Brian Kang

MATH 381

January 25, 2019

Writing Assignment 3

|  |
| --- |
| Problem 2. (My right-most digit is 8)  Define a graph G = (V, E) as follows.  Let V = {1, 2, 3, . . . , 12}. (n = number of vertices = 12)  Define E = {(i, j) : i, j ∈ V, i 6= j, i2 + j 2 + 18 is prime }.  Create and solve (using lpsolve) an IP to find the chromatic number of G, χ(G) |

Above is the question I must solve. We want to find the chromatic number of the graph *G*, which is the smallest number of colors we need to color the whole graph. Note that by the definition of graph coloring, two vertices are assigned different colors if they are connected by an edge. Because we want the smallest number of colors to be used that will be our objective function to be minimized. Let us define our variables first,

|  |
| --- |
|  |

Now our objective function and our constraints of our IP,

|  |
| --- |
|  |

We want to find the chromatic number of our graph i.e., the smallest number of colors to use. Then our attention should be on minimizing the sum of the number of colors to be used. Constraint (1) speeds up our IP solving process. It says that if we do not use the earlier color then we cannot use the following color i.e., we cannot use color 2 without using 1 and we cannot use color 3 if we have not used 2 yet and so on. Constraint (2) says that all vertices have only one color. It is saying that from all the vertices available, only one will be assigned a certain color, which explains the entire sum of our 12 terms being equal to 1. It also implies the fact that the largest chromatic number obtainable is equal to the number of total vertices, meaning every vertex is assigned a unique color different from all other colors. Constraint (3) says that if we color a vertex *i* with color *k*, then we are for sure using color *k*. When we know *x­ik=1*, then *yk* must *=1*, meaning that we are using color *k*. The final constraint (4) is critical to our problem. It says that for every edge in *E*, both vertices at the end of the edges cannot have the same color; therefore, either *xik* is *=1* or *xjk=1*. Creating the IP based on this objective function and constraints will solve for the chromatic number of my graph *G*.

I used Java to generate the LPSolve input text file because typing all constraints with the massive number of variables will take a long time by hand. The Java code I wrote is,

|  |
| --- |
| */\*  \* Brian Kang  \* 1/22/2019  \* MATH 381  \* HW #3  \*  \* This program will create the input file to solve an IP to find  \* the chromatic number of a defined graph G, X(G).  \*  \* Define graph G = (V, E) as follows.  \* Let V = {1, 2, 3, . . . , 12}.  \* Define E = {(i, j) : i, j ∈ V, i != j, i^2 + j^2 + 18 is prime}.  \*/* **import** java.io.\*; **import** java.io.FileNotFoundException; **import** java.util.\*;  **public class** hw3 {  **public static final** String ***FILE\_NAME*** = **"math381hw3\_2.txt"**; *// output file name* **public static** List<Integer> *vertices*; *// list of vertices* **public static** List<String> *objvars*; *// list of objective function variables* **public static** List<String> *constvars*; *// list of additional constraint variables* **public static void** main(String[] args) **throws** FileNotFoundException{  *// outputting into text file* PrintStream output = **new** PrintStream(**new** File(***FILE\_NAME***));   *// define V = {1,...,12}  vertices* = **new** ArrayList<Integer>();  *// highest chromatic number = number of vertices  // objective function variables: y\_k, k in {1,...,12} and y is binary  objvars* = **new** ArrayList<String>();  **for** (**int** k = 1; k <= 12; k++) {  *vertices*.add(k);  *objvars*.add(**"y\_"**+k);  }   *// define all constraint variables  // x\_i\_k are binary, i&k in {1,...,12}  constvars* = **new** ArrayList<String>();  **for** (**int** i = 1; i <= 12; i++) {  **for** (**int** k = 1; k <= 12; k++) {  *constvars*.add(**"x\_"**+i+**"\_"**+k);  }  }   *// output objective function  // we want to minimize the the number of colors we will use* output.println(**"/\* Objective function \*/"**);  output.print(**"min: "**);  **for** (**int** i=1; i <= *objvars*.size(); i++) {  output.print(**"+"**+*objvars*.get(i-1));  }  output.println(**";\n"**);   *// keep count of number of constraints* **int** count = 1;  *// start outputting the constraints* output.println(**"/\* Variable bounds \*/"**);   *// first constraint  // speeds up computing  // if we don't use the next color, we don't use the earlier one first  // yk <= yk-1, for k = 2,...,12* **for** (**int** k = 2; k <= 12; k++) {  output.print(**"c"**+count+**": "**);  output.print(*objvars*.get(k-1)+**" <= "**+*objvars*.get(k-2)+**";"**);  count++;  output.println();  }   *// second constraint  // all vertices have one color  // sum from k=1 to 12 of xik =1, where i = 1,...,12* **for** (**int** i = 1; i <= 12; i++) {  output.print(**"c"**+count+**": "**);  output.print(*constvars*.subList(12\*(i-1),12\*i).toString()  .replace(**"["**,**""**).replace(**"]"**,**""**).replace(**", "**,**"+"**)+**" = 1;"**);  count++;  output.println();  }   *// third constraint  // if we color vertex i with color k, then we are using color k  // xik <= yk, where i&k = 1,...,12* **for** (**int** k = 1; k <= 12; k++) {  **for** (**int** i = 1; i <= 12; i++) {  output.print(**"c"**+count+**": "**);  output.print(*constvars*.get(12\*(i-1)+(k-1))+**" <= "**+*objvars*.get(k-1)+**";"**);  count++;  output.println();  }  }   *// fourth constraint: requires solving for edges  // for every edge, both vertices can't have the same color  // xik + xjk <=1, where k = 1,...,12 and for edge vertices are vi,vj  // define E = {(i,j), i != j, i^2+j^2+18 is prime}* **for** (**int** i=0; i <= *vertices*.size()-2; i++) {  **for** (**int** j=i+1; j <= *vertices*.size()-1; j++) {  **int** v1 = *vertices*.get(i);  **int** v2 = *vertices*.get(j);  **int** checkNum = v1\*v1 + v2\*v2 + 18;  *// if i^2+j^2+18 is prime* **if** (*isPrime*(checkNum)) {  *// test display all edges and its i^2+j^2+18 value  // System.out.println("(" + v1 + "," + v2 + ") and " + checkNum);* **for** (**int** k = 1; k <= 12; k++) {  output.print(**"c"** + count + **": "**);  output.print(**"x\_"**+v1+**"\_"**+k+**"+x\_"**+v2+**"\_"**+k+**" <= 1;"**);  count++;  output.println();  }  }  }  }  output.println(**"\n"**);   *// output all the variable definitions  // all variables are binary* output.println(**"/\* Variable definitions \*/"**);  output.print(**"bin "**);  *// the objective function variables* output.print(*objvars*.toString().replace(**"["**,**""**).replace(**"]"**,**""**).replace(**" "**,**""**)+**","**);  *// the constraint variables* output.print(*constvars*.toString().replace(**"["**,**""**).replace(**"]"**,**""**).replace(**" "**,**""**));  output.println(**";\n"**);  }   *// This methods checks if an inputted number is prime.  // It returns True if prime and False if not.* **public static boolean** isPrime(**int** n) {  *// 2 is a prime* **if** (n == 2) {  **return true**;  }  *// all other even numbers are not prime* **if** (n % 2 == 0) {  **return false**;  }  *// check odd numbers up to sqrt(number)* **for** (**int** i = 3; i\*i <= n; i+=2) {  **if** (n % i == 0) {  **return false**;  }  }  **return true**;  } } |

I first defined all the variables in the objective function (the *y*s) and the constraints (the *x*s) and the vertices in order to make later variable printing to an output text source easier. Because we already defined that all variables are binary, I kept that in mind as I coded, since all variables were saved as strings, not binary integers. I then outputted the objective function, minimizing the sum of the *y*s. I then started outputting the many constraints we have through many for loops. The first constraint prevented us from using unnecessary colors. The second made sure all vertices had exactly one color. The third made sure if we coloring a vertex with a color, then we are using that color. To do this I made a method that will check if *i^2+j^2+18* of all combinations of vertices in *V* (never evaluate when *i=*j) is actually a prime number. If the resulting value is in fact a prime number, we have an edge. (Note: our edges are that (a,b) = (b,a), also recall that from our definition of *E*, *ab*, which explains why, in third constraint, the program does not loop for each i from 1 to 12, we do not traverse through all j from 1 to 12.) The fourth made sure colors on each side of every edge was different.

The edges and their corresponding prime number are,

|  |
| --- |
| (1,2) and 23  (1,8) and 83  (1,12) and 163  (2,3) and 31  (2,5) and 47  (2,7) and 71  (2,9) and 103  (3,4) and 43  (3,10) and 127  (4,5) and 59  (4,7) and 83  (5,6) and 79  (5,8) and 107  (6,7) and 103  (7,8) and 131  (7,10) and 167  (7,12) and 211  (8,9) and 163  (9,10) and 199  (10,11) and 239  (11,12) and 283 |

The resulting text file outputted, ready to be inputted to LPSolve is the following,

|  |
| --- |
| /\* Objective function \*/  min: +y\_1+y\_2+y\_3+y\_4+y\_5+y\_6+y\_7+y\_8+y\_9+y\_10+y\_11+y\_12;  /\* Variable bounds \*/  c1: y\_2 <= y\_1;  .  .  .  c11: y\_12 <= y\_11;  /\* 11 lines of the above type:  higher number colors cannot be used until lower number colors are used \*/  c12: x\_1\_1+x\_1\_2+x\_1\_3+x\_1\_4+x\_1\_5+x\_1\_6+x\_1\_7+x\_1\_8+x\_1\_9+x\_1\_10+x\_1\_11+x\_1\_12 = 1;  .  .  .  c23: x\_12\_1+x\_12\_2+x\_12\_3+x\_12\_4+x\_12\_5+x\_12\_6+x\_12\_7+x\_12\_8+x\_12\_9+x\_12\_10+x\_12\_11+x\_12\_12 = 1;  /\* 12 lines of the above type:  all vertices have exactly one color \*/  c24: x\_1\_1 <= y\_1;  .  .  .  c167: x\_12\_12 <= y\_12;  /\* 12\*12=144 lines of the above type:  make sure of what colors are assigned \*/  c168: x\_1\_1+x\_2\_1 <= 1;  .  .  .  c419: x\_11\_12+x\_12\_12 <= 1;  /\* 21 edges\*12=252 lines of the above type:  vertices with edges are assigned different colors \*/  /\* Variable definitions:  all variables are declared binary \*/  bin y\_1,...,y\_12,x\_1\_1,...,x\_12\_12; |

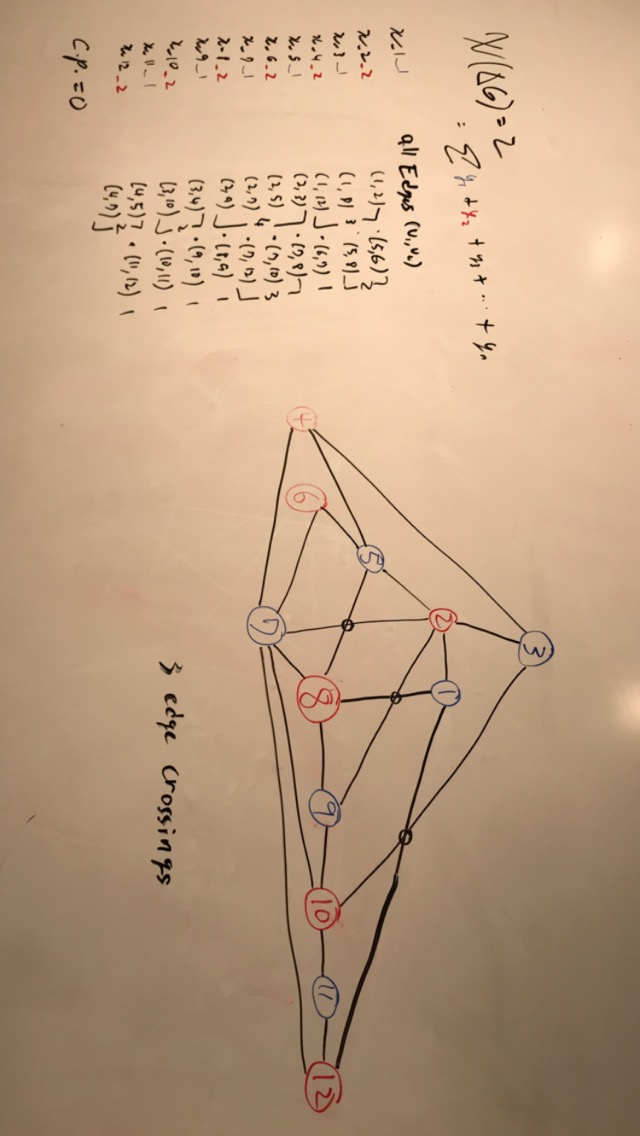
We can see that the objective function is being minimized. We have almost a total of 420 constraints to solve this IP. All variables are declared. Although very truncated, we can visually see the summed components of each equality and inequality in the objective function and all constraints.

Using this input, I solved the IP using LPSolve. With my Windows machine, a ThinkPad T470s, the entire process took 0.028 seconds, which was a lot shorter than expected. I was expecting at least a few minutes due to the shear amount of variables and constraints. The output of the IP is the following,

|  |  |
| --- | --- |
| Variables | Result |
| objective | 2 |
| y\_1 | 1 |
| y\_2 | 1 |
| x\_1\_1 | 1 |
| x\_2\_2 | 1 |
| x\_3\_1 | 1 |
| x\_4\_2 | 1 |
| x\_5\_1 | 1 |
| x\_6\_2 | 1 |
| x\_7\_1 | 1 |
| x\_8\_2 | 1 |
| x\_9\_1 | 1 |
| x\_10\_2 | 1 |
| x\_11\_1 | 1 |
| x\_12\_2 | 1 |
| All other variables | =0 |

Just as we defined our objective function and constraints, the minimized value is 2. We see that the first two colors to be used are 1 and 2. We see that all vertices have exactly one color, either 1 or 2. After properly graphing this I am sure that for every edge the vertices will have different colors. As define, all variables are binary. Therefore, our overall conclusion answering what is the chromatic number is **2**. We only need two colors to color our entire graph.

The following figure is my attempt at illustrating the feasibility of coloring our graph with the chromatic number we solved with the least number of edge crossings,

s

We can see that our graph coloring diagram with a chromatic number of 2 is actually feasible. Also, I was able to rearrange the positions of each vertices to reduce the number of edge crossings down to 3. I believe that it is possible to reduce it even further given the edges that we solved, but I personally cannot look for a potential solution.