Assignment 2

Christian Diangco

Math 381 A

A group of chess players want to solve a chess-related problem.

They want to know what the maximal number of knights that can be placed on an *n* by *n* chessboard such that each knight is attacking exactly *m* knights?

We can solve this problem using an LP.

We will make a variable that corresponds to row *i* column *j* on the chess-board.

can be either 1 or 0, where 1 means there is a knight in position (*i*, *j*)and 0 means there is not.

So the objective function we are trying to maximize is

If a position is occupied by a knight, we want *m* knights to be able to attack that position.

If a position is not occupied by a knight, the number of knights able to attack that position does not matter.

To accomplish this, each position (*i, j*) on the board will need two constraints.

One for the upper bound and one for the lower bound of the number of knights that can or need to attack each position.

The lower-bound will be

When there is a knight in position (*i, j*)*,* (= 1), then the lower bound will be *m* because we want this knight to be able to attack *m* other knights at positions *a, b*.

When there is not a knight in position (*i, j*)*,* (= 0), then the lower bound will be 0 because we want to put no constraints on the number of knights able to attack (*i, j*).

The upper bound will be

When there is a knight in position (*i, j*)*,* (= 1), then the upper bound will be *m* because we want this knight to be able to attack *m* other knights at positions *a, b*.

Notice that the lower bound is also *m* when = 1, making the number of knights at positions *a, b* have to be m.

When there is not a knight in position (*i, j*)*,* (= 0), then the upper bound will be 8 because, as stated previously, we want to put no constraints on the number of knights able to attack (*i, j*).

Additionally, we chose the number 8 because that is the max number of knights that can attack a single position on the board.

To generate the LPSolve input file for *n* = 4 and *m* = 1, we will use this Python code:

# This is the input file for LPSolve.

f = open("Assignment2.txt", "w")

# For an nxn chess board, we want to find the maximal number of knights such that each knight is attacking exactly

# m knights.

n = 4

m = 1

# This string contains the function we are trying to maximize i.e. the number of knights.

obj\_function = "max: "

# This string contains each of the binary variables.

variables = "bin "

# These strings contain lower-bound and upper-bound constraints, respectively.

constraint1, constraint2 = "", ""

# This loop creates each of the lines for the LP input file.

for i in range(1, n + 1):

for j in range(1, n + 1):

variables += "x\_" + str(i) + "\_" + str(j) + ","

obj\_function += "+" + "x\_" + str(i) + "\_" + str(j)

# top half of possible attack positions

if i - 1 >= 1:

if j - 2 >= 1:

constraint1 += "+x\_" + str(i - 1) + "\_" + str(j - 2)

constraint2 += "+x\_" + str(i - 1) + "\_" + str(j - 2)

if j + 2 <= n:

constraint1 += "+x\_" + str(i - 1) + "\_" + str(j + 2)

constraint2 += "+x\_" + str(i - 1) + "\_" + str(j + 2)

if i - 2 >= 1:

if j - 1 >= 1:

constraint1 += "+x\_" + str(i - 2) + "\_" + str(j - 1)

constraint2 += "+x\_" + str(i - 2) + "\_" + str(j - 1)

if j + 1 <= n:

constraint1 += "+x\_" + str(i - 2) + "\_" + str(j + 1)

constraint2 += "+x\_" + str(i - 2) + "\_" + str(j + 1)

# bottom half of possible attack positions

if i + 1 <= n:

if j - 2 >= 1:

constraint1 += "+x\_" + str(i + 1) + "\_" + str(j - 2)

constraint2 += "+x\_" + str(i + 1) + "\_" + str(j - 2)

if j + 2 <= n:

constraint1 += "+x\_" + str(i + 1) + "\_" + str(j + 2)

constraint2 += "+x\_" + str(i + 1) + "\_" + str(j + 2)

if i + 2 <= n:

if j - 1 >= 1:

constraint1 += "+x\_" + str(i + 2) + "\_" + str(j - 1)

constraint2 += "+x\_" + str(i + 2) + "\_" + str(j - 1)

if j + 1 <= n:

constraint1 += "+x\_" + str(i + 2) + "\_" + str(j + 1)

constraint2 += "+x\_" + str(i + 2) + "\_" + str(j + 1)

constraint1 += " <= " + str(m - 8) + "x\_" + str(i) + "\_" + str(j) + "+8" + ";\n"

constraint2 += " >= " + str(m) + "x\_" + str(i) + "\_" + str(j) + ";\n"

variables = variables[:-1] + ";"

obj\_function += ";\n"

f.writelines([obj\_function, constraint1, constraint2, variables])

f.close()

The generated LPSolve input file from this code looks like this:

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;

+x\_2\_3+x\_3\_2 <= -7x\_1\_1+8;

+x\_2\_4+x\_3\_1+x\_3\_3 <= -7x\_1\_2+8;

+x\_2\_1+x\_3\_2+x\_3\_4 <= -7x\_1\_3+8;

+x\_2\_2+x\_3\_3 <= -7x\_1\_4+8;

+x\_1\_3+x\_3\_3+x\_4\_2 <= -7x\_2\_1+8;

+x\_1\_4+x\_3\_4+x\_4\_1+x\_4\_3 <= -7x\_2\_2+8;

+x\_1\_1+x\_3\_1+x\_4\_2+x\_4\_4 <= -7x\_2\_3+8;

+x\_1\_2+x\_3\_2+x\_4\_3 <= -7x\_2\_4+8;

+x\_2\_3+x\_1\_2+x\_4\_3 <= -7x\_3\_1+8;

+x\_2\_4+x\_1\_1+x\_1\_3+x\_4\_4 <= -7x\_3\_2+8;

+x\_2\_1+x\_1\_2+x\_1\_4+x\_4\_1 <= -7x\_3\_3+8;

+x\_2\_2+x\_1\_3+x\_4\_2 <= -7x\_3\_4+8;

+x\_3\_3+x\_2\_2 <= -7x\_4\_1+8;

+x\_3\_4+x\_2\_1+x\_2\_3 <= -7x\_4\_2+8;

+x\_3\_1+x\_2\_2+x\_2\_4 <= -7x\_4\_3+8;

+x\_3\_2+x\_2\_3 <= -7x\_4\_4+8;

+x\_2\_3+x\_3\_2 >= 1x\_1\_1;

+x\_2\_4+x\_3\_1+x\_3\_3 >= 1x\_1\_2;

+x\_2\_1+x\_3\_2+x\_3\_4 >= 1x\_1\_3;

+x\_2\_2+x\_3\_3 >= 1x\_1\_4;

+x\_1\_3+x\_3\_3+x\_4\_2 >= 1x\_2\_1;

+x\_1\_4+x\_3\_4+x\_4\_1+x\_4\_3 >= 1x\_2\_2;

+x\_1\_1+x\_3\_1+x\_4\_2+x\_4\_4 >= 1x\_2\_3;

+x\_1\_2+x\_3\_2+x\_4\_3 >= 1x\_2\_4;

+x\_2\_3+x\_1\_2+x\_4\_3 >= 1x\_3\_1;

+x\_2\_4+x\_1\_1+x\_1\_3+x\_4\_4 >= 1x\_3\_2;

+x\_2\_1+x\_1\_2+x\_1\_4+x\_4\_1 >= 1x\_3\_3;

+x\_2\_2+x\_1\_3+x\_4\_2 >= 1x\_3\_4;

+x\_3\_3+x\_2\_2 >= 1x\_4\_1;

+x\_3\_4+x\_2\_1+x\_2\_3 >= 1x\_4\_2;

+x\_3\_1+x\_2\_2+x\_2\_4 >= 1x\_4\_3;

+x\_3\_2+x\_2\_3 >= 1x\_4\_4;

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;

After running LPSolve with this input file, we find that the maximal number of knights where *n* = 1 and *m* = 2 is equal to 8.

We can solve this problem for multiple values of *n* and *m* by simply changing the values assigned to *n* and *m* in the Python code and running LPSolve with the generated input file.

The following tables show the maximal number of knights and computation times for and increasing values of *n*.

For , there is no solution.

For *m* = 1, I computed the maximal number of knights and computation times for .

I attempted to compute the maximal number of knights for *n* = 6 but the computation time was past 10 minutes, so its computation was terminated.

***m* = 1**

| Board Size: *n* x *n* | Maximal Number of Knights | Computation Time (s) |
| --- | --- | --- |
| 4 x 4 | 8 | 3.089 |
| 5 x 5 | 10 | 482.683 |

Here is the solution for a 5 x 5 chess board where each knight must be able to attack 1 other knight.

The maximal number of knights for this case is 10.

|  | K | K | K |  |
| --- | --- | --- | --- | --- |
| K |  |  | K |  |
|  |  |  |  | K |
| K |  |  | K |  |
|  | K | K |  |  |

For *m* = 2, I computed the maximal number of knights and computation times for .

I attempted to compute the maximal number of knights for *n* = 7 but the computation time was past 10 minutes, so its computation was terminated.

***m* = 2**

| Board Size: *n* x *n* | Maximal Number of Knights | Computation Time (s) |
| --- | --- | --- |
| 4 x 4 | 10 | 1.018 |
| 5 x 5 | 16 | 23.468 |
| 6 x 6 | 20 | 123.553 |

Here is the solution for a 6 x 6 chess board where each knight must be able to attack 2 other knights.

The maximal number of knights for this case is 20.

|  |  | K | K | K |  |
| --- | --- | --- | --- | --- | --- |
|  | K | K |  | K | K |
| K | K |  |  |  | K |
| K |  |  |  | K | K |
| K | K |  | K | K |  |
|  | K | K | K |  |  |

For *m* = 3, I computed the maximal number of knights and computation times for .

I attempted to compute the maximal number of knights for *n* = 8 but the computation time was past 10 minutes, so its computation was terminated.

***m* = 3**

| Board Size: *n* x *n* | Maximal Number of Knights | Computation Time (s) |
| --- | --- | --- |
| 4 x 4 | 0 | 0.340 |
| 5 x 5 | 0 | 6.414 |
| 6 x 6 | 16 | 22.286 |
| 7 x 7 | 20 | 423.082 |

Here is the solution for a 7 x 7 chess board where each knight must be able to attack 3 other knights.

The maximal number of knights for this case is 20.

|  | K |  |  | K |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | K | K |  |  |  |
| K | K | K | K | K | K |  |
|  |  |  |  |  |  |  |
| K | K | K | K | K | K |  |
|  |  | K | K |  |  |  |
|  | K |  |  | K |  |  |

For *m* = 4, I computed the maximal number of knights and computation times for .

I started computing at *n* = 6 because we already know that the maximal number of knights for *n* = 5, *m* = 3 is 0.

I attempted to compute the maximal number of knights for *n* = 10 but the computation time was past 10 minutes, so its computation was terminated.

***m* = 4**

| Board Size: *n* x *n* | Maximal Number of Knights | Computation Time (s) |
| --- | --- | --- |
| 6 x 6 | 0 | 2.928 |
| 7 x 7 | 16 | 3.909 |
| 8 x 8 | 16 | 13.795 |
| 9 x 9 | 16 | 91.938 |

Here is the solution for a 9 x 9 chess board where each knight must be able to attack 4 other knights.

The maximal number of knights for this case is 16.

|  |  |  | K |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | K |  |  |  | K |  |  |  |
|  |  | K | K | K |  |  |  |  |
| K |  | K |  | K |  | K |  |  |
|  |  | K | K | K |  |  |  |  |
|  | K |  |  |  | K |  |  |  |
|  |  |  | K |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |