Homework #2

Jonathan Beaubien
Math 381 - Discrete Modeling
University of Washington

January 22nd, 2021

Equally Attacking Knights

In this write-up I will explain attempt to find some solutions to the equally attacking knights problem.
This problem consists of asking the question: what is the maximal number of knights that we can place on an n by n chessboard such that each knight is being attacked by exactly m knights.
To begin, we must start defining some variables.
We will use the notation $x_{ij} \in \{0, 1\}$ to be a binary variable representing if there is (or isn't) a knight on the square at row i and column j on the chessboard.
This can be summarized as:

$$x_{ij} \in \{0, 1\} \text{ where } 1 \leq i \leq n, 1 \leq j \leq n$$

If $x_{ij} = 0$ then there is not a knight on that square and, similarly, if $x_{ij} = 1$ then there is a knight on that square.
Also, our chessboard starts at index 1 and ends at index n.
Next, we need to find some constraints and an objective function that help define the problem we are trying to solve.
Our objective is to find the maximal number of knights (given the rest of the constraints) so our objective function should be:

$$\max \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}$$

The important constraint we need to describe is this statement:

If there is a knight, it is attacked by exactly m other knights

This can be done by using two constraints,

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \geq m x_{ij} \tag{1}$$

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \leq m x_{ij} + 8(1 - x_{ij}) \tag{2}$$

To prove to you that these constraints are equivalent to the statement above I will examine all possible permutations.
First, consider the case that $x_{ij} = 0$. We arrive with the two constraints being

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \geq 0$$

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \leq 8$$

These two constraints are valid. If there is no knight on a square, we do not care about whether there are 0 or even 8 knights attacking that square (8 is the maximum amount of knights that can attack a square).

	x		x	
x				x
		K		
x				x
	x		x	

Next, we can consider the case that $x_{ij} = 1$.

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \geq m$$

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \leq m$$

Together, these two constraints taken in combination require

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} = m$$

This makes perfect sense in the context of our problem: if there is a knight on a square, we want there to be exactly m knights attacking that square.

Code

Now that we have defined our problem mathematically, we can now construct our .lp file to be solved using lpsolve.
As an initial step I will create some helper functions that return a list of attacking squares given any square on an n by n board.

```
# This function returns a list of all the squares on an n x n chess board
# in which a knight on that square attacks the given square (row, column).
def getAttackingSquares(row, column, n):
  # This is a list of all potential (or possible) squares.
  potentialSquares = [
    (row + 2, column + 1),
    (row - 2, column - 1),
    (row + 1, column + 2),
    (row - 1, column - 2),
    (row + 2, column - 1),
    (row - 2, column + 1),
    (row + 1, column - 2),
    (row - 1, column + 2),
  ]
  squares = []
  # We loop through all potential squares.
  for square in potentialSquares:
    # If a square is actually within the bounds of the board
    # it gets added to the list of attacking squares.
    if validSquare(square, n):
      squares.append(square)

  return squares

# This function checks if the given coordinates are valid on an n x n chess board
# indexed starting at 0 and ending with n.
# (i.e. greater than 0 and less than n + 1)
def validSquare(location, n):
  for coordinate in location:
    if coordinate < 1 or coordinate > n:
      return False
  return True
```

Next, I will create another function that will construct the lpsolve input file (saved in the format 'eq_attacking_knights_n_m.lp') using n, m, both constraints, and the objective function.

```
# This function takes the length and width of a chess board (n x n) as well
# as a number of knights (m) to answer the question:

# "What is the maximal number of knights that we can place on an n by n chessboard
# such that each knight is attacking exactly m knights?"

# By creating and saving an lpsolve file in the format: 'eq_attacking_knights_n_m.lp'
# Each binary variable, x_i_j, represents a square in which a knight exists on (1) or doesn'
    t (0).
def equallyAttackingKnightsLPSolve(n, m):

  # Create and name the lpsolve file to write to.
  f = open('eq_attacking_knights_' + str(n) + '_' + str(m) + '.lp', "w+")

  # Write the objective function that maximizes the number of knights on the board
  # i.e. maximize sum(x_i_j) for 1<=i<=n, 1<=j<=n
  f.write('max: ')
  for i in range(1, n + 1):
    for j in range(1, n + 1):
      f.write('+x_' + str(i) + '_' + str(j))
  f.write(";\n")

  # Now I will write the many constraints (each square creates two different constraints)
  # We will loop over each square, creating
  for i in range(1, n + 1):
    for j in range(1, n + 1):
      attackingSquares = getAttackingSquares(i, j, n)

      # If there are no attacking squares, we fix the value of that square to 0.
```

```
      # Otherwise, we continue creating the two constraints for each square.
      if not attackingSquares:
        f.write('assign_x_' + str(i) + '_' + str(j) + ': x_' + str(i) + '_' + str(j) + '=0;\
            n')
      else:
        # Constraint 1: sum_[a,b: knight at a,b attacks i,j](x_a_b) >= m * x_i_j
        for attackingSquare in attackingSquares:
          f.write('+x_' + str(attackingSquare[0]) + '_' + str(attackingSquare[1]))
        f.write(' >= ' + str(m) + '*' + 'x_' + str(i) + '_' + str(j) + ';\n')
        # Constraint 2: sum_[a,b: knight at a,b attacks i,j](x_a_b) <= m * x_i_j + 8 * (1 -
            x_i_j)
        for attackingSquare in attackingSquares:
          f.write('+x_' + str(attackingSquare[0]) + '_' + str(attackingSquare[1]))
        f.write(' <= ' + str(m) + '*' + 'x_' + str(i) + '_' + str(j) + '+8-8*x_' + str(i) +
            '_' + str(j) + ';\n')

  # Finally, we declare all the variables as binary since there is
  # either a knight or no knight on each square.
  f.write('bin ')
  for i in range(1, n + 1):
    for j in range(1, n + 1):
      if not (i == 1 and j == 1):
        f.write(',')
      f.write('x_' + str(i) + '_' + str(j))
  f.write(";")
  f.close()
```

This function makes it incredibly easy to create an lpsolve input file for any value of n or m simply by calling the function. As an example,

```
equallyAttackingKnightsLPSolve(4, 1)
```

Will be saved as 'eq_attacking_knights_4_1.lp'. This is the (succinct) contents of that file:

```
max: +x_1_1+x_1_2+x_1_3+x_1_4+x_2_1+x_2_2+x_2_3+x_2_4+ ... +x_4_4;
+x_3_2+x_2_3 >= 1*x_1_1;
+x_3_2+x_2_3 <= 1*x_1_1+8-8*x_1_1;
+x_3_3+x_2_4+x_3_1 >= 1*x_1_2;
+x_3_3+x_2_4+x_3_1 <= 1*x_1_2+8-8*x_1_2;
+x_3_4+x_3_2+x_2_1 >= 1*x_1_3;
+x_3_4+x_3_2+x_2_1 <= 1*x_1_3+8-8*x_1_3;
+x_3_3+x_2_2 >= 1*x_1_4;
+x_3_3+x_2_2 <= 1*x_1_4+8-8*x_1_4;
+x_4_2+x_3_3+x_1_3 >= 1*x_2_1;
+x_4_2+x_3_3+x_1_3 <= 1*x_2_1+8-8*x_2_1;
+x_4_3+x_3_4+x_4_1+x_1_4 >= 1*x_2_2;

...
... (there are more lines, ellipses for succinctness)
...

bin x_1_1,x_1_2,x_1_3,x_1_4,x_2_1,x_2_2,x_2_3,x_2_4, ... ,x_4_4;
```

After running this file through lpsolve we get the output:

```
Value of objective function: 8.00000000

Actual values of the variables:
x_1_1                         1
x_1_2                         1
x_1_3                         1
x_1_4                         1
x_2_1                         1
x_2_2                         1
x_2_3                         1
x_2_4                         1
```

This output tells us that the first two rows of the board are filled with knights, giving us a maximum number of 8 knights on a 4 by 4 board such that each knight is being attacked by one knight.

K	K	K	K
K	K	K	K

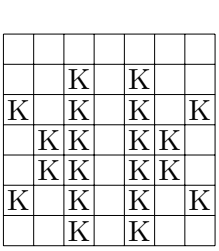
$n = 4, m = 1$
(8)

Solutions

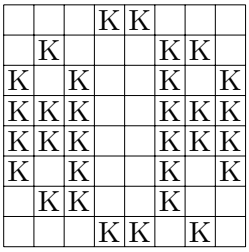
I ran lpsolve on many trials for $4 \leq n \leq 8$ and $1 \leq m \leq 3$ and summarized these results in this table

Sol. Found	Board Size (n)	Attacking Knights (m)	Max Knights	Computation Time
Y	4	1	8	< 1 sec
Y	4	2	10	< 1 sec
Y	4	3	0	< 1 sec
Y	5	1	10	< 1 sec
Y	5	2	16	< 1 sec
Y	5	3	0	< 1 sec
Y	6	1	16	9 sec
Y	6	2	20	< 1 sec
Y	6	3	16	< 1 sec
N	7	1	N/A	N/A
Y	7	2	24	150 sec
Y	7	3	20	1.5 sec
N	8	1	N/A	N/A
N	8	2	N/A	N/A
Y	8	3	32	52 sec

The rows in which a solution was not found were let to run for a long time (even an hour on one) but did not find a solution within that time.
It seems that an m of 1 or 2 takes longer to discover.
Here is a simple collection of some of the bigger boards ($n = 7, 8$).



$n = 7, m = 3$
(20)



$n = 8, m = 3$
(32)

The $n = 7, m = 3$ solution is sort of unique as it looks almost like a mirrored letter 'K'!
I believe if I had a faster computer I would be able to do more calculations in less time.
A way to make the calculations faster would be to limit our second constraint (2) to be, instead of 8, the number of squares that a knight would attack the current square.

$$\sum_{a,b\text{: knight at } a,b \text{ attacks i,j square}} x_{ab} \leq m x_{ij} + k(1 - x_{ij}) \text{ where } k = \#\text{ of possible } a, b \text{ pairs}$$