Math 381 A
Sheng Yu

Homework #2 Writeup: Equally Attacking Knights

I. Introduction:

On the chess board, a knight always moves in one direction for two blocks then the other direction for one block. The path it goes form this "L" shape pattern. The chessboard is a $n \times n$ square with n^2 blocks. Below we will be using Linear Programming to test that for different n values, what will be the maximal number of knights that we can place on the chessboard such that each placed knight is attacking exactly m knights. Depending on size of n, $1 \leq m \leq 8$.

II. Linear Programming:

(a) $n = 4, m = 1$.

For this case, we will have 16 blocks which looks like this:

(1, 4)	(2, 4)	(3, 4)	(4, 4)
(1, 3)	(2, 3)	(3, 3)	(4, 3)
(1, 2)	(2, 2)	(3, 2)	(4, 2)
(1,1)	(2, 1)	(3, 1)	(4, 1)

We mark the blocks with $(i, j),\ 1 \leq i \leq 4, 1 \leq j \leq 4$.

Let $x_{i,j}$ be a binary variable that is equal to one if and only if we should place a knight on the (i, j) block. Since we try to maximize the total number of knights on the chessboard, we can get the object function:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j}$$

Before I define the constraints, it would be helpful to know the following three information:

1. Let $Z_{i,j}$ denotes the number of blocks that can attack block (i, j) if a knight is there.

2. For block (i, j) with a knight, then up to 8 possible blocks can attack it with index:

$$(i+1, j+2), (i+1, j-2), (i-1, j+2), (i-1, j-2)$$

$$(i+2, j+1), (i-2, j+1), (i+2, j-1), (i-2, j-1)$$

3. For all possible blocks in 2., blocks only exist if both indexes of that block in range from $1\ to\ n$. Because it's impossible to have a block that is out of the chessboard. We can use this condition to get the $Z_{i,j}$ value for block (i, j).

For example, for block $(1, 1)$, if a knight is placed in there, there are two locations where the block can be attacked. They are $(2, 3)$ & $(3, 2)$. Here, $Z_{1,1} = 2$, and we want:

$$If\ x_{1,1} = 1, then\ x_{2,3} + x_{3,2} = m$$

$$If\ x_{1,1} = 0, then\ x_{2,3} + x_{3,2}\ can\ be\ any\ value\ (no\ constraint)$$

We can write this as:

$$x_{2,3} + x_{3,2} \geq m \cdot\ x_{1,1}$$

$$x_{2,3} + x_{3,2} \leq x_{1,1} \cdot (m-2) + 2$$

We can check that if $x_{1,1} = 1$, then we get $x_{2,3} + x_{3,2} \geq m$ and $x_{2,3} + x_{3,2} \leq m$, which will force $x_{2,3} + x_{3,2} = m$. And if $x_{1,1} = 0$, we get $x_{2,3} + x_{3,2} \geq 0$ and $x_{2,3} + x_{3,2} \leq 2$, this basically puts no constraints on $x_{2,3}, x_{3,2}$ as they are binary variables.

Another example, for block $(1, 2)$, three blocks can reach it if a knight is on them. They are $(2, 4), (3, 3)$ & $(3, 1)$. Here, $Z_{1,2} = 3$, and we get:

$$x_{2,4} + x_{3,3} + x_{3,1} \geq m \cdot\ x_{1,2}$$

$$x_{2,4} + x_{3,3} + x_{3,1} \leq x_{1,2} \cdot (m-3) + 3$$

With same reasoning, we can see that the two constraints are correct.

One more example before we can see a pattern here, let's use block $(3, 2)$, then we can see that blocks $(4, 4), (2, 4), (1, 3)$ & $(1, 1)$ can attack it and $Z_{3,2} = 4$. We can get:

$$x_{4,4} + x_{2,4} + x_{1,3} + x_{1,1} \geq m \cdot x_{3,2}$$

$$x_{4,4} + x_{2,4} + x_{1,3} + x_{1,1} \leq x_{3,2} \cdot (m - 4) + 4$$

We can see a pattern with the second constraint in this constraints pair for block (i, j):

$$X1 + X2 + \cdots \leq x_{i,j} \cdot \left(m - Z_{i,j}\right) + Z_{i,j},$$

$$where\ X1, X2 \ldots are\ binary\ variables\ represent\ the\ blocks\ that\ can\ attack\ (i, j)$$

Since each block should have two constraints, for a $n \times n$ chessboard, we should have $2n^2$ constraints in total, and here is a summary of the Linear Programming:

$Maximize$:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{i,j}$$

$Subject\ to$:

$$X1 + X2 + \cdots \geq m \cdot x_{i,j} \quad for\ all\ 1 \leq i \leq n, 1 \leq j \leq n$$

$$X1 + X2 + \cdots \leq x_{i,j} \cdot \left(m - Z_{i,j}\right) + Z_{i,j}, \quad for\ all\ 1 \leq i \leq n, 1 \leq j \leq n$$

$$x_{i,j} \in \{0, 1\}, where\ 1 \leq i \leq n, 1 \leq j \leq n$$

Now, we can write the following code in MATLAB to generate the LPSolve input file:

```
% List n, m

n = 4;
m = 1;

% Set up

Z = zeros(n, n);                    % For given n, generate a matrix with Z_i_j values
coeff1 = zeros(n^2, n^2);           % Generate the first constraint coefficient
                                    % for the constraints pair for block (i, j).

for i = 1:n
   vec = zeros(n, n^2);
   for j = 1:n
      count = 0;                    % initialize with 0 block attacking (i, j)
      temp = zeros(n, n);           % try to find all locations that attacks (i, j).
      temp(i, j) = -m;
      if i + 1 <= n                 % test if (i+1,j+2), (i+1, j-2) exists
         if j + 2 <= n
            count = count + 1;
            temp(i+1, j+2) = 1;
         end
         if j - 2 >= 1
            count = count + 1;
            temp(i+1, j-2) = 1;
         end
      end
      if i - 1 >= 1                 % test if (i-1, j+2), (i-1, j-2) exists
         if j + 2 <= n
            count = count + 1;
            temp(i-1, j+2) = 1;
         end
         if j - 2 >= 1
            count = count + 1;
            temp(i-1, j-2) = 1;
         end
      end
      if j + 1 <= n                 % test if (i+2, j+1), (i-2, j+1) exists
         if i + 2 <= n
            count = count + 1;
            temp(i+2, j+1) = 1;
         end
         if i - 2 >= 1
            count = count + 1;
            temp(i-2, j+1) = 1;
         end
      end
      if j - 1 >= 1                 % test if (i+2, j-1), (i-2, j-1) exists
         if i + 2 <= n
            count = count + 1;
            temp(i+2, j-1) = 1;
         end
         if i - 2 >= 1
            count = count + 1;
            temp(i-2, j-1) = 1;
         end
      end
      Z(i, j) = count;
      temp = reshape(temp', [1, n^2]);
      vec(j,:) = temp;
   end
   coeff1(n*i-(n-1):n*i, :) = vec;
end

% Generate the second constraint coefficient for the constraints pair for block (i, j).

coeff2 = coeff1;
upper = reshape(Z', [1, n^2]);  % This is the Z_i_j on the right hand side
d = upper - m;
coeff2 = coeff2 - diag(diag(coeff2)) + diag(d);
```

```
% Generate the lpsolve input file

lp = mxlpsolve('make_lp', 0, n^2);

mxlpsolve('set_maxim', lp);                                    % Maximize
mxlpsolve('set_binary', lp, (1:n^2));                          % Set xi to be 0 or 1
mxlpsolve('set_obj_fn', lp, ones(1, n^2));                     % Set object fun

% Add constraints and name variables

for k = 1:n^2
    co1 = coeff1(k, :);
    co2 = coeff2(k, :);
    mxlpsolve('add_constraint', lp, co1, 2, 0);
    mxlpsolve('add_constraint', lp, co2, 1, upper(k));
end

% Name x_ij

start = 1;
for i = 1:n
    for j = 1:n
        names{start} = strcat('x_', num2str(i), '_', num2str(j));
        start = start + 1;
    end
end

for i = 1:n^2
    mxlpsolve('set_col_name', lp, i, names{i});
end

% Write lpsolve input file

mxlpsolve('write_lp', lp, strcat('hw2_n', num2str(n), 'm', num2str(m), '.lp'));
```

For different n & m, we can simply change their value in the beginning of the code. By running the script above, we can generate the following LPSolve input file, and get the output file:

```
/* Objective function */
max: +x_1_1 +x_1_2 +x_1_3 +x_1_4 +x_2_1 +x_2_2 +x_2_3 +x_2_4 +x_3_1 +x_3_2 +x_3_3 +x_3_4 +x_4_1
+x_4_2 +x_4_3 +x_4_4;

/* Constraints */
-x_1_1 +x_2_3 +x_3_2 >= 0;
+x_1_1 +x_2_3 +x_3_2 <= 2;
-x_1_2 +x_2_4 +x_3_1 +x_3_3 >= 0;
+2 x_1_2 +x_2_4 +x_3_1 +x_3_3 <= 3;
-x_1_3 +x_2_1 +x_3_2 +x_3_4 >= 0;
+2 x_1_3 +x_2_1 +x_3_2 +x_3_4 <= 3;
-x_1_4 +x_2_2 +x_3_3 >= 0;
+x_1_4 +x_2_2 +x_3_3 <= 2;
+x_1_3 -x_2_1 +x_3_3 +x_4_2 >= 0;
+x_1_3 +2 x_2_1 +x_3_3 +x_4_2 <= 3;
+x_1_4 -x_2_2 +x_3_4 +x_4_1 +x_4_3 >= 0;
+x_1_4 +3 x_2_2 +x_3_4 +x_4_1 +x_4_3 <= 4;
+x_1_1 -x_2_3 +x_3_1 +x_4_2 +x_4_4 >= 0;
+x_1_1 +3 x_2_3 +x_3_1 +x_4_2 +x_4_4 <= 4;
+x_1_2 -x_2_4 +x_3_2 +x_4_3 >= 0;
+x_1_2 +2 x_2_4 +x_3_2 +x_4_3 <= 3;
+x_1_2 +x_2_3 -x_3_1 +x_4_3 >= 0;
+x_1_2 +x_2_3 +2 x_3_1 +x_4_3 <= 3;
+x_1_1 +x_1_3 +x_2_4 -x_3_2 +x_4_4 >= 0;
+x_1_1 +x_1_3 +x_2_4 +3 x_3_2 +x_4_4 <= 4;
+x_1_2 +x_1_4 +x_2_1 -x_3_3 +x_4_1 >= 0;
+x_1_2 +x_1_4 +x_2_1 +3 x_3_3 +x_4_1 <= 4;
+x_1_3 +x_2_2 -x_3_4 +x_4_2 >= 0;
+x_1_3 +x_2_2 +2 x_3_4 +x_4_2 <= 3;
+x_2_2 +x_3_3 -x_4_1 >= 0;
+x_2_2 +x_3_3 +x_4_1 <= 2;
+x_2_1 +x_2_3 +x_3_4 -x_4_2 >= 0;
+x_2_1 +x_2_3 +x_3_4 +2 x_4_2 <= 3;
+x_2_2 +x_2_4 +x_3_1 -x_4_3 >= 0;
+x_2_2 +x_2_4 +x_3_1 +2 x_4_3 <= 3;
+x_2_3 +x_3_2 -x_4_4 >= 0;
+x_2_3 +x_3_2 +x_4_4 <= 2;

/* Variable bounds */
bin
x_1_1,x_1_2,x_1_3,x_1_4,x_2_1,x_2_2,x_2_3,x_2_4,x_3_1,x_3_2,x_3_3,x_3_4,x_4_1,x_4_2,x_4_3,x_4_4;
```

```
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
```

We can interpret from the output file that for a 4×4 chessboard, if we want to have exactly 1 knight attaching each other, then there should be at most 8 knights in total on the board to fulfill this requirement. Their positions are:

$$(1,1), (1,2), (1,3), (1,4), (2,1), (2,2), (2,3), (2,4)$$

(b) For different n value, $m = 1$.

Here is a table to summarize the information:

$\underline{m = 1}$:

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	8	0.013
5	10	0.276
6	16	9.141
7	20	578.62
8	32	232.47
9	\	Stopped the run at 624.78

Before we continue with $m = 2, 3, ...$, one thing to notice that when $n = 7$, the time of computation was 578.62s which is very close to 10 minutes (600s), and we want to see if it can be reduced. So, we will implement some constraints to speed up the computation.

III. Improvement:

We can assume that at least half of the knights on the board are on the left side of the board, and if n is odd, the center blocks count as the left side of the board. It's much easier to demonstrate with a graph. For a 5×5 chessboard:

(1, 5)	(2, 5)	(3, 5)	(4, 5)	(5, 5)
(1, 4)	(2, 4)	(3, 4)	(4, 4)	(5, 4)
(1, 3)	(2, 3)	(3, 3)	(4, 3)	(5, 3)
(1, 2)	(2, 2)	(3, 2)	(4, 2)	(5, 2)
(1, 1)	(2, 1)	(3, 1)	(4, 1)	(5, 1)

What we want for this constraint is that:

$$\sum_{i=1}^{3}\sum_{j=1}^{5} x_{i,j} \geq \frac{1}{2} \cdot \sum_{i=1}^{5}\sum_{j=1}^{5} x_{i,j}$$

So, in general we can write this as:

$$2 \cdot \sum_{i=1}^{\frac{n}{2}}\sum_{j=1}^{n} x_{i,j} \geq \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j}\,, if\ n\ is\ an\ even\ number$$

$$2 \cdot \sum_{i=1}^{floor(\frac{n}{2})+1}\sum_{j=1}^{n} x_{i,j} \geq \sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j}\,, if\ n\ is\ a\ odd\ number$$

We add the following to our original MATLAB code above to implement this new constraint:

```
% Get coefficient for the new constraint for speed up the computation

reminder = mod(n, 2);       % See if n is an even number of odd number
coeff3 = ones(1, n^2);
if reminder == 0            % Case for n is even
   v = ones(1, n^2/2);
   v(1, (n^2/2)+1:n^2) = 0;
   coeff3 = coeff3 - 2.*v;
end
if reminder == 1            % Case for n is odd
   f = floor(n/2);
   v = ones(1, (f+1)*n);
   v(1, (f+1)*n+1:n^2) = 0;
   coeff3 = coeff3 - 2.*v;
end

% New constraint

mxlpsolve('add_constraint', lp, coeff3, 1, 0);
```

With this, for $m = 1$, we get:

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	8	0.013
5	10	0.266 (was 0.276)
6	16	6.708 (was 9.141)
7	20	Stopped at 656.57 (was 578.62)
8	32	27.68 (was 232.47)
9	\	Stopped the run at 614.48

We can see that after implementing this constraint, when $n = 7$, we went from 578.62s up to 656.57s without finishing the computation. But, for case $n = 8$, we went from 232.47s down to 27.68s (204.8s improvement). Also, for other n values we can see that is an overall decrease in the computation time. Even though that for the case $n = 7$ we did not finish the run under 10 minutes, the benefit for speed up 204.8s for $n = 8$ can compensate with this. Since we want the LP to be able to solve for large n values fast. In the next section, we will have a summary for different n, m values with this new constraint.

IV. Summaries of Different n, m Values:

For different n, m_{max} are different. We can simply know what m_{max} can be by finding the maximal value in $Z_{i,j}$ since it contains all the possible m values. Here we can have a summary for m_{max} for different n values:

n	m_{max}
4	4
5	8
6	8
For all $n \geq 7$	8

We have already shown the case for $m = 1$ in the previous section, so we simply copy it here.

For $m = 1$:

For $n = 7$, we can use what we got before implement the new constraint for less than 10 mins.

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	8	0.013
5	10	0.266
6	16	6.708
7	20	Stopped at 656.57
8	32	27.68
9	\	Stopped the run at 614.48

For $m = 2$:

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	10	0.013
5	16	0.034
6	20	0.139
7	24	44.02
8	32	393.32
9	\	Stopped the run at 605.31

$\underline{For\ m = 3:}$

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	0	0.01
5	0	0.02
6	16	0.072
7	20	2.307
8	32	37.90
9	\	Stopped the run at 639.39

$\underline{For\ m = 4:}$

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
4	0	0.00
5	0	0.014
6	0	0.016
7	16	0.025
8	16	0.059
9	16	0.359
10	16	155.15
11	\	Stopped the run at 611.39

$\underline{For\ m = 5:}$

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
5	0	0.01
6	0	0.01
7	0	0.01
8	0	0.02
9	0	0.03
10	0	0.06
11	0	0.12

For $m = 6$:

Chessboard Size ($n\ value$)	Maximal Number of knights	Time of Computation (s)
5	0	0.01
6	0	0.01
7	0	0.01
8	0	0.01
9	0	0.01
10	0	0.03
11	0	0.07

I went ahead and tested the case where $m = 7, 8$, we get the same table as for $m = 5, 6$. So, for case $m = 5, 6, 7, 8$, no matter how large the n value is, it is hardly to have any knights on the board attaching each other exactly m times.

V. Diagrams:

In this section, we will draw the solutions for $m = 1, 2, 3, 4$ with the biggest n we get above:

For $m = 1, n = 8, 32\ knights$:

K	K	K	K			K	K
K	K	K	K			K	K
						K	K
						K	K
K	K						
K	K						
K	K			K	K	K	K
K	K			K	K	K	K

For $m = 2, n = 8, 32$ knights:

K	K	K			K	K	K
K		K			K		K
K	K	K			K	K	K
K	K	K			K	K	K
K		K			K		K
K	K	K			K	K	K

For $m = 3, n = 8, 32$ knights:

		K	K	K	K		
	K		K	K		K	
	K	K	K	K	K	K	
K							K
K							K
	K	K	K	K	K	K	
	K		K	K		K	
		K	K	K	K		

For $m = 4, n = 10, 16$ knights:

			K						
	K				K				
		K	K	K					
K		K		K		K			
		K	K	K					
	K				K				
			K						