# Assignment 2

MATH 381 A - Winter 2022

Yixuan Liu

January 18, 2022

**Introduction**

For this assignment, we will solve a knight's domination problem. Suppose we have a chess board, and we are trying to put black and white knights on it. Our goal is to minimize the total number of knights that are placed on the board such that every square on the board can be attacked by at least one white knight and at least one black knight.
To solve this problem with an LP, we define a binary variable $x_{ijk}$ that indicates whether the $i^{th}$ row $j^{th}$ column square on the chess board should be placed with a $k^{th}$ color knight, which $k = 0$ is for white and $k = 1$ is for black. When $x_{ijk}$ equals to 1, a knight with the $k^{th}$ color is placed on the $i^{th}$ row $j^{th}$ column square.
Suppose we have a $n \times m$ chess board with $n$ rows and $m$ columns. For every square $(i, j)$ where $0 \le i \le n - 1$ and $0 \le j \le m - 1$ is defined as on board. We want to minimize the total number of knights on the board, which can be expressed as:

$$\text{Minimize } \sum_{k=0}^{1} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} x_{ijk}$$

and we will have constraints for each square on the chess board to make sure that every square is attacked by at least one white knight and at least one black knight. We know that for a knight on a chess board, square $(i, j)$ can only make moves to eight positions

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

that are on the chess board. For example, for the first square on the upper left of the chess board $(0, 0)$, if we want to make sure that at least one white knight and at least one black knight can attack this square, we must have the following constraint:

$$x_{1,2,0} + x_{1,2,1} + x_{2,1,0} + x_{2,1,1} \ge 1$$

since positions $(-1, 2)$, $(-1, -2)$, $(-2, 1)$, and $(-2, -1)$ are not exist on the board. We define the positions a knight can move to to be

$$M_{ij} = S_{ij} \cap \{(a, b) : 0 \le a \le n - 1, 0 \le b \le m - 1, a, b \in \mathbb{Z}_{\ge 0}\}$$

We can summrize this constraint as

$$\sum_{(a,b)\in M_{ij}} x_{ab0} \geq 1 \text{ for all } 0 \leq i \leq n-1,\ 0 \leq j \leq m-1$$
$$\sum_{(a,b)\in M_{ij}} x_{ab1} \geq 1 \text{ for all } 0 \leq i \leq n-1,\ 0 \leq j \leq m-1$$

to make sure that for all the squares on board can be attacked by white knight for at least one time and by black knight for at least one time.
We also want to make sure that on each board there is at most one knight:

$$x_{ij0} + x_{ij1} \leq 1 \text{ for all } 0 \leq i \leq n-1,\ 0 \leq j \leq m-1$$

Last, we want to keep all variables binary:

$$x_{ijk} \in \{0,1\} \text{ for all } 0 \leq i \leq n-1,\ 0 \leq j \leq m-1$$

So, our LP should be:

$$\begin{aligned}
\text{Minimize } & \sum_{i=0}^{n-1}\sum_{j=0}^{m-1} x_{ijk} \\
\text{subject to } & \sum_{(a,b)\in M_{ij}} x_{ab0} \geq 1 \\
& \sum_{(a,b)\in M_{ij}} x_{ab1} \geq 1 \\
& x_{ij0} + x_{ij1} \leq 1 \\
& x_{ijk} \in \{0,1\} \text{ for all } 0 \leq i \leq n-1,\ 0 \leq j \leq m-1
\end{aligned}$$

**1**

We want to start from square chess boards, where $n = m$. We will start from a $4 \times 4$ board and then increase the number to bigger boards. To solve this LP, we will use lpsolve. Here is the code written in Java to generate an input file for lpsolve for a $5 \times 5$ square board:

```java
// This code will generate an lpsolve-formatted LP which solves how are the
   different colors of knights should be placed on a 5 x 5 square board
import java.util.*;
import java.io.*;

public class A2 {
  // set the rows and columns of the chess board and types of colors
  public static final int COLORS = 2;
  public static final int n = 5;
  public static final int m = 5;
  // define possible moves a knight can make on the chess board
  public static final int[][] move = {
    {1, 2},
    {1, -2},
    {-1, 2},
    {-1, -2},
    {2, 1},
    {2, -1},
    {-2, 1},
    {-2, -1},
};

public static void main(String[] args) throws FileNotFoundException {
    PrintStream input = new PrintStream(new File("a2_input.txt"));

    // generate lpsolve input file
    // print objective function
    input.print("min: ");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < COLORS; k++) {
                input.print("+x_" + i + "_" + j + "_" + k);
            }
        }
    }
    input.println(";");

    // print constriants
    // each position must be attacked by at least one knight of each color
    for (int k = 0; k < COLORS; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                for (int p = 0; p < 8; p++) {
                    int[][] possiblePosition = makeMove(p);
                    int x = i + possiblePosition[0][0];
                    int y = j + possiblePosition[0][1];
                    if (onboard(x, y)) {
                        input.print("+x_" + x + "_" + y + "_" + k);
                    }
                }
                input.println(">=1;");
            }
        }
    }

    // each position can only have at most one knight
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < COLORS; k++) {
                input.print("+x_" + i + "_" + j + "_" + k);
            }
            input.println("<=1;");
        }
        }

        // define variables to be bianary
        input.print("bin ");
        for (int i = 0; i < n; i++) {
           for (int j = 0; j < m; j++){
              for (int k = 0; k < COLORS; k++) {
                 if ((i == (n-1)) && (j == (m-1)) && (k == (COLORS-1))) {
                    input.println("x_" + i + "_" + j + "_" + k + ";");
                 } else {
                    input.print("x_" + i + "_" + j + "_" + k + ",");
                 }
              }
           }
        }
     }

     // a method to check whether a position is on the board
     public static boolean onboard(int i, int j) {
        return (i >= 0 && j >= 0 && i < n && j < m);
     }

     // a method to make one single posiible position a given square can be
         attacked
     public static int[][] makeMove(int p) {
        int[][] newPosition = new int[1][2];
        newPosition[0][0] = move[p][0];
        newPosition[0][1] = move[p][1];
        return newPosition;
     }
  }
```

We set a 5 × 5 chess board and make sure that each square on the board can be attacked by at least one white knight and at least one black knight, and each square can have at most one knight. After printing out the objective function and all the constraints, we get the following output file:

```
min: +x_0_0_0+...,+x_4_4_1;
(50 lines of the following type:
ensure that every square is attacked at least by
one white knight and at least by one black knight)
+x_1_2_0+x_2_1_0>=1;
.
.
+x_3_2_1+x_2_3_1>=1;
(25 linew of the following type:
ensure that every square has at most one knight)
+x_0_0_0+x_0_0_1<=1;
```

```
.
.
+x_4_4_0+x_4_4_1<=1;
bin x_0_0_0,...,x_4_4_1;
```

Then run the lpsolve with the following command that only presenting non-zero variables and also show the time for lpsolve to solve this LP:

```
lp_solve -ia -time a2_input.txt
```

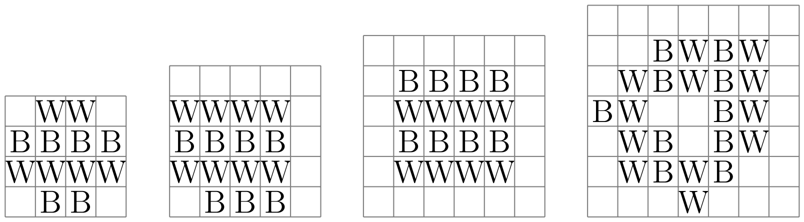
and get the output:

```
CPU Time for Parsing input: 0.002s (0.002s total since program start)
CPU Time for solving: 0.006s (0.008s total since program start)

Value of objective function: 15.00000000

Actual values of the variables:
x_0_1_1                          1
x_0_2_1                          1
x_0_3_1                          1
x_1_0_0                          1
x_1_1_0                          1
x_1_2_0                          1
x_1_3_0                          1
x_2_0_1                          1
x_2_1_1                          1
x_2_2_1                          1
x_2_3_1                          1
x_3_0_0                          1
x_3_1_0                          1
x_3_2_0                          1
x_3_3_0                          1
```

From the output we can find that to solve this LP, the lpsolve took 0.008 seconds, and for the solution of the LP, there should be 15 knights in total, 8 white knights and 7 black knights. Repeating the same procedure for a $4 \times 4$ square board and increasing the size of board by increasing the value of $n$ an $m$ in the java code, we will get the following result for each board up to a $10 \times 10$ square board:



4×4 5×5 6×6 7×7

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | B | B | B | B | | |
| | W | W | W | W | W | W | |
| | B | B | | | B | B | |
| | W | W | | | W | W | |
| | B | B | B | B | B | B | |
| | | W | W | W | W | | |
| | | | | | | | |

8×8

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | W | | |
| | W | B | W | W | | W | | |
| | W | B | B | | B | B | B | W |
| | W | W | B | | B | W | | |
| | B | | | | | | B | |
| | | W | B | | B | W | W | |
| W | B | B | B | | B | B | W | |
| | | W | | W | W | B | W | |
| | | W | | | | | | |

9×9

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | B | | |
| | W | B | W | B | B | | B | | |
| | W | B | W | | | W | W | W | B |
| | W | B | B | | | W | B | | |
| | W | | | | W | | | W | |
| | B | | | B | | | | B | |
| | | W | | | | | W | B | |
| W | B | B | B | | | B | B | B | W |
| | | W | | W | W | B | W | | |
| B | | W | | | | | W | | |

10×10

For the first 6 square boards, the lpsolve took less than a minute to solve the LP, but for the $10 \times 10$ board, it took 308.273 seconds to solve it, so we want to rearrange the order of the constraints presenting in the input file to see that whether it will speed up the solving time or not. By printing the constraints to ensure each position can only have at most one knight before printing the constraints to ensure each position must be attacked by at least one knight of each color, the lpsolve took 421.73 seconds to solve the new LP. It slows down the computation, but provide a alternative solution with the same amount of knights in different colors:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | B | | |
| | W | B | W | B | B | W | B | | |
| | W | B | W | | | W | W | W | B |
| | W | B | B | | | | B | W | |
| | W | | | | | | | W | |
| | B | | | | | | | B | |
| | B | W | | | | | W | B | |
| W | B | | B | | | B | B | B | W |
| | B | W | B | W | W | B | W | | |
| | | W | | | | | W | | |

10×10

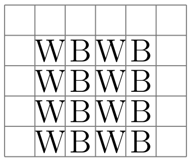
The white knights on (5,5) and (6,6) in the prvious solution are moved to (8,6) and (6,8) in the new solution, and the black knights on (0,0), (2,2), and (4,4) are moved to (1,1), (1,3), and (3,1).We might include that for lpsolve to compute faster, it's better to put the most restrictive constraints early in the input file, but the solution might also be changed.
When the lpsolve trying to solve the $11 \times 11$ board, it took more than an hour, so we might stop at here and only discuss the result for the previous 7 boards. Here is a table comparing the number of knights in different colors and computational time between different size of square boards:

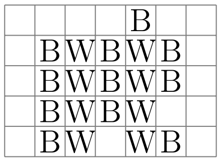
| Size | Total number of all knights | White knights | Black knights | Computational time | Number of knights/number of board squares |
|---|---|---|---|---|---|
| $4 \times 4$ | 12 | 6 | 6 | 0.006s | 0.75 |
| $5 \times 5$ | 15 | 8 | 7 | 0.008s | 0.6 |
| $6 \times 6$ | 16 | 8 | 8 | 0.01s | 0.44 |
| $7 \times 7$ | 22 | 12 | 10 | 0.011s | 0.45 |
| $8 \times 8$ | 28 | 14 | 14 | 0.185s | 0.4375 |
| $9 \times 9$ | 38 | 20 | 18 | 46.774s | 0.47 |
| $10 \times 10$ | 44 | 22 | 22 | 308.273s | 0.44 |

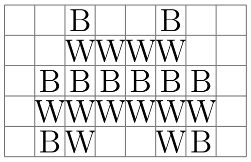
We can see that the ratio of number of knights to number of board squares is decreasing at first, but floating around 0.4 as the board size getting bigger. There is a interesting note that for all the square boards that has odd rows and columns, the number of white knights and black knights are not the same. It might due to that these square boards do not have a symmetrical features to layout the same number of black knights and white knights.

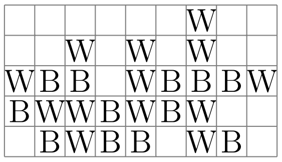
## 2

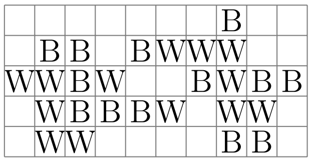
From now, we will try to solve the same knight's domination problem but with non-square boards, which means that $n$ and $m$ are not equal anymore. We will use the same LP and the same Java code from the previous one except change the value of $n$ and $m$ in the code to generate new lpsolve input file. From the previous part we can find that for $5 \times 5$ square board, the total number of all knights is odd, which could due to the odd squares on the board, but what about non-square boards with 5 rows but different amount of columns starting from 6? Here is the drawing of non-square boards with appropriate positions of knights up to a $5 \times 16$ board, since the $5 \times 17$ board took more than an hour to be solved:

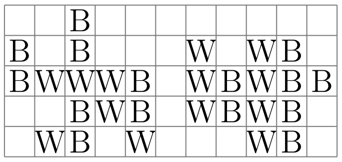

5×6

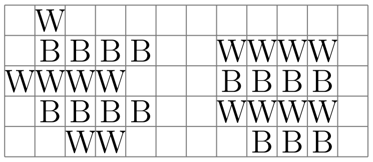

5×7

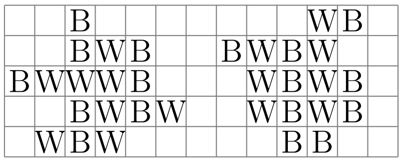

5×8

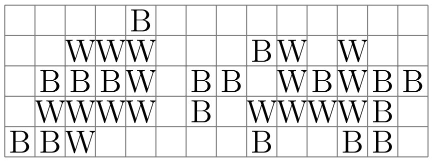

5×9

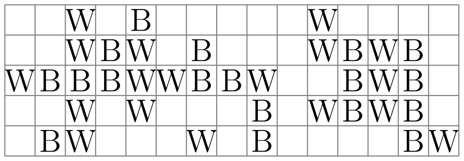

5×10

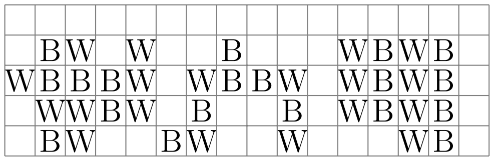

5×11


5×12


5×13


5×14


5×15


5×16

Here is a table comparing the number of knights in different colors and computational time between different size of square boards:

| Size | Total number of all knights | White knights | Black knights | Computational time | Number of knights/number of board squares |
|---|---|---|---|---|---|
| $5 \times 6$ | 16 | 8 | 8 | 0.015s | 0.53 |
| $5 \times 7$ | 19 | 8 | 11 | 0.035s | 0.54 |
| $5 \times 8$ | 22 | 12 | 10 | 0.038s | 0.55 |
| $5 \times 9$ | 25 | 13 | 12 | 0.251s | 0.56 |
| $5 \times 10$ | 26 | 13 | 13 | 0.36s | 0.52 |
| $5 \times 11$ | 28 | 13 | 15 | 0.852s | 0.51 |
| $5 \times 12$ | 30 | 15 | 15 | 3.247s | 0.5 |
| $5 \times 13$ | 32 | 15 | 17 | 111.759s | 0.49 |
| $5 \times 14$ | 34 | 17 | 17 | 40.253s | 0.49 |
| $5 \times 15$ | 36 | 18 | 18 | 31.549s | 0.48 |
| $5 \times 16$ | 38 | 19 | 19 | 311.348s | 0.475 |

We can see from the table that for $5 \times 6$ board, there are even number of knights in total, and odd number of knights for $5 \times 7$ and $5 \times 9$ boards. However, starting from $5 \times 10$ board, the total number of all knights is always even. One possible reason for this pattern might be that for relatively small boards with odd number of squares, its asymmetry makes the number of knigts to be odd; but when the board is getting big enough, the asymmetry does not matter anymore. The ratio of number of knights to number of board squares is slightly increasing till the $5 \times 9$ board and then decreasing. Since starting from $5 \times 10$ board, each time the board increased with 5 more squares, only two knights are added, we may assume that there would finally be a limit to the ratio of number of knights to number of board squares.