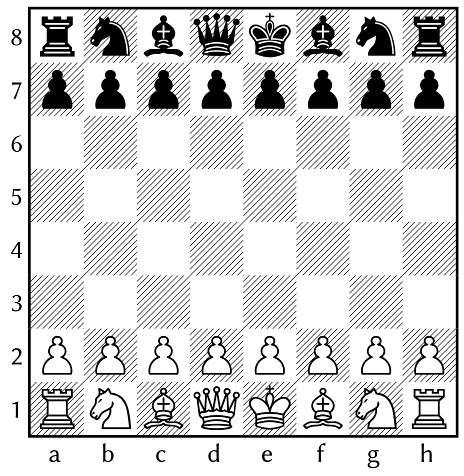
Knight's Attack

Ryan Mouw

January 21, 2021

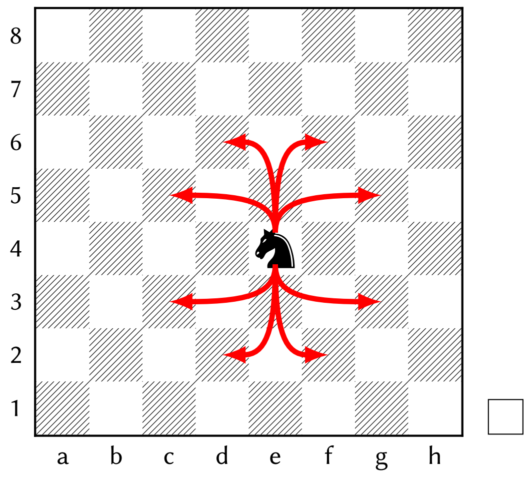
1 Background

Traditionally, a chessboard is a square set of 8x8 alternating dark and light tiles. Each player has one queen, one king, two rooks, two knights, two bishops, and eight pawns (as shown below).

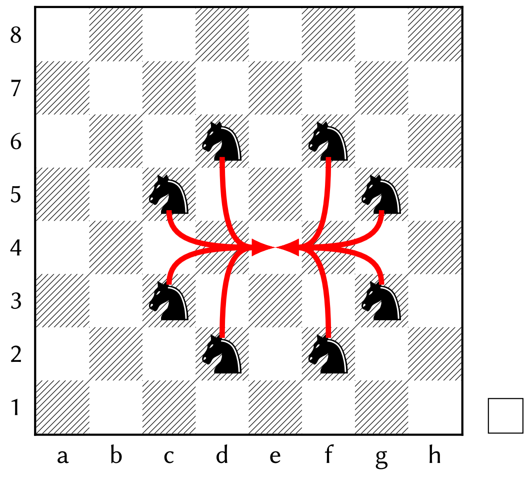


2 The Problem

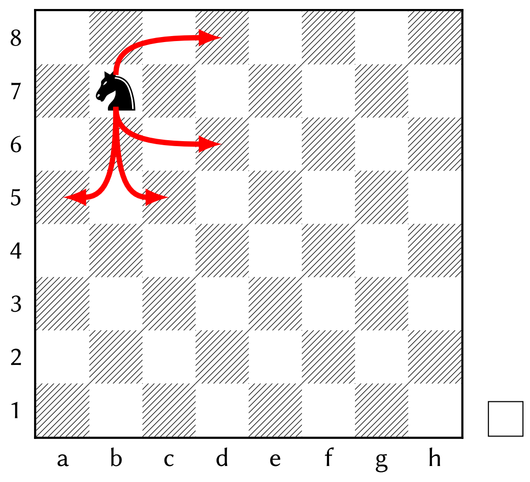
The knight is only allowed to move or attack in an "L" shape. When a knight is placed on the chessboard, it is "attacking" up to 8 different positions at any one one time. See the image below.



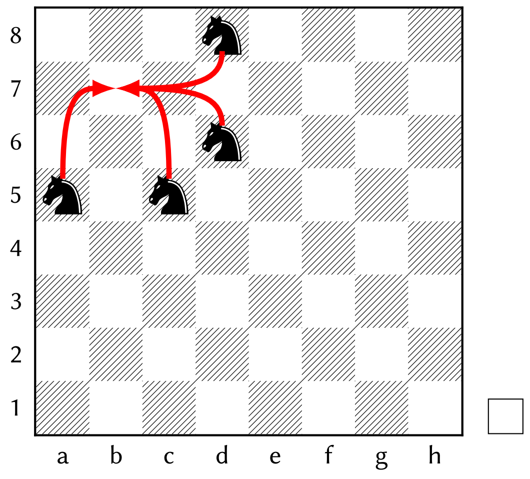
Conversely, there are up to 8 other knights on the board that can attack any 1 tile on the board.



If the knight is placed towards the edge of the board, its attacking squares are limited to a subset of the original 8. See image below.



Conversely, fewer knights can attack a tile that is close to the edge of the board.

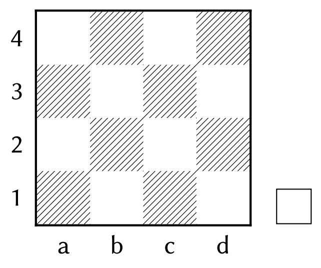


Note: In chess it is best to keep the knight towards the center of the board it increase it's effectiveness.

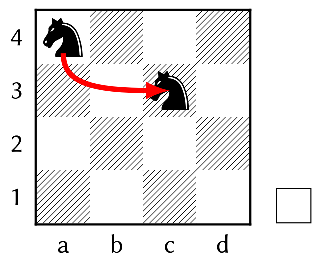
Now, let's introduce the problem.

What is the maximum number of knights that can be placed on an $n \times n$ chess board such that each knight is attacking exactly m knights? To ask the same question in a different way; what is the maximal number of knights that we can place on an $n \times n$ chessboard such that each knight is being attacked by exactly m knights?

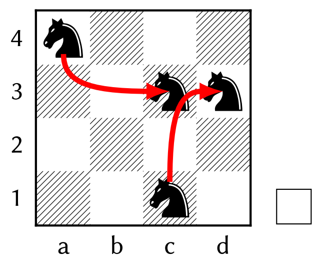
For demonstration purposes we will be using a 4×4 chessboard. See below.



An example of a situation where every knight on the board is being attacked exactly $m = 1$ times is shown below.



But is this the maximum number of knights that can be placed on the board where each knight is being attacked exactly $m = 1$ times? A quick check reveals it is not. See below.



To maximize the number of knights on our 4×4 chessboard that are each being attacked by $m = 1$ knights, we will use linear programming.

3 Objective Function and Constraints

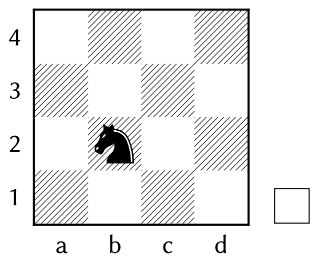
The objective function of a linear program represents the thing in the problem that needs to be maximized or minimized. In our case, we want to maximize the number of knights on the $n \times n$ board. To do this, we will create a set of variables x_{ij} that represent whether a knight is on tile (i, j).

Notice that there is only two options for each tile on the chessboard. Either the tile has a knight placed on it, or it does not. We can incorporate this constraint into our linear program by making the x_{ij} variables binary. If $x_{ij} = 1$ then there is a knight at the (i, j) tile on the chessboard. If $x_{ij} = 0$ then the (i, j) tile is empty.

$$x_{ij} \in \{0, 1\} \qquad [1]$$

Note: The indexing used in the pictures is different than the indexing used in our linear program. The picture uses $rows = 4, 3, 2, 1$ to index what I will call $rows\ i = 1, 2, 3, 4$ and uses $columns = a, b, c, d$ to reference what I will call $columns\ j = 1, 2, 3, 4$, respectively. For example, tile $b2$ in the image will be referred to as tile $(3, 2)$.

$$x_{32} = 1$$

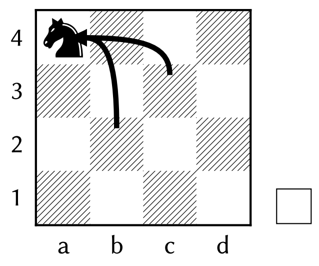


If $x_{ij} = 1$ when there is a knight on tile (i, j) and $x_{ij} = 0$ when tile (i, j) is empty, then we must maximize the sum of all the x_{ij} variables. Because $i = n$ $(the\ number\ of\ rows)$ and $j = n$ $(the\ number\ of\ columns)$, our objective function will be to maximize the follow summation;

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \qquad [2]$$

Further, we need to ensure that each knight on the board is being attacked by exactly m other knights. To implement this, we must add further constraints.

As discussed before, each tile (i, j) has a number of "attacking tiles" associated with it. For example, if there is a knight on tile $(1, 1)$ on a 4×4 board, then the "attacking tiles" associated with it are tiles $(2, 3)$ and $(3, 2)$. See the image below.



What we must do is imply that if $x_{ij} = 1$, then the sum of the "attacking tiles" associated with (i, j) that have knights on them is equal to m. Let;

$$a_{ij} \subseteq \{x_{ij} \; variables \; that \; represent \; attacking \; tiles \; of \; (i, j)\}$$

Then;

$$\sum a_{ij} \geq m \times x_{ij} \qquad [3]$$

Let's check that this is correct.

If $x_{ij} = 0$ then the inequality becomes;

$$\sum a_{ij} \geq 0$$

This is exactly what we want. If there is no knight on (i, j) then it does not matter how many knights attack (i, j).

If $x_{ij} = 1$, then the inequality becomes;

$$\sum a_{ij} \geq m$$

Again this is what we want. If there is a knight on tile (i, j) then the sum of the knights on attacking tiles must equal at least m.

Notice that if we stopped here, the sum of attacking tiles could be greater than m. We must apply an upper bound constraint, requiring that the sum of the attacking tiles is not greater than m. We know that a single knight can attack no more than 8 other tiles at any one time. So we will require our upper bound to be 8. To do this, we add the constraint;

$$\sum a_{ij} \leq 8 - (8 - m)x_{ij} \qquad [4]$$

Let's check if this is correct. If $x_{ij} = 1$ then the inequality becomes;

$$\sum a_{ij} \leq m$$

This is exactly what we want. If there is a knight on tile (i, j) then the sum of the knights on attacking tiles must equal at most m.

If $x_{ij} = 0$ then the inequality becomes;

$$\sum a_{ij} \leq 8$$

This again is exactly what we want. If there is no knight on (i, j) then up to 8 knights could attack tile (i, j).

Note: LPSolve only works with non-negative numbers, so when inputting constraint (4) into LPSolve, the negative expression on the right hand side of the inequality must be added to both sides, resulting in:

$$\sum a_{ij} + (8 - m)x_{ij} \leq 8$$

Applying inequalities (3) and (4) as constraints requires that the number of times any knight is attacked always equals m.

4 Objective Function/ Constraint Structures for LPSolve Input File

For demonstrating the structure of the LPSolve input file, we will assume $n = 4$ and $m = 1$. The objective function needs to be maximized in this problem. So the form of our objective function (2) LPSolve will be as follows;

```
max: +x_1_1+x_1_2+x_1_3...+x_4_4;
```

The first (3) and second (4) constraints can be written consecutively for each variable x_{ij} and take on the form;

```
+x_2_3+x_3_2 >= +1*x_1_1;
+x_2_3+x_3_2 + 7*x_1_1 <= 8;
+x_2_4+x_3_3+x_3_1 >= +1*x_1_2;
+x_2_4+x_3_3+x_3_1 + 7*x_1_2 <= 8;
.........................
.........................
+x_2_3+x_3_2 >= +1*x_4_4;
+x_2_3+x_3_2 + 7*x_4_4 <= 8;
```

Lastly we must require that the variables x_{ij} be binary. Constraint (1) takes the form;

```
    bin  x_1_1, x_1_2, x_1_3,..., x_4_3, x_4_4;
```

All together, the LPSolve input file (4×4, $m = 1$) will have the form;

```
    max: +x_1_1+x_1_2+x_1_3...+x_4_4;

    +x_2_3+x_3_2 >= +1*x_1_1;
    +x_2_3+x_3_2 + 7*x_1_1 <= 8;
    +x_2_4+x_3_3+x_3_1 >= +1*x_1_2;
    +x_2_4+x_3_3+x_3_1 + 7*x_1_2 <= 8;
    .........................
    .........................
    +x_2_3+x_3_2 >= +1*x_4_4;
    +x_2_3+x_3_2 + 7*x_4_4 <= 8;


    bin  x_1_1, x_1_2, x_1_3,..., x_4_3, x_4_4;
```

5 Solutions

The LPSolve output file for $n = 4$; $m = 1$ is below.

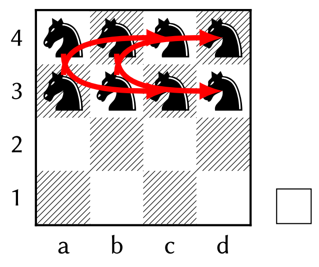
```
 Value of objective function: 8.00000000

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
```

Note: LPSolve output files only provide non-zero variables. All variables (x_{ij}) not listed are zero.

Remember that

```
    x_1_1                         1
```

signifies that there is a $x_{11} = 1$. Let's check that this solution is correct.



This works! Every knight is attacked exactly 1 time.

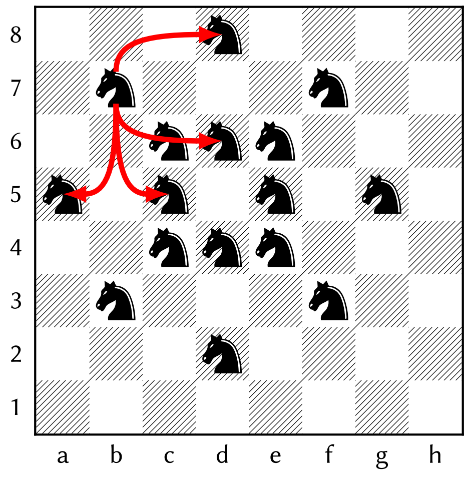
Below is a table with each row representing m, and each column representing n. Where each row and column meet, the solution is given (# of knights (k), time (s)). N/A is given when LPSolve could not find a solution in under 10 minutes. This does not mean that an optimal solution does not exist.

If the solution for a certain value of m on a board of size $n \times n$ is 0, then the solution for a board $k \times k$ where $k < n$ must be zero because the $k \times k$ board is a subset of the $n \times n$ board. In this case, no times will be given as the solution is deduced.

$n \times n$	4×4	5×5	6×6	7×7	8×8
$m = 1$	8 k, 0.017 s	10 k, 3.814 s	16 k, 17.957 s	N/A	N/A
$m = 2$	10 k, 0.018 s	16 k, 0.090 s	20 k, 0.503 s	24 k, 219.706 s	N/A
$m = 3$	0 k	0 k, 0.036 s	16 k, 0.109 s	20 k, 2.190 s	32 k, 80.315 s
$m = 4$	0 k	0 k	0 k, 0.035 s	16 k, 0.041 s	16 k, 0.087 s
$m = 5$	0 k	0 k	0 k	0 k	0 k, 0.032 s
$m = 6$	0 k	0 k	0 k	0 k	0 k, 0.016 s
$m = 7$	0 k	0 k	0 k	0 k	0 k, 0.019 s
$m = 8$	0 k	0 k	0 k	0 k	0 k, 0.019 s

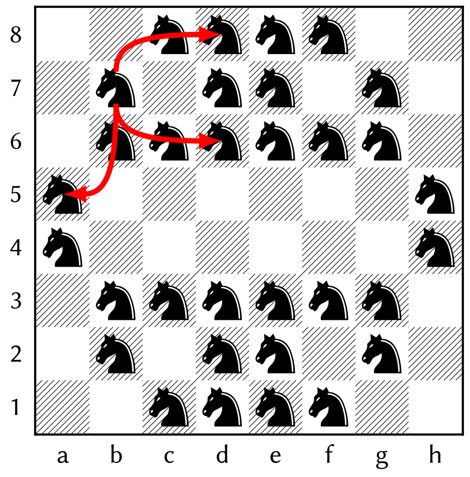
Feel free to verify some of the more complex solutions below.

(n = 8 and m = 4):



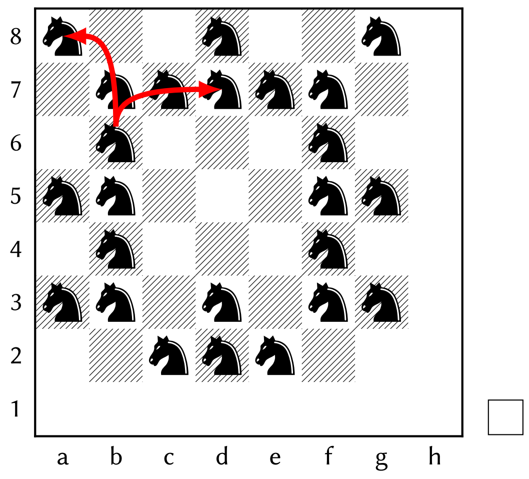
$\square$

(n = 8 and m = 3):

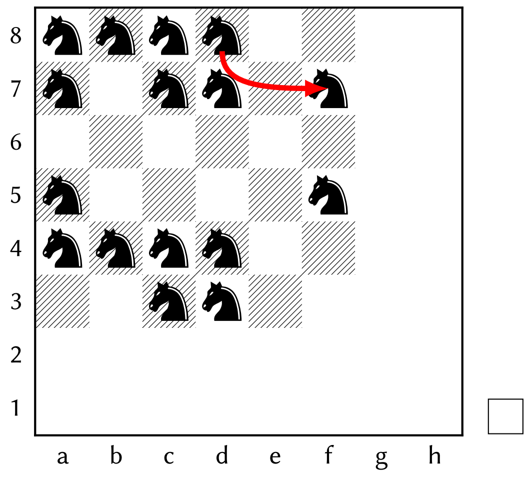


$\square$

(n = 7 and m = 2):



(n = 6 and m = 1):



6 Python Code for LPSolve Input File

```
#declare n and m variables
n = 8
m = 8

##Create variables that represent each square on chess board of size nxn
obj_func = ""
for i in range(n):
    for j in range(n):
        obj_func = obj_func + " +" + "x_"+str(i+1)+"_"+str(j+1)
obj_func = "max: "+obj_func+";"+"\n"



#create and array that will hold the (i,j) attacking tiles of variable x_i_j

knight_array = []

#8 if statements are needed to check if each (i,j) tile has attacking tiles
#that are off the board of size nxn
for i in range(n):
    temp = []
    for j in range(n):
        constraint_1 = ""
        #(-2,-1)
        if 0 < i+1-2 < n+1 and 0 < j+1-1 < n+1:
            constraint_1 = constraint_1+"+x_"+ str(i+1-2)+"_"+str(j+1-1)
        #(-2,1)
        if 0 < i+1-2 < n+1 and 0 < j+1+1 < n+1:
            constraint_1 = constraint_1+"+x_"+ str(i+1-2)+"_"+str(j+1+1)
        #(-1,2)
        if 0 < i+1-1 < n+1 and 0 < j+1+2 < n+1:
            constraint_1 = constraint_1+"+x_"+ str(i+1-1)+"_"+str(j+1+2)
        #(1,2)
        if n+1 > i+1+1 > 0 and n+1 > j+1+2 > 0:
            constraint_1 = constraint_1+"+x_"+ str(i+1+1)+"_"+str(j+1+2)
        #(2,1)
        if n+1 > i+1+2 > 0 and n+1 > j+1+1 > 0:
            constraint_1 = constraint_1+"+x_"+ str(i+1+2)+"_"+str(j+1+1)
        #(2,-1)
        if n+1 > i+1+2 > 0 and n+1 > j+1-1 > 0:
            constraint_1 = constraint_1+"+x_"+ str(i+1+2)+"_"+str(j+1-1)
        #(1,-2)
        if n+1 > i+1+1 > 0 and n+1 > j+1-2 > 0:
            constraint_1 = constraint_1+"+x_"+ str(i+1+1)+"_"+str(j+1-2)
        #(-1,-2)
        if n+1 > i+1-1 > 0 and n+1 > j+1-2 > 0:
            constraint_1 = constraint_1+"+x_"+ str(i+1-1)+"_"+str(j+1-2)
        temp.append(constraint_1)
    knight_array.append(temp)


#This creates the two constraints that imply attacking squares must be
#m or <=8
attack_constraint = ''
for i in range(n):
    for j in range(n):

        attack_constraint = attack_constraint +knight_array[i][j]+ " >= +"+
        str(m)+"*x_"+str(i+1)+"_"+str(j+1)+";" +"\n"

        attack_constraint = attack_constraint +knight_array[i][j]+" +
        "+str(8-m)+"*x_"+str(i+1)+"_"+str(j+1)+ " <= " + "8;" +"\n"



    #This creates the binary constraint
    bin_con = ""
    for i in range(n):
        for j in range(n):
            bin_con = bin_con + " x_"+str(i+1)+"_"+str(j+1)+","
    bin_con = "bin "+bin_con[:-1]+";"

    #This prints the output to a txt file
    with open("knight_2.17_m8_n8.txt", "w") as text_file:
        print(obj_func, file = text_file)
        print(attack_constraint, file = text_file)
        print(bin_con, file = text_file)
```