 \leq i, j \leq n$$

$$\sum_{1}^{8} x_{i \pm 1,\ 2;\ j \pm 1,\ 2} \geq m x_{ij}$$

$$\text{such } i = \ \pm 1, 2 \text{ and } j = \ \pm 1, 2,$$

$$1 \leq i \pm 1,\ 2;\ j \pm 1,\ 2 \leq n\ ,$$

$$\text{and } \Delta i + \Delta j = 3$$

$$\text{for all } 1 \leq i, j \leq n$$

Code

Now that the LP has been formulated, the LP must actually be written.
Python was used to write a script that creates a text file.
The contents of this text file are a LP in a format that could be read by lpsolve.
Lpsolve is the name of the software that was used to compute the solutions.
Included below is the python code:

At the top of the code, the m and n can be changed easily.

```
#chessboard size, size x size
boardSize = 5
# desired number of knights each piece is attacked by
attackingKnights = 2

fh = open("hw2_lp.txt", 'w')
```

The objective function is written here. Using n it determines how many board positions there are.
It then creates the string that will be read by ipsolve.
The string is "max: " and then the variable names of each square, with "+" signs in between.

```
# writing the objective function
#
# need to maximize the number of knights
objectiveString = 'max: '
for i in range(1, boardSize + 1):
   for j in range(1, boardSize + 1):
       # if not the last board position to check
       if (i + j) != (boardSize * 2):
           objectiveString += 'x_' + str(i) + '_' + str(j) + " + "
# printing the final board coordinate
objectiveString += 'x' + '_' + str(boardSize) + '_' + str(boardSize) + ';\n'
fh.write(objectiveString)
```

Here, the following process is repeated at every board position:
All 8 possible attacking squares are checked to see if they are legal board positions.
If they are, they're added to a new constraint line as the LHS.
The RHS is then added, determined using the m value designated at the top of the code.

```
# writing the constraints
#
# each piece can only attack (attackingKnights) number of pieces
possibleMoves = [[1, 2],[1, -2],[-1, 2],[-1, -2],[2, 1],[2, -1],[-2, 1],[-2, -1]]
# looping through every board position
for i in range(1, boardSize + 1):
   for j in range(1, boardSize + 1):
```

```
        # constraint string for every board position
        knightAttackersString = ''
        # finding legal positions
        for x in possibleMoves:
            # is legal space, and not the last possible move to check
            if (i + x[0] >= 1) and (j + x[1] >= 1) and (i + x[0] <= boardSize) and (j + x[1] <= boardSize):
                knightAttackersString += '+' + 'x_' + str(i + x[0]) + '_' + str(j + x[1])
        # if there exist legal attacker spaces
        if knightAttackersString != '':
            # copying the LHS to handle the differing RHSs
            knightAttackersString2 = knightAttackersString
            knightAttackersString2 += ' <= ' + str(attackingKnights - 8) + '*' + 'x_' + str(i) + '_' + str(j) + '
+8'+ ';\n'
            knightAttackersString += ' >= ' + str(attackingKnights) + '*' + 'x_' + str(i) + '_' + str(j) + ';\n'
        fh.write(knightAttackersString2)
        fh.write(knightAttackersString)
```

Here, the variables are declared.

Similar to the objective function, it lists all possible board position variables, based on the size of the board, using the previously assigned n value at the top of the program.

```
# variable declarations
#
# One for each square of the board
# Binary, 0 for no piece on the square, 1 for piece on the square
var_string = 'bin '
for i in range (1, boardSize + 1):
    for j in range (1, boardSize + 1):
        # if on final board coordinate, don't print
        if (i + j) != (boardSize * 2):
            var_string += 'x_' + str(i) + '_' + str(j) + ', '
# printing the final board coordinate
var_string += 'x_' + str(boardSize) + '_' + str(boardSize) + ';'
fh.write(var_string)

fh.close()
```

Sample Input and Output

This included code, using n = 5 and m = 2, would produce the following text file:

```
max: x_1_1 + x_1_2 + x_1_3 + x_1_4 + x_1_5 + x_2_1 + x_2_2 + x_2_3 + x_2_4 + x_2_5 + x_3_1 + x_3_2 + x_3_3 +
x_3_4 + x_3_5 + x_4_1 + x_4_2 + x_4_3 + x_4_4 + x_4_5 + x_5_1 + x_5_2 + x_5_3 + x_5_4 + x_5_5;
+x_2_3+x_3_2 <= -6*x_1_1 +8;
+x_2_3+x_3_2 >= 2*x_1_1;
+x_2_4+x_3_3+x_3_1 <= -6*x_1_2 +8;
+x_2_4+x_3_3+x_3_1 >= 2*x_1_2;
+x_2_5+x_2_1+x_3_4+x_3_2 <= -6*x_1_3 +8;
+x_2_5+x_2_1+x_3_4+x_3_2 >= 2*x_1_3;
+x_2_2+x_3_5+x_3_3 <= -6*x_1_4 +8;
+x_2_2+x_3_5+x_3_3 >= 2*x_1_4;
+x_2_3+x_3_4 <= -6*x_1_5 +8;
+x_2_3+x_3_4 >= 2*x_1_5;
+x_3_3+x_1_3+x_4_2 <= -6*x_2_1 +8;
+x_3_3+x_1_3+x_4_2 >= 2*x_2_1;
+x_3_4+x_1_4+x_4_3+x_4_1 <= -6*x_2_2 +8;
+x_3_4+x_1_4+x_4_3+x_4_1 >= 2*x_2_2;
+x_3_5+x_3_1+x_1_5+x_1_1+x_4_4+x_4_2 <= -6*x_2_3 +8;
+x_3_5+x_3_1+x_1_5+x_1_1+x_4_4+x_4_2 >= 2*x_2_3;
+x_3_2+x_1_2+x_4_5+x_4_3 <= -6*x_2_4 +8;
+x_3_2+x_1_2+x_4_5+x_4_3 >= 2*x_2_4;
+x_3_3+x_1_3+x_4_4 <= -6*x_2_5 +8;
+x_3_3+x_1_3+x_4_4 >= 2*x_2_5;
+x_4_3+x_2_3+x_5_2+x_1_2 <= -6*x_3_1 +8;
+x_4_3+x_2_3+x_5_2+x_1_2 >= 2*x_3_1;
+x_4_4+x_2_4+x_5_3+x_5_1+x_1_3+x_1_1 <= -6*x_3_2 +8;
+x_4_4+x_2_4+x_5_3+x_5_1+x_1_3+x_1_1 >= 2*x_3_2;
```

```
+x_4_5+x_4_1+x_2_5+x_2_1+x_5_4+x_5_2+x_1_4+x_1_2 <= -6*x_3_3 +8;
+x_4_5+x_4_1+x_2_5+x_2_1+x_5_4+x_5_2+x_1_4+x_1_2 >= 2*x_3_3;
+x_4_2+x_2_2+x_5_5+x_5_3+x_1_5+x_1_3 <= -6*x_3_4 +8;
+x_4_2+x_2_2+x_5_5+x_5_3+x_1_5+x_1_3 >= 2*x_3_4;
+x_4_3+x_2_3+x_5_4+x_1_4 <= -6*x_3_5 +8;
+x_4_3+x_2_3+x_5_4+x_1_4 >= 2*x_3_5;
+x_5_3+x_3_3+x_2_2 <= -6*x_4_1 +8;
+x_5_3+x_3_3+x_2_2 >= 2*x_4_1;
+x_5_4+x_3_4+x_2_3+x_2_1 <= -6*x_4_2 +8;
+x_5_4+x_3_4+x_2_3+x_2_1 >= 2*x_4_2;
+x_5_5+x_5_1+x_3_5+x_3_1+x_2_4+x_2_2 <= -6*x_4_3 +8;
+x_5_5+x_5_1+x_3_5+x_3_1+x_2_4+x_2_2 >= 2*x_4_3;
+x_5_2+x_3_2+x_2_5+x_2_3 <= -6*x_4_4 +8;
+x_5_2+x_3_2+x_2_5+x_2_3 >= 2*x_4_4;
+x_5_3+x_3_3+x_2_4 <= -6*x_4_5 +8;
+x_5_3+x_3_3+x_2_4 >= 2*x_4_5;
+x_4_3+x_3_2 <= -6*x_5_1 +8;
+x_4_3+x_3_2 >= 2*x_5_1;
+x_4_4+x_3_3+x_3_1 <= -6*x_5_2 +8;
+x_4_4+x_3_3+x_3_1 >= 2*x_5_2;
+x_4_5+x_4_1+x_3_4+x_3_2 <= -6*x_5_3 +8;
+x_4_5+x_4_1+x_3_4+x_3_2 >= 2*x_5_3;
+x_4_2+x_3_5+x_3_3 <= -6*x_5_4 +8;
+x_4_2+x_3_5+x_3_3 >= 2*x_5_4;
+x_4_3+x_3_4 <= -6*x_5_5 +8;
+x_4_3+x_3_4 >= 2*x_5_5;
bin x_1_1, x_1_2, x_1_3, x_1_4, x_1_5, x_2_1, x_2_2, x_2_3, x_2_4, x_2_5, x_3_1, x_3_2, x_3_3, x_3_4, x_3_5,
x_4_1, x_4_2, x_4_3, x_4_4, x_4_5, x_5_1, x_5_2, x_5_3, x_5_4, x_5_5;
```

This could then be copied into lpsolve, and be solved.
This is the following lpsolve output.

Actual values of the variables:

x_1_2	1
x_1_3	1
x_1_4	1
x_2_1	1
x_2_2	1
x_2_4	1
x_2_5	1
x_3_1	1
x_3_5	1
x_4_1	1
x_4_2	1
x_4_4	1
x_4_5	1
x_5_2	1
x_5_3	1
x_5_4	1

Optimal solution 16 after 2039 iter

This output lists all the variables that are equal to 1, meaning all the board positions where a knight should be.
If a variable is not listed, its value is 0, meaning there is no knight at any position not listed.

It states the optimal value in this case is 16, meaning the most knights that can be placed with the given constraints is 16.

Here is a visual description of the solution.
It is visually and manually verifiable that each knight is attacking exactly m other knights.
The dark squares represent where a knight should be, where the variable is equal to 1.
The light squares mean the variable written within are equal to 0; no knight placed there.

x_1_1	x_1_2	x_1_3	x_1_4	x_1_5
x_2_1	x_2_2	x_2_3	x_2_4	x_2_5
x_3_1	x_3_2	x_3_3	x_3_4	x_3_5
x_4_1	x_4_2	x_4_3	x_4_4	x_4_5
x_5_1	x_5_2	x_5_3	x_5_4	x_5_5

Solutions for Various n and m Values

Here is a chart with computed solutions for corresponding n and m value pairs.
The optimal solution describes the maximum number of knights that can be placed on the board, (for the given constraints)
The optimal positions describe which x_{ij} values are equal to 1, meaning the optimal solution involves a knight at board position x_{ij}

Any x_{ij} value not included in the chart is equal to 0, meaning there is no knight there. The computation time is also included for each test.

<table>
<tr><th>n</th><th>m</th><th>Optimal Solution</th><th>Optimal Positions</th><th>Computation Time</th></tr>
<tr><td>4</td><td>1</td><td>8</td><td><table><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>0.038</td></tr>
<tr><td>4</td><td>2</td><td>10</td><td><table><tr><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td></tr></table></td><td>0.031</td></tr>
<tr><td>4</td><td>3</td><td>0</td><td></td><td>0.030</td></tr>
<tr><td>4</td><td>4</td><td>0</td><td></td><td>0.026</td></tr>
<tr><td>4</td><td>5</td><td>0</td><td></td><td>0.028</td></tr>
<tr><td>4</td><td>6</td><td>0</td><td></td><td>0.027</td></tr>
<tr><td>4</td><td>7</td><td>0</td><td></td><td>0.029</td></tr>
<tr><td>4</td><td>8</td><td>0</td><td></td><td>0.026</td></tr>
<tr><td>5</td><td>1</td><td>10</td><td><table><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table></td><td>4.356</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table></td><td></td></tr>
<tr><td>5</td><td>2</td><td>16</td><td><table>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
</table></td><td>0.172</td></tr>
<tr><td>5</td><td>3</td><td>0</td><td></td><td>0.054</td></tr>
<tr><td>5</td><td>4</td><td>0</td><td></td><td>0.029</td></tr>
<tr><td>5</td><td>5</td><td>0</td><td></td><td>0.029</td></tr>
<tr><td>5</td><td>6</td><td>0</td><td></td><td>0.031</td></tr>
<tr><td>5</td><td>7</td><td>0</td><td></td><td>0.029</td></tr>
<tr><td>5</td><td>8</td><td>0</td><td></td><td>0.027</td></tr>
<tr><td>6</td><td>1</td><td>16</td><td><table>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table></td><td>40.179</td></tr>
<tr><td>6</td><td>2</td><td>20</td><td><table>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
</table></td><td>1.207</td></tr>
</table>

<table>
<tr><td></td><td></td><td></td><td><table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table></td><td></td></tr>
<tr><td>6</td><td>3</td><td>16</td><td><table><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table></td><td>0.193</td></tr>
<tr><td>6</td><td>4</td><td>0</td><td></td><td>0.040</td></tr>
<tr><td>6</td><td>5</td><td>0</td><td></td><td>0.031</td></tr>
<tr><td>6</td><td>6</td><td>0</td><td></td><td>0.029</td></tr>
<tr><td>6</td><td>7</td><td>0</td><td></td><td>0.026</td></tr>
<tr><td>6</td><td>8</td><td>0</td><td></td><td>0.038</td></tr>
<tr><td>7</td><td>1</td><td></td><td></td><td>> 600</td></tr>
<tr><td>7</td><td>2</td><td>24</td><td><table><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table></td><td>455.405</td></tr>
</table>

			$\begin{array}{\|c\|c\|c\|c\|c\|c\|c\|}\hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline \end{array}$	
7	3	20	$\begin{array}{\|c\|c\|c\|c\|c\|c\|c\|}\hline 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ \hline \end{array}$	4.966
7	4	16	$\begin{array}{\|c\|c\|c\|c\|c\|c\|c\|}\hline 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline \end{array}$	0.059
7	5	0		0.041
7	6	0		0.037
7	7	0		0.043
7	8	0		0.040
8	1			> 600
8	2			> 600

<table>
<tr><td>8</td><td>3</td><td>32</td><td>
<table>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table>
</td><td>166.167</td></tr>
<tr><td>8</td><td>4</td><td>16</td><td>
<table>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td><td>0.134</td></tr>
<tr><td>8</td><td>5</td><td>0</td><td></td><td>0.067</td></tr>
<tr><td>8</td><td>6</td><td>0</td><td></td><td>0.072</td></tr>
<tr><td>8</td><td>7</td><td>0</td><td></td><td>0.049</td></tr>
<tr><td>8</td><td>8</td><td>0</td><td></td><td>0.065</td></tr>
</table>