Knights

Xinming Zhang

10/15/21

Problem Description:

Let's suppose we have three different colors of knights: black, white and green.

Problem to solve: What is the maximal number of knights that we can put on a chess board, such that no knights of different colors attack each other, and there are equal numbers of all three colors (e.g., four black, four white and four green knights)?

Note that: The boards we are saying here are all square chess boards.

Problem solving:

First step is setting up our LP(Linear Programming).

Defining variables:

Set n and m as the number of rows and columns of the board.

Set variable k as the knight.

Set x, y as the number of the rows and columns of the knight.

Set c as the color of the knight, and 1, 2, 3 represent black, white, and green in order.

For example, k_{121} represents the black knight on the first row and the second column on the board.

We set k variables as binary numbers, which means it is either one or zero. One represents that particular knight is on the board, zero means it is not on the board.

From the description above, we know that we are maximizing the number of knights we can put on the board. That will be our objective function. For the constraints, at most one color knight could be put on each space will be the first. That also means on xth row and yth column, there is only one color knight. Then, we want to make sure there are no knights attacking each other. That means there is no different-color knight in every knight's attacking range (note that we are talking about the case in which the knight is on the board). After all that, the amount of every colored knight on the board must agree.

Transform those information to LP, it will be following:

(Objective function: maximizing the number of knights on the board)

$$\text{Maximize: } \sum_{x=1}^{n} \sum_{y=1}^{m} \sum_{c=1}^{3} k_{xyc}$$

(There will be only one or zero knight on each space)

$$\text{ST: } \sum_{c=1}^{3} k_{xyc} \leq 1 \quad \text{,for all } 1 \leq x \leq n, \text{ and } 1 \leq y \leq m$$

(No different-color knight in every knight's attacking range)

$$k_{x_1 y_1 c_1} + k_{x_2 y_2 c_2} \leq 1, \text{under } |(x_1 - x_2)(y_1 - y_2)| = 2 \; and \; c_1 \neq c_2, \text{ for all } 1 \leq x_1, x_2 \leq n,$$

$$1 \leq y_1, y_2 \leq m, \text{ and } 1 \leq c_1, c_2 \leq 3$$

(There are equal numbers of all three colors)

$$\sum_{x=1}^{n} \sum_{y=1}^{m} k_{xy1} = \sum_{x=1}^{n} \sum_{y=1}^{m} k_{xy2}$$

$$\sum_{x=1}^{n} \sum_{y=1}^{m} k_{xy2} = \sum_{x=1}^{n} \sum_{y=1}^{m} k_{xy3}$$

(Make the first knight on the top - left corner is black for a shorter runtime)

$$k_{112} = k_{113} = 0$$

(The number of knight on each space is either one or zero)

$$k_{xyc} \in \{0, 1\} \text{ ,for all } 1 \leq x \leq n, 1 \leq y \leq m, \text{ and } 1 \leq c \leq 3$$

From the LP, we can generate the code to get the input file to lp-solve (Java):
Note that comments are in green, and the code is for 5 by 5 boards. We can change the size of the board by changing the number of n and m in the first two lines.

```
public class M381_2 {
   public static void main(String[] args) {
      //Set n and m as the number of rows and columns of the board.
      //For example: when n=m=5, it will be a 5 by 5 board.
      int n = 5;
      int m = 5;
      //Using a for loop to print the objective function, maximizing the number
      //of color 1 (black) knights to make it run faster on lp-solve
      System.out.print("max:");
      for (int i = 1; i <= n; i++) {
         for (int j = 1; j <= m; j++) {
            //Comment it back if we want to do the number of total knights on the
            //board, the solutions will be the same, but the runtime might differ
            //for (int k = 1; k <= 3; k++)
            System.out.print(" + x_" + i + "_" + j + "_" + 1);
         }
      }
         //Finish the objective function
         System.out.println(";");
         //Using for loop to print the first constraint which every space has only
         //one knight
         for (int x = 1; x <= n; x++) {
```

```
        for (int y = 1; y <= m; y++) {
            for (int c = 1; c <= 3; c++) {
                System.out.print(" + x_" + x + "_" + y + "_" + c);
            }
            System.out.println(" <= 1;");
        }
    }

    //Using for loops and attack method to print the no attack constraint which
    //will ensure that no knights of different colors attack each other
    for (int x = 1; x <= n; x++) {
        for (int y = 1; y <= m; y++) {
            for (int c = 1; c <= 3; c++) {
                for (int i = 1; i <= n; i++) {
                    for (int j = 1; j <= m; j++) {
                        for (int k = 1; k <= 3; k++) {
                            if (attack(x, y, c, i, j, k)) {
                                System.out.print(" + x_" + x + "_" + y + "_" + c);
                                System.out.print(" + x_" + i + "_" + j + "_" + k);
                                System.out.println(" <= 1;");
                            }
                        }
                    }
                }

            }
        }
    }
    //Using for loops to print the 1 knight per space constraint which every color
has
    //the same amount of knights (in following four for loops) The amount of
    //knights in color 1 will be equal to color 2. It's the same with color 2 and 3.
    for (int j = 1; j <= m; j++) {
        for (int i = 1; i <= n; i++) {
            System.out.print(" + x_" + i + "_" + j + "_" + 1);
        }
    }
    System.out.print(" = ");
    for (int j = 1; j <= m; j++) {
        for (int i = 1; i <= n; i++) {
            System.out.print(" + x_" + i + "_" + j + "_" + 2);
        }
    }
    System.out.println(";");

    for (int j = 1; j <= m; j++) {
        for (int i = 1; i <= n; i++) {
            System.out.print(" + x_" + i + "_" + j + "_" + 2);
        }
    }
```

```
            System.out.print(" = ");
            for (int j = 1; j <= m; j++) {
                for (int i = 1; i <= n; i++) {
                    System.out.print(" + x_" + i + "_" + j + "_" + 3);
                }
            }

            //The same amount knights for each color constraint finishes
            System.out.println(";");
           //Making sure the first knight (first row and column) is a black one
            System.out.println("x_1_1_2 = 0;");
            System.out.println("x_1_1_3 = 0;");
            //All the solutions should be either 0 or 1(binary numbers)
            System.out.print("bin");
            for (int i = 1; i <= n; i++) {
                for (int j = 1; j <= m; j++) {
                    for (int k = 1; k <= 3; k++) {
                        if (i == n && j == m && k == 3) {
                            System.out.println(" x_" + i + "_" + j + "_" + k + ";");
                        } else {
                            System.out.print(" x_" + i + "_" + j + "_" + k + ",");
                        }
                    }
                }

            }

        }
        //A method to check if the knights can attack each other, if it return
        //true that means it is good that no knights can attack each other. They can
        //attack each other when it returns false. We need it to be true.
    public static boolean attack(int x, int y, int c, int i, int j, int k) {
        if (Math.abs((x - i) * (y - j)) == 2 && c != k) {
            return true;
        } else {
            return false;
        }
    }
}
```

After running the code, we can get the input file as (size as 5*5):

```
(Objective function: maximizing the number of knights can put on the board)
max: + x_1_1_1 + x_1_2_1 + … +  + x_5_3_1 + x_5_4_1 + x_5_5_1;
(Making sure there is only one knight on each space)
 + x_1_1_1 + x_1_1_2 + x_1_1_3 <= 1;
 + x_1_2_1 + x_1_2_2 + x_1_2_3 <= 1;
… (21 liked lines)
 + x_5_4_1 + x_5_4_2 + x_5_4_3 <= 1;
 + x_5_5_1 + x_5_5_2 + x_5_5_3 <= 1;
(Making sure that no knights of different colors attack each other)
 + x_1_1_1 + x_2_3_2 <= 1;
```

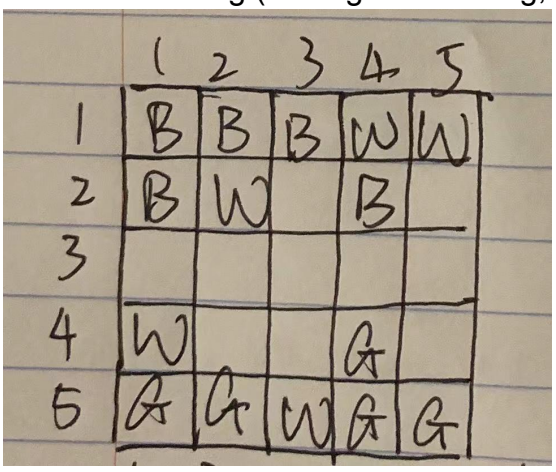
+ x_1_1_1 + x_2_3_3 <= 1;
… (572 liked lines)
+ x_5_5_3 + x_4_3_1 <= 1;
+ x_5_5_3 + x_4_3_2 <= 1;
(Making sure there are equal numbers of all three colors)
+ x_1_1_1 + x_2_1_1 + … + x_5_5_1 = + x_1_1_2 + x_2_1_2 + … + x_5_5_2;
+ x_1_1_2 + x_2_1_2 + … + x_5_5_2 = + x_1_1_3 + x_2_1_3 + … + x_5_5_3;
(All solution are in binary [0 or 1])
bin x_1_1_1, x_1_1_2, x_1_1_3, … , x_5_5_1, x_5_5_2, x_5_5_3;

Output from the input above (for 5*5 board):
Value of objective function: 15.00000000
Actual values of the variables:

x_1_1_1	1
x_1_2_1	1
x_1_3_1	1
x_1_4_2	1
x_1_5_2	1
x_2_1_1	1
x_2_2_2	1
x_2_4_1	1
x_4_1_2	1
x_4_4_3	1
x_5_1_3	1
x_5_2_3	1
x_5_3_2	1
x_5_4_3	1
x_5_5_3	1

All other variables are zeros.

This output is for the 5*5 boards, and it is saying that there can be at most 15 knights we can put on the board. This solution can be express as a picture, and I drew it by hand as following (not a good drawing, but clear to see the position of knights):



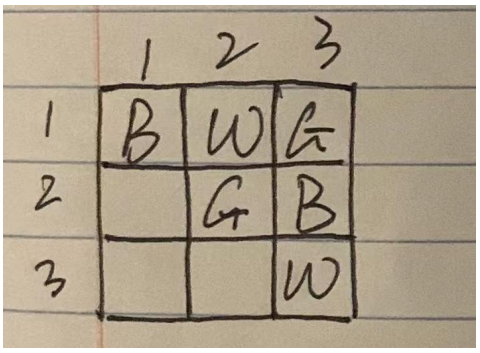
Note that: B, W, and G stands for black, white and green knights in order.

When I was trying to get the solution, it took a long time to get it at first. It took me 367 seconds which is about 6 minutes. It was quite a long time, then after I added a

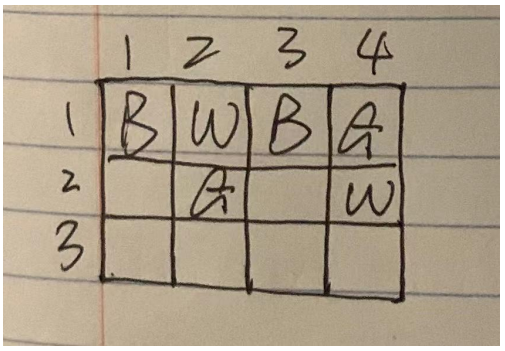
constraint (black knight is on the top- left corner) and changed the objective function to maximize the number of only black knights to LP and code. Then the runtime got a lot faster to 19.366 seconds. It was an amazing difference, then I used this code to solve the larger board problem.

I drew the picture to show the result of each board as following:

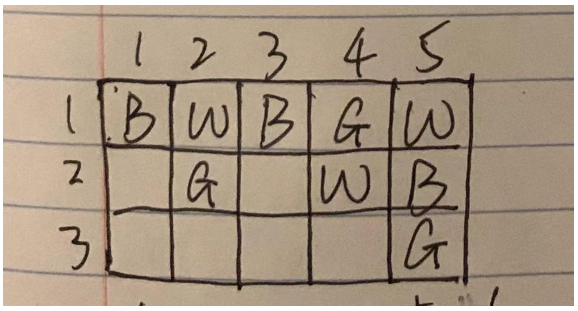
Picture of 3*3 board:



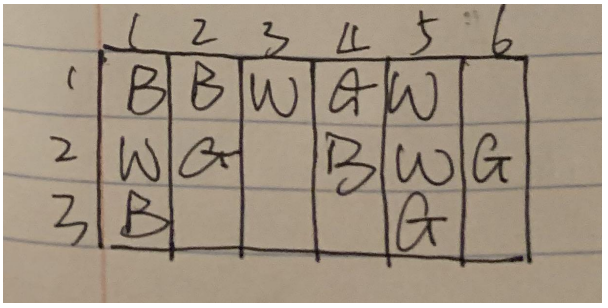
Picture of 3*4 board:



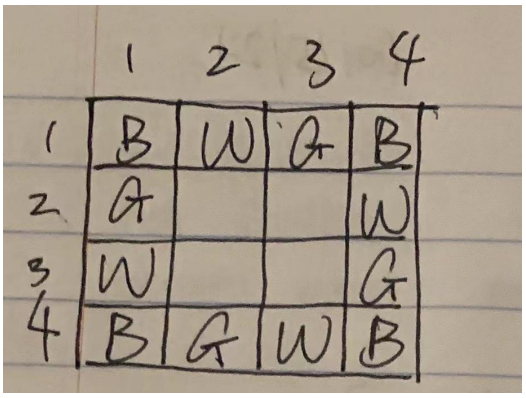
Picture of 3*5 board:



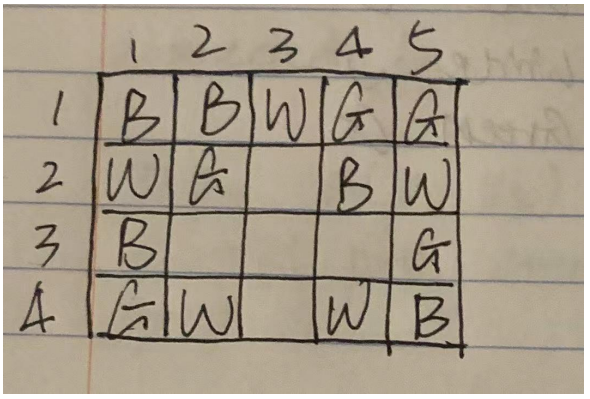
Picture of 3*6 board:



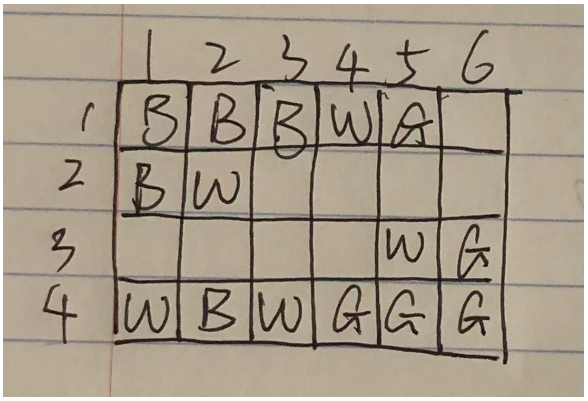
Picture of 4*4 board:



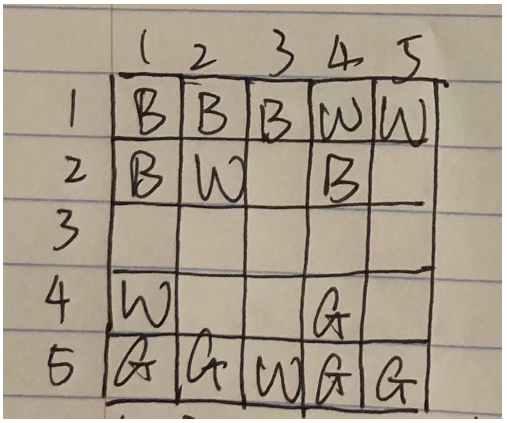
Picture of 4*5 board:



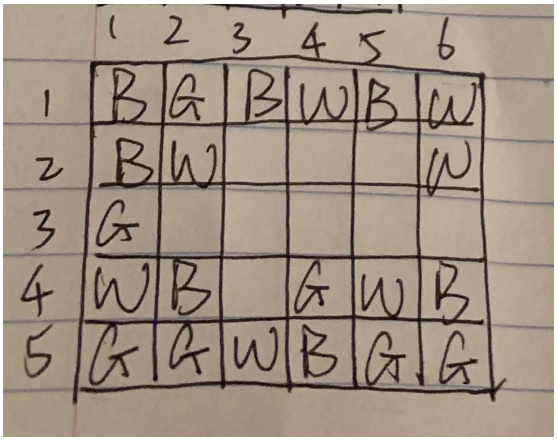
Picture of 4*6 board:



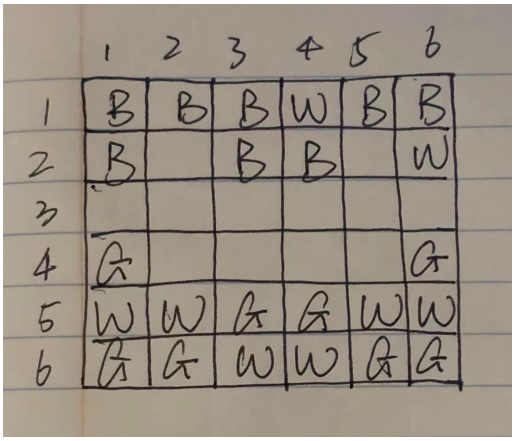
Picture of 5*5 board:



Picture of 5*6 board:



Picture of 6*6 board:



Solutions in table (the number of total knights can put on the board):

	3 columns	4 columns	5 columns	6 columns
3 rows	6 knights	6 knights	9 knights	12 knights
4 rows		12 knights	15 knights	15 knights
5 rows			15 knights	21 knights
6 rows				24 knights

Lp-solve runtime for each board (shortest from my data):

	3 columns	4 columns	5 columns	6 columns
3 rows	0.26 seconds	0.354 seconds	1 seconds	1.7 seconds
4 rows		0.309 seconds	0.7 seconds	41 seconds
5 rows			19 seconds	14 seconds
6 rows				605 seconds

Summary: From the first table, we can know the number of the knights can put on the board mainly depends on the number of the rows and columns (size). One interesting finding is there is no number 18 in the table. I thought the number would increase three by three, because there are three colors of knights. The number increased from 15 to 21 without going through 18. The second interesting thing is 4*5, 4*6, 5*5 tables all have the same number of knights on the board. I did not think about that.
Form the drawing for 3*3 and 4*4, they look pretty. The picture for 3*3 looks like a triangle, and the picture for 4*4 is a square, and it leaves a small square inside as well.
From the second table which is about runtime. Runtime depends on the size of the board, but there are some interesting cases. The runtime for 5*6 board is actually shorter than 5*5 and 4*6 boards. Also for 4*4 and 4*5 board, the runtime is extremely shorter than other ones. We can say the runtime does not only depend on the board size, and it also depends on the shape of the board.
I tried to solve the 7*7 boards, but it seems like it is taking forever. I waited for an hour, but it did not give an answer. There must be somewhere I could improve my LP to make the runtime shorter, and I would like to spend more to know about it.