Total Domination Problem:

A Knight on the Rim is Dim, But a Knight on the Corner is Forlorner

MATH 381 HW 2

Tim Ferrin, 1764309

October 16, 2020

I Background Information

The class of problems known as domination problems involve filling a chessboard with pieces such that every square is attacked. The goal is to find the minimum number of pieces and an arrangement of those pieces that accomplishes this requirement. The three pieces of interest for us can attack squares in the following ways:

1. Queens can attack any square that is in its row or column and any square that is along either of its diagonals.
2. Bishops can attack any square that is along either of its diagonals.
3. Knights can attack any square that is 1 away vertically and 2 away horizontally, or is 1 away horizontally and 2 away vertically.

The 8×8 grids directly below visually demonstrate the squares that each piece can attack, where a knight is represented by a **K**, a queen is represented by a **Q**, a bishop is represented by a **B**, and any spot they can attack is represented with an X.

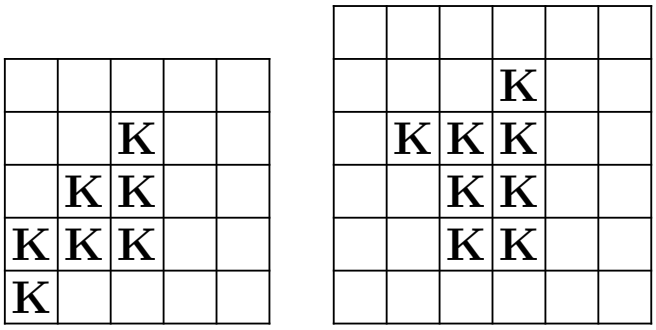
	X		X				
X				X			
		K					
X				X			
	X		X				

		X			X		
			X		X		X
				X	X	X	
X	X	X	X	X	**Q**	X	X
				X	X	X	
			X		X		X
		X			X		
	X				X		

							X
X						X	
	X				X		
		X		X			
			B				
		X		X			

8×8 boards showing the squares that a knight, queen, and bishop can attack.

The grids below give examples of 5×5 and 6×6 boards which are totally dominated only with knights.



Examples of knight total domination boards with minimum number of knights needed in the 5×5 and 6×6 cases.

The particular domination problem with which we are concerned is using one and only one queen, one and only one bishop, and the minimum number of knights necessary to totally dominate the board. This means that all squares should be attacked, even ones where the pieces are residing. We are also attempting to generalize this to not just an 8x8 board, but any board of size $n \times n$. The only limitation that we run into is the computational power/speed of the computer used to solve this problem.

II Mathematical Model

In order to solve the general problem described above, we will transform its verbal description into that of a linear program and allow our computer to do the computation for us. First off, for a board of size $n \times n$ we will establish n^2 binary variables $k_{i,j}$ which is 1 if knight is sitting on the square in row i and column j or a 0 if no knight is sitting on the aforementioned square, for any particular arrangement. Likewise, we will establish similar variables $q_{i,j}$ for queens and $b_{i,j}$ for bishops. Squares can be referenced with their coordinate pairs (i, j)

Since we are trying to find the minimum number of knights we need to dominate each size board with one queen and one bishop, our objective function is just the sum of all $k_{i,j}$ variables:

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

The first set of constraints arises from the fact that on any individual square there can be at most one of the three pieces sitting. Stated mathematically, this means that

$$k_{i,j} + q_{i,j} + b_{i,j} \leq 1 \qquad \text{for all } 1 \leq i, j \leq n \tag{2}$$

The second set of constraints arises form the requirement that every space must be attacked

by at least one piece. As this is unique for each square given the nature of the different attacking capabilities of the pieces, we will give the general mathematical formulation:

$$\sum_{\substack{(i,j):\\ \text{knight at } (i,j)\\ \text{attacks } (a,b)}} k_{i,j} + \sum_{\substack{(i,j):\\ \text{queen at } (i,j)\\ \text{attacks } (a,b)}} q_{i,j} + \sum_{\substack{(i,j):\\ \text{bishop at } (i,j)\\ \text{attacks } (a,b)}} b_{i,j} \geq 1 \qquad \text{for all } 1 \leq a, b \leq n \tag{3}$$

The final two constraints arise from the fact that there must be one queen and one bishop on every board. Stated mathematically, these constraints are:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} q_{i,j} = 1 \tag{4}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{i,j} = 1 \tag{5}$$

III Linear Programs

As described above, the linear program that needs to be solved for a board of size $n \times n$ is:

Minimize

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

subject to

$$k_{i,j} + q_{i,j} + b_{i,j} \leq 1 \qquad \text{for all } 1 \leq i, j \leq n$$

$$\sum_{\substack{(i,j):\\ \text{knight at } (i,j)\\ \text{attacks } (a,b)}} k_{i,j} + \sum_{\substack{(i,j):\\ \text{queen at } (i,j)\\ \text{attacks } (a,b)}} q_{i,j} + \sum_{\substack{(i,j):\\ \text{bishop at } (i,j)\\ \text{attacks } (a,b)}} b_{i,j} \geq 1 \qquad \text{for all } 1 \leq a, b \leq n$$

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

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

Where $k_{i,j}, q_{i,j}, b_{i,j} \in \{0, 1\}$ for all $1 \leq i, j \leq n$

IV Code

The code to generate the general $n \times n$ size board linear programs was written in MATLAB. This linear program was written to an .lp file which actually solves the LPs. The commented MATLAB code is written below.

```
for n=2:20
    % Next line is important, as there needs to be a new index matrix per
    % board size
    clear Is
    % Creates an nxn index matrix
    for i=1:n
        for j=1:n
            if n<10
                Is(i,j)=i*10+j;
            else
                Is(i,j)=i*100+j;
            end
        end
    end

    fileID = fopen(['Chess',num2str(n),'.lp'], 'w');
    % Objective function
    fprintf(fileID, 'min: ');
    for i=1:n
        for j=1:n
            if (i~=n)||(j~=n)
                fprintf(fileID, ['k_',num2str(i),'_',num2str(j),' + ']);
            else
                fprintf(fileID, ['k_',num2str(n),'_',num2str(n),';\n']);
            end
        end
    end

    % Can't be more than one piece per square requirement
    for i=1:n
        for j=1:n
            fprintf(fileID, ['q_',num2str(i),'_',num2str(j),' + ','b_',num2str(i),'_',nu
        end
    end

    % Each square must be attacked requirement
    for i=1:n
        for j=1:n
            % Specifies the queens that might attack a square in its column
```

```
for k=find(1:n~=i)
    fprintf(fileID, ['q_',num2str(k),'_',num2str(j),' + ']);
end
% Specifies the queens that might attack a square in its row
for m=find(1:n~=j)
    fprintf(fileID, ['q_',num2str(i),'_',num2str(m),' + ']);
end
% Specifies the queens and bishops that might attack a square diagonally
mainDiag = ismember(Is,diag(Is,j-i));
offDiag = flipud(ismember(flipud(Is),diag(flipud(Is),i+j-n-1)));
diagLocs = Is(1==mainDiag+offDiag);
% **The next 6 lines of code are repeated later but used here for
% formatting the printing
if n<10
    potKLocs = (i+[-2 -1 1 2  2 1 -1 -2])*10+(j+[1 2 2 1 -1 -2 -2 -1]);
else
    potKLocs = (i+[-2 -1 1 2  2 1 -1 -2])*100+(j+[1 2 2 1 -1 -2 -2 -1]);
end
kLocs = potKLocs((i+[-2 -1 1 2  2 1 -1 -2])>0&(j+[1 2 2 1 -1 -2 -2 -1])>0&(i
for p=1:length(diagLocs)
    specLoc=diagLocs(p);
    if n<10
        specRow=floor(specLoc/10);
        specCol=mod(specLoc,10);
    else
        specRow=floor(specLoc/100);
        specCol=mod(specLoc,100);
    end
    fprintf(fileID, ['q_',num2str(specRow),'_',num2str(specCol),' + ']);
    if (length(kLocs)==0)&&(p==length(diagLocs))
        fprintf(fileID, ['b_',num2str(specRow),'_',num2str(specCol)]);
    else
        fprintf(fileID, ['b_',num2str(specRow),'_',num2str(specCol),' + ']);
    end
end
% Specifies the knights that might attack a square
if n<10
    potKLocs = (i+[-2 -1 1 2  2 1 -1 -2])*10+(j+[1 2 2 1 -1 -2 -2 -1]);
else
    potKLocs = (i+[-2 -1 1 2  2 1 -1 -2])*100+(j+[1 2 2 1 -1 -2 -2 -1]);
end
kLocs = potKLocs((i+[-2 -1 1 2  2 1 -1 -2])>0&(j+[1 2 2 1 -1 -2 -2 -1])>0&(i
if length(kLocs)>1
    for q=1:(length(kLocs)-1)
        specLoc=kLocs(q);
```

```
                if n<10
                    specRow=floor(specLoc/10);
                    specCol=mod(specLoc,10);
                else
                    specRow=floor(specLoc/100);
                    specCol=mod(specLoc,100);
                end
                fprintf(fileID, ['k_',num2str(specRow),'_',num2str(specCol),' + ']);
            end
            specLoc=kLocs(end);
            if n<10
                specRow=floor(specLoc/10);
                specCol=mod(specLoc,10);
            else
                specRow=floor(specLoc/100);
                specCol=mod(specLoc,100);
            end
            fprintf(fileID, ['k_',num2str(specRow),'_',num2str(specCol),' >= 1;\n'])
        elseif length(kLocs)==1
            specLoc=kLocs(end);
            if n<10
                specRow=floor(specLoc/10);
                specCol=mod(specLoc,10);
            else
                specRow=floor(specLoc/100);
                specCol=mod(specLoc,100);
            end
            fprintf(fileID, ['k_',num2str(specRow),'_',num2str(specCol),' >= 1;\n'])
        else
            fprintf(fileID, ' >= 1;\n');
        end
    end
end

% There must be 1 queen requirement
for i=1:n
    for j=1:n
        if (i~=n)||(j~=n)
            fprintf(fileID, ['q_',num2str(i),'_',num2str(j),' + ']);
        else
            fprintf(fileID, ['q_',num2str(n),'_',num2str(n),' = 1;\n']);
        end
    end
end
```

```
    % There must be 1 bishop requirement
    for i=1:n
        for j=1:n
            if (i~=n)||(j~=n)
                fprintf(fileID, ['b_',num2str(i),'_',num2str(j),' + ']);
            else
                fprintf(fileID, ['b_',num2str(n),'_',num2str(n),' = 1;\n']);
            end
        end
    end

    % Specifies the piece indicator variables must be binary
    fprintf(fileID, 'bin ');
    for i=1:n
        for j=1:n
            fprintf(fileID, ['q_',num2str(i),'_',num2str(j),', ']);
            fprintf(fileID, ['b_',num2str(i),'_',num2str(j),', ']);
            if (i~=n)||(j~=n)
                fprintf(fileID, ['k_',num2str(i),'_',num2str(j),', ']);
            else
                fprintf(fileID, ['k_',num2str(n),'_',num2str(n),';']);
            end
        end
    end
    fclose(fileID);

end
```

V LPSolve Input File

The following is the generic LPSolve input file for a board of size 6×6

```
min: k_1_1 + k_1_2 +...+ k_6_6;
(36 lines of the following type:
ensure that there is at most one piece on every square of the board)
q_1_1 + b_1_1 + k_1_1 <= 1;
.
.
q_6_6 + b_6_6 + k_6_6 <= 1;
(36 lines of the following type:
ensure that each square is being attacked by at least 1 piece)
q_2_1 + q_3_1 + q_4_1 +...+ b_2_2 + b_3_3 +...+ k_2_3 + k_3_2 >= 1;
.
```

```
.
(ensure there is one queen)
q_1_1 + q_1_2 + q_1_3 +...+ q_6_6 = 1;
(ensure there is one bishop)
b_1_1 + b_1_2 + b_1_3 +...+ b_6_6 = 1;
(ensure all variables are binary)
bin q_1_1, b_1_1, k_1_1, q_1_2, b_1_2, k_1_2,...,q_6_6, b_6_6, k_6_6;
```

VI LPSolve Output and Results

The following is the output for the 6×6 LP:

```
Value of objective function: 4.00000000

k_2_3                          1
k_3_3                          1
k_4_4                          1
k_4_5                          1
b_2_4                          1
q_5_2                          1
```

All other variables are 0.

This corresponds to a solution board of

		K	**B**		
		K			
			K	**K**	
	Q				

Note that this is not necessarily a unique solution to the 6×6 problem, and indeed quarter, half, and three-quarter rotations of the board are also solutions. There may be still other board positionings, but 4 is the minimum number of nights required with one queen and one bishop. The computer used to solve this is a Dell Inspiron 5570 and it took 0.40 seconds to find the solution.

Below is a table recording the minimum number of knights required and the time it took the computer to solve the LP for each board size.

Board Size	Min. Number of Knights	Computation Time (sec)
2×2	0	0.01
3×3	0	0.06
4×4	2	0.06
5×5	3	0.23
6×6	4	0.4
7×7	5	0.62
8×8	8	8.14
9×9	11	147.84

After being able to solve the 9×9 board relatively quickly, the computer was unable to solve the size 10 board within 8 hours. At this point the program was terminated. Computation time could be reduced if a further condition is added, such that the queen must be in the upper left quadrant of the board. This is allowable since solutions are rotationally symmetric and could significantly cut down computation time. Further inquiry could be beneficial.