Chess Domination Problem: What is the least number of knights needed to cover the entire board?

Jiasui(Maggie) Qin

October 16, 2020

Let's suppose that we have a square chessboard, a queen, and a bishop. Our goal is to put the minimum number of knights so that all squares can be attacked. And we will always use the queen and the bishop. To solve this problem, I will create a linear Lpsolve program.

Mathematical formulation:

For a $n \times n$ square chessboard, we use i and j to represent the row number and the column number:

$$0 \leq i < n, 0 \leq j < n$$

We define k_{ij} to be a binary variable that will be equal to one if we decide to put a knight at that position:

$$k_{ij} \in \{0, 1\}$$

We define q_{ij} to be a binary variable that will be equal to one if we decide to put the queen at that position:

$$q_{ij} \in \{0, 1\}$$

We define b_{ij} to be a binary variable that will be equal to one if we decide to put the bishop at that position:

$$b_{ij} \in \{0, 1\}$$

To minimize the number of knights, our objective function is the sum of all k values:

$$\sum_{i=0}^{n} \sum_{j=0}^{n} k_{ij}$$

There are several constraints to consider. First of all, we only have one bishop and one queen. Therefore, the number of bishop and the number of queen on the chess board must be equal to one:

$$\sum_{i=0}^{n}\sum_{j=0}^{n} q_{ij} = 1, \sum_{i=0}^{n}\sum_{j=0}^{n} b_{ij} = 1$$

In addition, a single square can only have one chess piece, either a knight, or a queen, or a bishop:

$$k_{ij} + q_{ij} + b_{ij} \leq 1$$

Lastly, for each square, we need to make sure that at least one of either the knights, or the queen, or the bishop could attack it. Therefore, among all possible positions to attack, one of them has to be occupied:

$$\sum_{\substack{(i,j):\\ \text{knight at } (i,j)\\ \text{attacks } (a,b)}} k_{i,j} + \sum_{\substack{(i,j):\\ \text{queen at } (i,j)\\ \text{attacks } (a,b)}} q_{i,j} + \sum_{\substack{(i,j):\\ \text{bishop at } (i,j)\\ \text{attacks } (a,b)}} b_{i,j} \geq 1 \text{ for all } 1 \leq a, b \leq n$$

Generate the input file:

To use Lpsolve to solve this LP problem, I wrote the following python code to generate the Lpsolve input file:

```
# Jiasui Qin
#
# This code outputs an lpsolve-formatted LP which
# decides how to use the minimum number of knights
# to attack the entire chess board.

import numpy as np
import math

# define the size of the chess board
size = 6

# print the objective function
outstring = "min: "
for i in range(0, size):
    for j in range(0, size):
        outstring = outstring + "+k_" + str(i) + "_" + str(j)
outstring = outstring+";"
print(outstring)

# generate a constraint that we use exactly one queen
outstring = ""
for i in range(0, size):
    for j in range(0, size):
        outstring = outstring + "+q_" + str(i) + "_" + str(j)
outstring = outstring+"=1;"
print(outstring)
```

```
# generate a constraint that we use exactly one bishop
outstring = ""
for i in range(0, size):
    for j in range(0, size):
        outstring = outstring + "+b_" + str(i) + "_" + str(j)
outstring = outstring+"=1;"
print(outstring)

# generate a constraint that each square can only has one piece
outstring = ""
for i in range(0, size):
    for j in range(0, size):
        outstring = outstring + "+k_" + str(i) + "_" + str(j) + "+
            ↪ q_" + str(i) + "_" + str(j) + "+b_" + str(i) + "_" +
            ↪  str(j)
        outstring = outstring+"<=1;\n"
print(outstring)

# generate a constraint to ensure that each square on
# the board is attacked
outstring = ""
for i in range(0, size):
    for j in range(0, size):
        # find positions of knights that can attack the square
        if(i-2>-1):
            if(j-1>-1):
                outstring = outstring + "+k_" + str(i-2) + "_" +
                    ↪ str(j-1)
            if(j+1<size):
                outstring = outstring + "+k_" + str(i-2) + "_" +
                    ↪ str(j+1)
        if(i-1>-1):
            if(j-2>-1):
                outstring = outstring + "+k_" + str(i-1) + "_" +
                    ↪ str(j-2)
            if(j+2<size):
                outstring = outstring + "+k_" + str(i-1) + "_" +
                    ↪ str(j+2)
        if(i+1<size):
            if(j-2>-1):
                outstring = outstring + "+k_" + str(i+1) + "_" +
                    ↪ str(j-2)
            if(j+2<size):
                outstring = outstring + "+k_" + str(i+1) + "_" +
                    ↪ str(j+2)
        if(i+2<size):
            if(j-1>-1):
                outstring = outstring + "+k_" + str(i+2) + "_" +
                    ↪ str(j-1)
            if(j+1<size):
                outstring = outstring + "+k_" + str(i+2) + "_" +
                    ↪ str(j+1)
        # find vertical and horizontal positions where we can place
            ↪  the queen
        for q_up in range(0,i):
            outstring = outstring+"+q_" + str(q_up) + "_" + str(j)
```

```
        for q_down in range(i+1,size):
            outstring = outstring+"+q_" + str(q_down) + "_" + str(j
                ↪ )
        for q_left in range(0,j):
            outstring = outstring+"+q_" + str(i) + "_" + str(q_left
                ↪ )
        for q_right in range(j+1,size):
            outstring = outstring+"+q_" + str(i) + "_" + str(
                ↪ q_right)
        # find diagonal positions where we can place the queen or
            ↪ the bishop
        x = j-1
        y = i-1
        while x>-1 and y>-1:
            outstring = outstring+"+q_" + str(y) + "_" + str(x) + "
                ↪ +b_" + str(y) + "_" + str(x)
            x = x-1
            y = y-1
        x = j+1
        y = i-1
        while x<size and y>-1:
            outstring = outstring+"+q_" + str(y) + "_" + str(x) + "
                ↪ +b_" + str(y) + "_" + str(x)
            x = x+1
            y = y-1
        x = j-1
        y = i+1
        while x>-1 and y<size:
            outstring = outstring+"+q_" + str(y) + "_" + str(x) + "
                ↪ +b_" + str(y) + "_" + str(x)
            x = x-1
            y = y+1
        x = j+1
        y = i+1
        while x<size and y<size:
            outstring = outstring+"+q_" + str(y) + "_" + str(x) + "
                ↪ +b_" + str(y) + "_" + str(x)
            x = x+1
            y = y+1
        outstring = outstring+">=1;\n"
print(outstring)

# define variables to be bianary
outstring = "bin "
for i in range(0, size):
    for j in range(0, size):
        if (i+j > 0):
            outstring = outstring+","
        outstring = outstring + "k_" + str(i) + "_" + str(j) + ",q_
            ↪ " + str(i) + "_" + str(j) + ",b_" + str(i) + "_" +
            ↪ str(j)
outstring = outstring+";"
print(outstring)
```

After I ran the python code while setting the size to 6, I got the following Lp-solve input file:

```
min: +k_0_0+k_0_1+k_0_2+k_0_3+k_0_4+k_0_5+k_1_0+k_1_1+k_1_2+k_1_3+
    ↪ k_1_4+k_1_5+k_2_0+k_2_1+k_2_2+k_2_3+k_2_4+k_2_5+k_3_0+k_3_1+
    ↪ k_3_2+k_3_3+k_3_4+k_3_5+k_4_0+k_4_1+k_4_2+k_4_3+k_4_4+k_4_5+
    ↪ k_5_0+k_5_1+k_5_2+k_5_3+k_5_4+k_5_5;
(ensure that we use exactly one queen)
+q_0_0+q_0_1+...+q_5_5=1;
(ensure that we use exactly one bishop)
+b_0_0+b_0_1+...+b_5_5=1;
(36 lines of the following type:
ensure that each square contains no more than one chess piece)
+k_0_0+q_0_0+b_0_0<=1;
.
.
(36 lines of the following type:
ensure every square can be attacked)
+k_1_2+k_2_1+q_1_0+q_2_0+q_3_0+q_4_0+q_5_0+q_0_1+q_0_2+q_0_3+q_0_4+
    ↪ q_0_5+q_1_1+b_1_1+q_2_2+b_2_2+q_3_3+b_3_3+q_4_4+b_4_4+q_5_5+
    ↪ b_5_5>=1;
.
.
(ensure all variables are binary)
bin k_0_0,q_0_0,b_0_0,k_0_1,q_0_1,b_0_1,...,k_5_5,q_5_5,b_5_5;
```

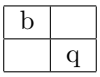
Conclusion

I then generated different input files for different board sizes and ran the Lp-solve. The computer I'm using is a MacBook Air. The following table shows each board size along with the resulting number of knights and the computation time:

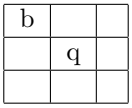
Board Size	Number of Knights	Computation Time
2×2	0	0.000608s
3×3	0	0.003394s
4×4	2	0.008439s
5×5	3	0.027019s
6×6	4	0.111238s
7×7	5	0.146121s
8×8	8	5.23695s
9×9	11	104.225s
10×10	13	128.81s
11×11	16	2267.86s

To better visualize where each chess piece is placed on the chess board, I plotted them on each sized table with k representing the knight, q representing the queen, and b representing the bishop.

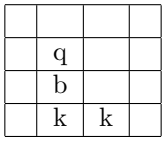
size = $2\times2 : (\mathrm{q}_{1,1}, b_{0,0})$



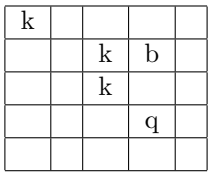
size = $3\times3 : (\mathrm{q}_{1,1}, b_{0,0})$



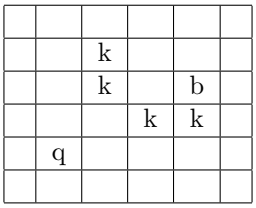
size = $4\times4 : (\mathrm{k}_{3,1}, k_{3,2}, q_{1,1}, b_{2,1})$



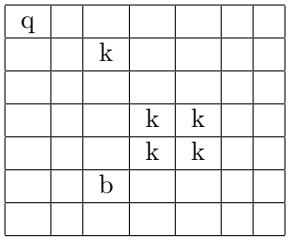
size = $5\times5 : (\mathrm{k}_{0,0}, k_{1,2}, k_{2,2}, q_{3,3}, b_{1,3})$



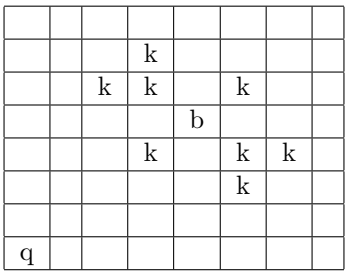
size = $6\times6 : (\mathrm{k}_{1,2}, k_{2,2}, k_{3,3}, k_{3,4}, q_{4,1}, b_{2,4})$



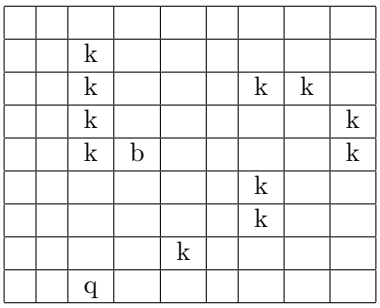
size = 7×7 : $(\mathrm{k}_{1,2}, k_{3,3}, k_{3,4}, k_{4,3}, k_{4,4}, q_{0,0}, b_{5,2})$



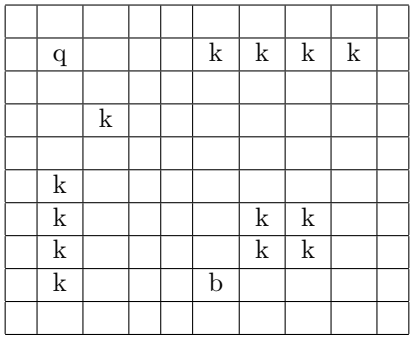
size = 8×8 : $(\mathrm{k}_{1,3}, k_{2,2}, k_{2,3}, k_{2,5}, k_{4,3}, k_{4,5}, k_{4,6}, k_{5,5}, q_{7,0}, b_{3,4})$



size = 9×9 : $(\mathrm{k}_{1,2}, k_{2,2}, k_{2,6}, k_{2,7}, k_{3,2}, k_{3,8}, k_{4,2}, k_{4,8}, k_{5,6}, k_{6,6}, k_{7,4}, q_{8,2}, b_{4,3})$



size $= 10\times10 : (\mathrm{k}_{1,5}, k_{1,6}, k_{1,7}, k_{1,8}, k_{3,2}, k_{5,1}, k_{6,1}, k_{6,6}, k_{6,7}, k_{7,1}, k_{7,6}, k_{7,7}, k_{8,1}, q_{1,1}, b_{8,5})$



size $= 11\times11 : (\mathrm{k}_{1,2}, k_{1,7}, k_{2,2}, k_{2,7}, k_{2,8}, k_{3,2}, k_{3,8}, k_{4,2}, k_{5,7}, k_{6,1}, k_{7,5}, k_{7,9}, k_{8,7}, k_{8,8}, k_{8,9}, k_{9,3}, q_{8,2}, b_{8,1})$