Math381 HW2

Siyue Zhu

January 22, 2022

Introduction

We have a chessboard with two colors of knights: black and white. Now, we are going to consider a question: what is the minimum number of knights I need to put on the chessboard so that every square on the board can be attacked by at least one black knight and one white knight?

Our chess board can be square or non-square. Also, we can only put one knight in each square. We are going to start with a chessboard of 4*4, and increase the size up. We will first look at square chessboard up to where it takes more than 10 minutes for the computer to solve the lp and then look at non-square chessboard.

General Idea and Problem Setup

I'm going to use two binary variables,

$$x_{i,j,b}(1 \leq i \leq n_x \text{ and } 1 \leq j \leq n_y)$$

to represent whether a square i,j is occupied by a black knight and

$$x_{i,j,w}(1 \leq i \leq n_x \text{ and } 1 \leq j \leq n_y)$$

to represent whether a square i,j is occupied by a white knight. Also, n_x represents the size of the chessboard in x-axis, and n_y represents the size of the chessboard in y-axis.

When we want point (x,y) be attacked at least once by both black knight and white knight, we need to make sure that there is at least one square that can attack point (x,y) is taken by the black and the white knight. Thus, if a knight is on location (i,j) then there are eight potential position that the knight can move to

$$S_{i,j} = \{(i \pm 2, j \pm 1), (i \pm 1, j \pm 2)\}$$

However, knight can only move to those locations on the board, so the actual location knight can move to are:

$$M_{i,j} = S_{i,j} \cap \{(p,q) : 1 \leq p \leq n_x, 1 \leq q \leq n_y, \quad p, q \in Z\}$$

And each square can be attacked by at most eight different knight spots, for those knight spots on the board, we have the constraint for each location (i,j):

$$\sum M_{i,j} \geq 1$$

$x_{i,j,b}$ is the location of black knight, and $x_{i,j,w}$ is the location of white knight. Each square can take at most one knight, so

$$x_{i,j,b} + x_{i,j,w} \leq 1 \text{ for all locations (i,j) on the board}$$

Our objective function is to minimize the knight we need, so we need to minimize the sum over all x values:

$$\sum_{i=1}^{n_x} \sum_{j=1}^{n_y} (x_{i,j,b} + x_{i,j,w})$$

Thus, the lp for a general board is:

($2n_xn_y$ variables of the following type: this is the object function)

minimum x_1_1_b +x_1_1_w+...+x_n_x_n_y_$b + x$_n_x_n_y_w;

subject to

(n_xn_y lines of the following type: ensure each square should be attacked by black knight)

+x_2_3_b + x_3_2_b $\geq$ 1;

......

(n_xn_y lines of the following type: ensure each square should be attacked by white knight)

+x_2_3_w + x_3_2_w $\geq$ 1;

......

(n_xn_y lines of the following type: ensure that each board square can contain at most one knight)

+x_1_1_b + x_1_1_w $\leq$ 1;

......

(ensure all variables are binary)

x_1_1_b, x_1_1_w, ...,x_n_x_n_y_b, x_n_x_n_y_w = 0 or 1

Let's take 5*5 board as an example to show the input and the output. And the lp for a 5*5 board should look like this:

(50 variables of the following type: this is the object function)

minimum x_1_1_b +x_1_1_w+...+x_5_5_b+x_5_5_w;

subject to

(25 lines of the following type: ensure each square should be attacked by black knight)

+x_2_3_b + x_3_2_b $\geq$ 1;

......

(25 lines of the following type: ensure each square should be attacked by white knight)

+x_2_3_w + x_3_2_w $\geq$ 1;

......

(25 lines of the following type: ensure that each board square can contain at most one knight)

+x_1_1_b + x_1_1_w $\leq$ 1;

......

(ensure all variables are binary)

x_1_1_b, x_1_1_w, ...,x_5_5_b, x__5_w = 0 or 1

Here is the python code I use to generate the input file for lpsolve:

```
import math\\

# nx is the board size in x-axis, and ny is the board size in y-axis.
nx=5
ny=5

# define a function to check if the position is on the board
def existence(x,y,nx,ny):
  if (x>=1 and y>=1 and x<nx and y<ny):
    return(1)
  else:
    return(0)

# moves knights can make
```

```
knightMoves=[[1,2],[1,-2],[-1,2],[-1,-2],[2,1],[2,-1],[-2,1],[-2,-1]]

# objective function
output = ""
for a in range(1,nx+1):
    for b in range(1,ny+1):
        output += "+x_"+str(a)+"_"+str(b)+"_b"+"+x_"+str(a)+"_"+str(b)+"_w"
print("min: "+output+";")

# each square should be attacked by black knight
for a in range(1,nx+1):
    for b in range(1,ny+1):
        output=""
        for move in knightMoves:
            x = a+move[0]
            y = b+move[1]
            if (existence(x,y,nx+1,ny+1)):
                output+= "+x_"+str(x)+"_"+str(y)+"_b"
        print(output +">=1;")

# each square should be attacked by white knight
for a in range(1,nx+1):
    for b in range(1,ny+1):
        output=""
        for move in knightMoves:
            x = a+move[0]
            y = b+move[1]
            if (existence(x,y,nx+1,ny+1)):
                output+= "+x_"+str(x)+"_"+str(y)+"_w"
        print(output +">=1;")

# each square can have at most one knight
for a in range(1,nx+1):
    for b in range(1,ny+1):
        print("+x_"+str(a)+"_"+str(b)+"_b"+"+x_"+str(a)+"_"+str(b)+"_w"+"<=1;")

# all variables are binary
output = "bin "
for a in range(1,nx+1):
    for b in range(1,ny+1):
        if (a>1 or b>1):
            output += ","
        output += "x_"+str(a)+"_"+str(b)+"_b"+",x_"+str(a)+"_"+str(b)+"_w"
print(output+";")
```

And the lp input file I generate by using python looks like this:

```
(50 variables of the following type:
this is the object function)
min: +x_1_1_b+x_1_1_w+......+x_5_5_b+x_5_5_w;
.
.
(25 lines of the following type:
ensure each square should be attacked by black knight)
+x_2_3_b+x_3_2_b>=1;
.
.
(25 lines of the following type:
ensure each square should be attacked by white knight)
+x_2_3_w+x_3_2_w>=1;
.
.
(25 lines of the following type:
ensure that each board square can contain at most one knight)
+x_1_1_b+x_1_1_w<=1;
.
.
(ensure all variables are binary)
bin x_1_1_b,x_1_1_w,......,x_5_5_b,x_5_5_w;
```

Result

And our result of the lp is:

```
Value of objective function: 15.00000000

Actual values of the variables:
x_1_2_w                         1
x_1_3_w                         1
x_1_4_w                         1
x_2_1_b                         1
x_2_2_b                         1
x_2_3_b                         1
x_2_4_b                         1
x_3_1_w                         1
x_3_2_w                         1
x_3_3_w                         1
x_3_4_w                         1
x_4_1_b                         1
x_4_2_b                         1
x_4_3_b                         1
x_4_4_b                         1
./lp_solve -ia hw2_lp.txt  0.01s user 0.01s system 48% cpu 0.029 total
```

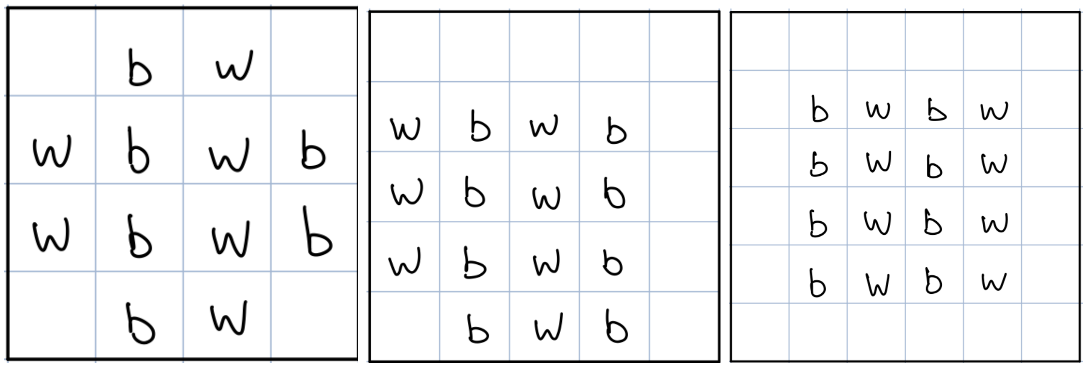
And the following table shows our results for square chessboard from 4*4 to 10*10. We stop at 10*10 because 11*11 takes more than 10 minutes to solve.

LP Result of Square Board					
size	num of knight	num of black knight	num of white knight	time	knights / board squares
4*4	12	6	6	0.01s	0.75
5*5	15	8	7	0.01s	0.6
6*6	16	8	8	0.01s	0.444
7*7	22	12	10	0.01s	0.49
8*8	28	14	14	0.1s	0.4375
9*9	38	20	18	9.05s	0.469
10*10	44	22	22	489.34s	0.44

We can see that with chessboard become larger, the number of total knight we need is increased. And the number of black knights is equal to the number of white knights when the board size is even and not equal when the board size is odd. The time consumption stays the same for the first 4 chessboard, and gets larger with the size of the board becomes larger. The ratio of the number of knight to the number of board square is getting smaller as the board size become larger, and finally stay around 0.44 to 0.49. Overall, the density of knight on the board is getting smaller and smaller.

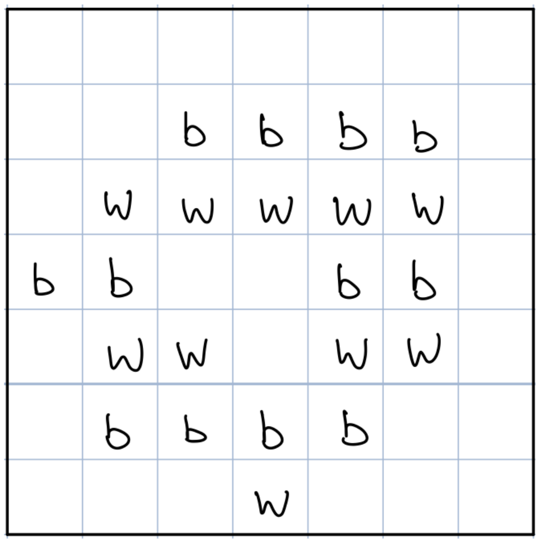
My conjecture is, with the size of the chessboard getting larger and larger, the number of knight will be greater and greater. Since one knight can at most attack 8 squares, so we need at least $\frac{2n_x n_y}{8}$ number of knights. The number of black and white knight would stay very close to each other, but not equal to each other in every scenario. Also, the time consumption would become larger and larger, and time would increase by a greater ratio as the size gets larger. Also, the ratio of the number of knights over the number of board squares would be stabilized around 0.44.

And here are graphs show the alignment of black knights and white knights from all chessboard from 4*4 to 10*10.

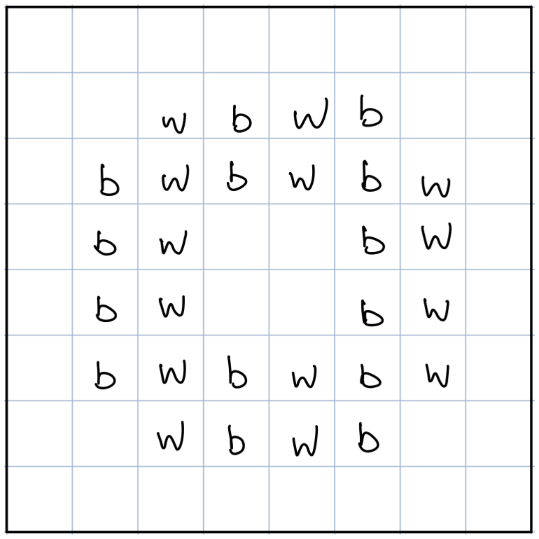


(a) (b) (c)

Figure 1: (a) 4*4 (b) 5*5 (c) 6*6

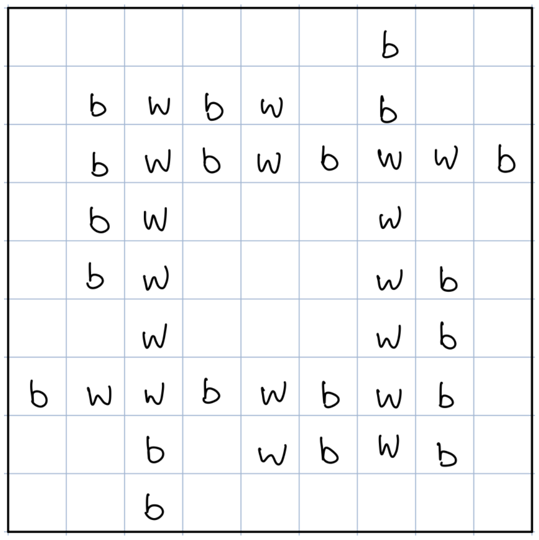


(a)

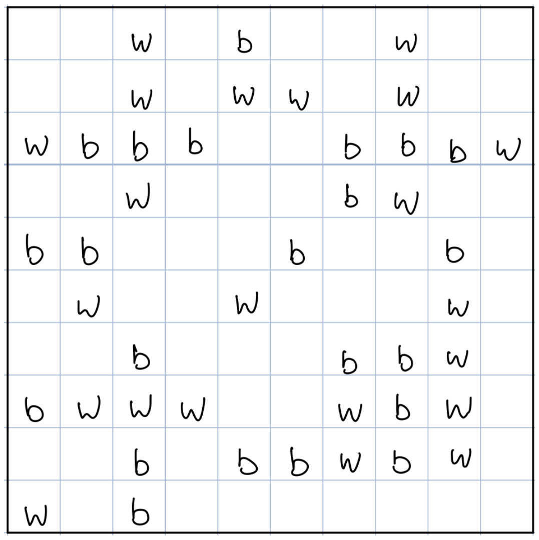


(b)

Figure 2: (d) 7*7 (e) 8*8



(a)



(b)

Figure 3: (f) 9*9 (g) 10*10

By observing the graphs, we can see some of the alignment for knights are symmetric if we fold the graph horizontally or vertivally throught the central point, such as the graph for 4*4, 6*6 and 8*8. And some graphs is symmetric if we fold the graph diagonally, such as 4*4, 5*5, 6*6, 7*7, 8*8, 9*9. I think with the chessboard gets larger and larger, the graph for even number chessboard would be like to be symmetrical over horizontal and vertical central line. And other graphs would likely be symmetrical diagonally.

Now, we are going to take a look at non-square chessboards. Variables, constraints and lp we need to solve non-square chessboards are the same with the previous situation.

Here is the table shows the result for non-square chessboard.

LP Result of Square Board					
size	num of knight	num of black knight	num of white knight	time	knights / board squares
4*5	14	8	6	0.01s	0.7
4*6	16	8	8	0.01s	0.667
4*7	18	9	9	0.01s	0.623
4*8	20	9	11	0.01s	0.625
4*9	22	12	10	0.06s	0.611
4*10	24	12	12	0.11s	0.6

We can see that with chessboard become larger, the number of total knight we need is increased. And the number of black night sometimes is less than or equal to or greater than the number of white knight, but they have a difference of at most 2. The time consumption stays the same for the first 4 chessboard, and gets larger with the size of the board becomes larger. The ratio of the number of knight to the number of board square is getting smaller as the board size become larger, from 0.7 to 0.6 as the table shows.

My conjecture is, with the size of the chessboard getting larger and larger, the number of knight will increase. The number of black and white knight would stay very close to each other, but we cannot predict with color would be larger. Also, the time consumption would become larger and larger, and time would increase by a greater ratio as the size gets larger. Also, the ratio of the number of knights over the number of board squares would be stabilized around 0.6.

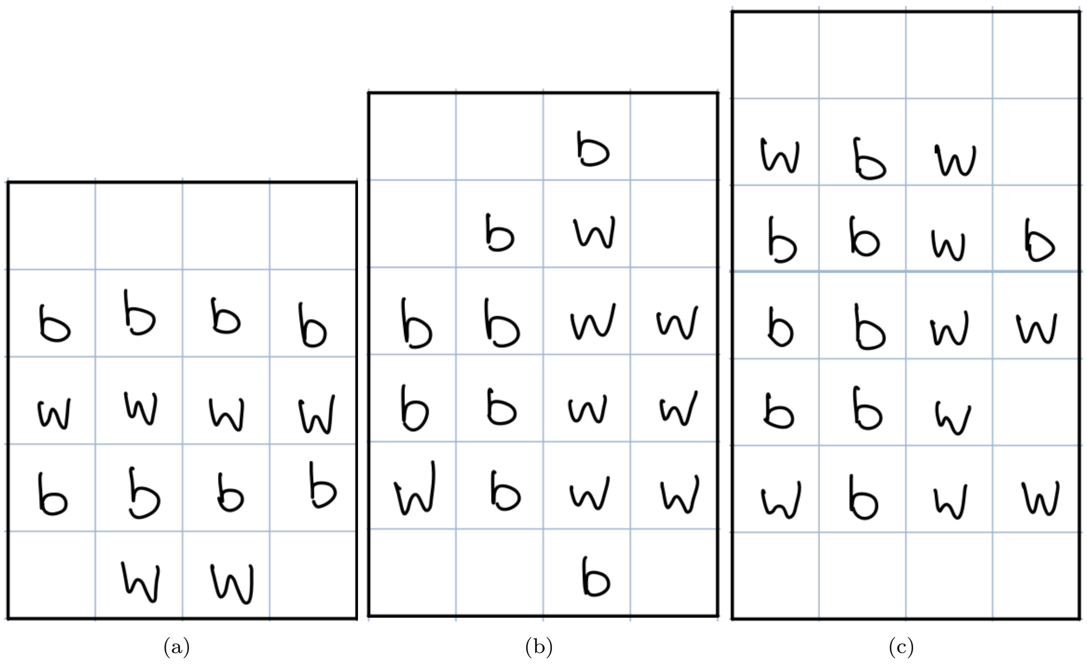


Figure 4: (h) 4*5 (i) 4*6 (j) 4*7

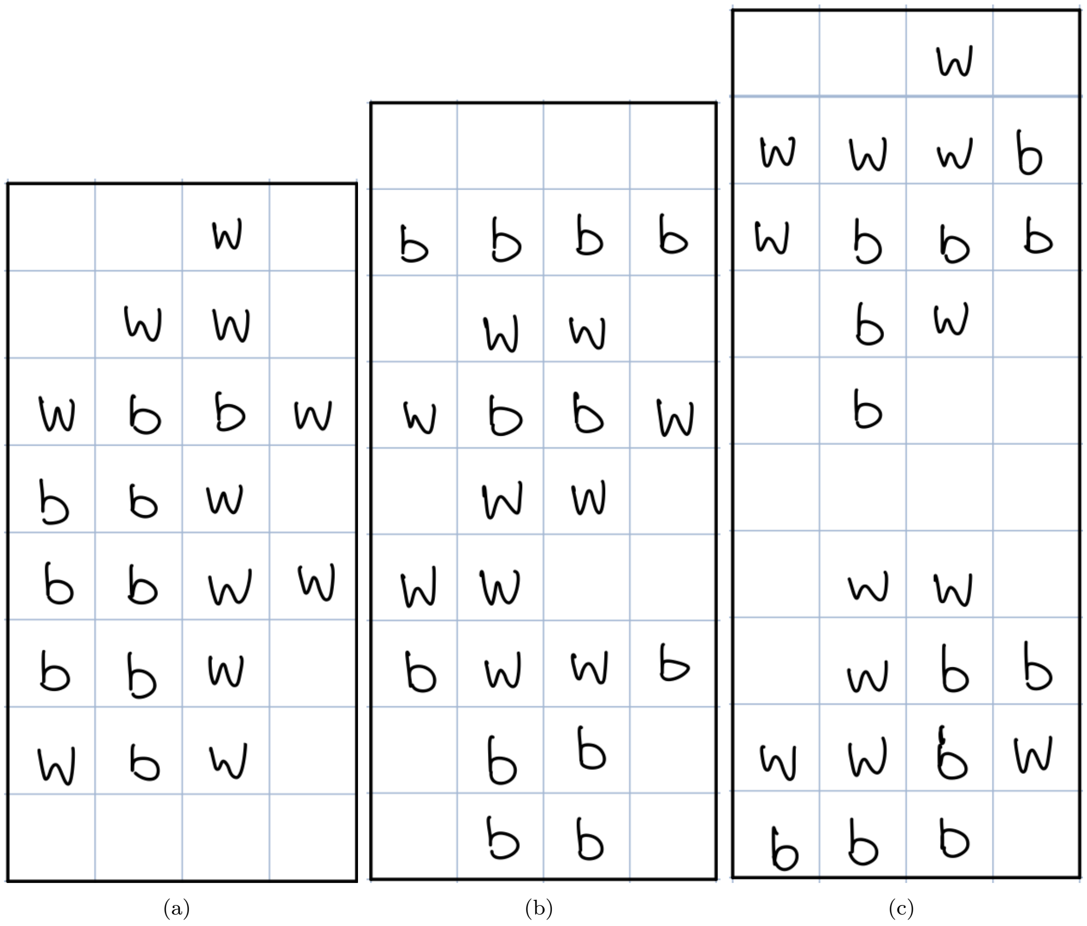


Figure 5: (k) 4*8 (l) 4*9 (m) 4*10

Among the six graphs, unlike square chessboard, only the graph for 4*5 is symmetrical over the vertical central line. Compare to the square chessboard, non-square graphs has less pattern over the alignment of knights. I think with non-square chessboard getting larger and larger, it's hard to see any symmetrical pattern in the graph.

Speeding up the Computation

I try to speed up my lpsolve by changing the order of constrains. In my previous computation, I put constraints of each board square should contain at most one knight after constraints each board should be attack at least once by black and white knights, now I will exchange the order of constraints. The following tables show the time consumption for two different constraint orders.

Time Consumption Before and After Constraint Order Change for Square Board		
size	before	after
4*4	0.01s	0.01s
5*5	0.01s	0.01s
6*6	0.01s	0.01s
7*7	0.01s	0.01
8*8	0.1s	0.21s
9*9	9.05s	55.08s
10*10	489.34s	297.47s

Time Consumption Before and After Constraint Order Change for Non-square Board		
size	before	after
4*5	0.01s	0.01s
4*6	0.01s	0.01s
4*7	0.01s	0.01s
4*8	0.01s	0.02s
4*9	0.06s	0.05s
4*10	0.11s	0.06s

We see that after the order has been changed, the time consumption decreased for those larger chessboard, increased by those medium chessboard and stay the same for those small chessboard. Thus, if we want to use lp to solve two knight domination problem for larger chessboard such as 10*10 or larger, I would recommend put the constrains of each board square should contain at most one knight before constraints each board should be attack at least once by black and white knights. And if we want to calculate relative small board, then the other way around is a faster way.