ISyE524: Introduction to Optimization
Problem Set #5
Due: *Tuesday* April 28, at 11:59PM

Notes and Deliverables:

- Submit solutions to all problems in the form of a single PDF file of the IJulia notebook to the course dropbox. (If you do not submit as a single PDF you will lose points.)
- Be sure that your answers (including any pictures you include in the notebook) are in the correct order. (If you do not submit the problems in order, you will lose points.)

1 Ahhhhhhh, Daniel-son. You SuDoku Master.

Sudoku is a puzzle game craze that once upon a time was very popular. is sweeping the world. According to a popular sudoku web site (`https://www.learn-sudoku.com/sudoku-requires-no-math.html` "Sudoku requries no math to solve." But we will show them that we can needlessly and endlessly complicate *all* things with math, even a fun puzzle like sudoku.

In sudoku, you are given an $n \times n$ grid with some of the entries of the grid being filled in with numbers from $\{1, 2, \dots n\}$. Typically $n = 9$, and in every sudoku puzzle $\sqrt{n}$ is an integer. The object of the game is to fill in the grid so that every row, every column, and every $\sqrt{n} \times \sqrt{n}$ "box" contains the digits 1 through n. Figure 1 is a sample sudoku.

Figure 1: Sample Sudoku

	6		1		4		5	
		8	3		5	6		
2								1
8			4		7			6
		6				3		
7			9		1			4
5								2
		7	2		6	9		
	4		5		8		7	

You may (and probably should) use the class example notebook `Sudoku.ipynb` to solve these problems.

Let me help you with this problem by fixing some notation.

- Rows/Columns/Digits of the grid: $[n] = \{1, 2, \dots, n\}$, we have $n = 9$ in standard sudoku, but you can make bigger instances.

- Clusters $\mathcal{C}$, which is a set of sets. Each set contains the grid positions (pairs) in that cluster.
- "Fixed" positions given as sets $\{F_1, F_2, \ldots F_n\}$ such that

$$F_k = \{(i,j) \mid \text{ the grid position } (i,j) \text{ is the digit } k \text{ in the given sudoku instance }\}$$

In the sudoku in Figure 1, we could have the math notation that

$$\begin{aligned}\mathcal{C} = \Big\{&\{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\},\\ &\{(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)\},\\ &\ldots,\{(4,4),(4,5),(4,6),(5,4),(5,5),(5,6),(6,4),(6,5),(6,6)\},\ldots\Big\}\end{aligned}$$

where there are 9 different sets containing the pairs of grid elements.

$$\begin{aligned}F_2 &= \{(3,1),(7,9),(8,4)\}\\ F_4 &= \{(1,6),(4,4),(6,9),(9,2)\}\end{aligned}$$

1-1 Problem Write an integer program that will demonstrate how to complete a given sudoku instance of size n, with clusters given by $\mathcal{C}$, and fixed positions $F_1, F_2, \ldots F_n$.

Answer:

The objective doesn't matter, as we only seek a feasible solution

$$\max x_{666}$$

$$\begin{aligned}
\sum_{(i,j)\in C} x_{ijk} &= 1 \quad \forall C \in \mathcal{C}, \forall k \in N && \text{Each number appears once in each box}\\
\sum_{i\in N} x_{ijk} &= 1 \quad \forall j \in N, \forall k \in N && \text{Each number appears in exactly one column}\\
\sum_{j\in N} x_{ijk} &= 1 \quad \forall i \in N, \forall k \in N && \text{Each number appears in exactly one row}\\
\sum_{k\in N} x_{ijk} &= 1 \quad \forall i \in N, \forall j \in N && \text{One value assigned to each cell}\\
x_{ijk} &= 1 \quad \forall k \in N, \forall (i,j) \in F_k && \text{Fix the entries}\\
x_{ijk} &\in \{0,1\} \quad \forall i \in N, \forall j \in N, \forall k \in N && \text{Binary vars}
\end{aligned}$$

1-2 Problem Solve the model you write in Problem 1-1 in Julia and JuMP. Be sure to print out your solution.

Answer:

This one was super easy, since you had the code from Sudoku.ipynb.

Julia Code

```julia
  given = [
    0 6 0  1 0 4  0 5 0
    0 0 8  3 0 5  6 0 0
    2 0 0  0 0 0  0 0 1

    8 0 0  4 0 7  0 0 6
    0 0 6  0 0 0  3 0 0
    7 0 0  9 0 1  0 0 4

    5 0 0  0 0 0  0 0 2
    0 0 7  2 0 6  9 0 0
    0 4 0  5 0 8  0 7 0
  ]

  # helper function to print a sudoku grid
  function printSudoku(arr)
      u = 0
      println("+-------+-------+-------+")
      for p in 1:3:9
        for q in 0:2
          print("| ")
          for r in 1:3:9
            for s in 0:2
              u = round(Int, arr[p+q,r+s])
              u == 0 ? print(" ") : print(u)
              print(" ")
            end
            print("| ")
                    end
          println()
        end
        println("+-------+-------+-------+")
      end
    end
    ;


using JuMP, HiGHS

m = Model(HiGHS.Optimizer)
set_silent(m)

@variable(m, x[1:9,1:9,1:9], Bin)

# exactly one number per cell
for i in 1:9
  for j in 1:9
    @constraint(m, sum(x[i,j,k] for k in 1:9) == 1)
  end
end

```

```
# exactly one of each number per row
for i in 1:9
  for k in 1:9
    @constraint(m, sum(x[i,j,k] for j in 1:9) == 1)
  end
end

# exactly one of each number per column
for j in 1:9
  for k in 1:9
    @constraint(m, sum(x[i,j,k] for i in 1:9) == 1)
  end
end

# exactly one of each number per 3x3 block
for k in 1:9
  for p in 0:3:6
    for q in 0:3:6
      @constraint(m, sum(x[p+i,q+j,k] for i in 1:3, j in 1:3) == 1)
    end
  end
end

# initial conditions
for i in 1:9
  for j in 1:9
    if given[i,j] != 0
      @constraint(m, x[i,j,given[i,j]] == 1)
    end
  end
end

@time(optimize!(m))

@assert is_solved_and_feasible(m)

solution = zeros(9,9)
for i in 1:9
    for j in 1:9
    for k in 1:9
        if value(x[i,j,k]) == 1
        solution[i,j] = k
        continue
        end
    end
    end
end

println("The given problem is: ")
printSudoku(given)

println("The solution is: ")
```

```
printSudoku(solution)
```

Julia Answer

```
The given problem is:
+-------+-------+-------+
|   6   | 1   4 |   5   |
|     8 | 3   5 | 6     |
| 2     |       |     1 |
+-------+-------+-------+
| 8     | 4   7 |     6 |
|     6 |       | 3     |
| 7     | 9   1 |     4 |
+-------+-------+-------+
| 5     |       |     2 |
|     7 | 2   6 | 9     |
|   4   | 5   8 |   7   |
+-------+-------+-------+
The solution is:
+-------+-------+-------+
| 9 6 3 | 1 7 4 | 2 5 8 |
| 1 7 8 | 3 2 5 | 6 4 9 |
| 2 5 4 | 6 8 9 | 7 3 1 |
+-------+-------+-------+
| 8 2 1 | 4 3 7 | 5 9 6 |
| 4 9 6 | 8 5 2 | 3 1 7 |
| 7 3 5 | 9 6 1 | 8 2 4 |
+-------+-------+-------+
| 5 8 9 | 7 1 3 | 4 6 2 |
| 3 1 7 | 2 4 6 | 9 8 5 |
| 6 4 2 | 5 9 8 | 1 7 3 |
+-------+-------+-------+
```

In real sudoku games, there is only *one* feasible solution to the IP formulation you built in Problem 1-1. (That is why it shouldn't matter what your objective function is). In this problem, we make the game more challenging. We will call my new game *Jeff-Doku.* In Jeff-Doku, we the set of main "diagonal" grid locations as

$$D = \{(1,1),(2,2),\ldots(n,n)\}$$

and we maximize the sum of the elements placed on the diagonal. But we need to "relax" the restriction that we have to match *all* of the given squares.

1-3 Problem Suppose now, that you need not fix *all* of the values given in the instance, but rather you need fix only at least $K = 24$ of them. Modify your instance from Problem 1-1 to solve this more challenging version of the game that maximizes the sum of the elements on the main diagonal. Even if you can solve sudoku by hand, I'll bet you *can't* solve Jeff-doku by hand!

Answer:

Julia Code

```julia
  using JuMP, HiGHS

m = Model(HiGHS.Optimizer)
set_silent(m)

@variable(m, x[1:9,1:9,1:9], Bin)
@variable(m, z[1:9,1:9], Bin)

# exactly one number per cell
for i in 1:9
  for j in 1:9
    @constraint(m, sum(x[i,j,k] for k in 1:9) == 1)
  end
end

# exactly one of each number per row
for i in 1:9
  for k in 1:9
    @constraint(m, sum(x[i,j,k] for j in 1:9) == 1)
  end
end

# exactly one of each number per column
for j in 1:9
  for k in 1:9
    @constraint(m, sum(x[i,j,k] for i in 1:9) == 1)
  end
end

# exactly one of each number per 3x3 block
for k in 1:9
  for p in 0:3:6
    for q in 0:3:6
      @constraint(m, sum(x[p+i,q+j,k] for i in 1:3, j in 1:3) == 1)
    end
  end
end

# initial conditions (relaxed)
for i in 1:9
  for j in 1:9
    if given[i,j] != 0
      @constraint(m, x[i,j,given[i,j]] >= z[i,j])
    end
  end
end

@constraint(m, sum(z[i,j] for i in 1:9, j in 1:9 if given[i,j] != 0) >= 24)
@objective(m, Max, sum(k*x[i,i,k] for i in 1:9, k in 1:9))


```

```
@time(optimize!(m))

@assert is_solved_and_feasible(m)

solution = zeros(9,9)
for i in 1:9
    for j in 1:9
    for k in 1:9
        if value(x[i,j,k]) > 0.9
        solution[i,j] = k
        continue
        end
    end
    end
end


println("The given problem is: ")
printSudoku(given)

println("The solution is: ")
printSudoku(solution)

sumdiag = sum(solution[i,i] for i in 1:9)
print("Sum of diagonal elements is:", sumdiag)

```

Julia Answer

```
  The given problem is:
+-------+-------+-------+
|   6   | 1   4 |   5   |
|     8 | 3   5 | 6     |
| 2     |       |     1 |
+-------+-------+-------+
| 8     | 4   7 |     6 |
|     6 |       | 3     |
| 7     | 9   1 |     4 |
+-------+-------+-------+
| 5     |       |     2 |
|     7 | 2   6 | 9     |
|   4   | 5   8 |   7   |
+-------+-------+-------+
The solution is:
+-------+-------+-------+
| 9 6 3 | 1 7 4 | 2 5 8 |
| 1 7 8 | 9 2 5 | 6 4 3 |
| 2 5 4 | 8 6 3 | 9 7 1 |
+-------+-------+-------+
| 8 2 1 | 4 3 7 | 5 9 6 |
| 4 9 6 | 5 8 2 | 3 1 7 |
| 7 3 5 | 6 1 9 | 8 2 4 |
+-------+-------+-------+
| 5 8 9 | 3 4 1 | 7 6 2 |
| 3 1 7 | 2 9 6 | 4 8 5 |
| 6 4 2 | 7 5 8 | 1 3 9 |
+-------+-------+-------+
Sum of diagonal elements is:65.0
```

1-4 Problem Even that problem was "too easy". Now add the additional constraints and variables necessary to enforce the constraint that if you put 2 or more "9"s on the main diagonal in your Jeff-doku solution, then you must also put *exactly* 3 "5"'s on the diagonal. You need not solve this model. Just write the new constraints that you need when compared to your answer to Problem 1-3.

Answer:

We need to model the conditions

$$\sum_{i \in N} x_{ii9} \geq 2 \Rightarrow \sum_{i \in N} x_{ii5} \geq 3$$
$$\sum_{i \in N} x_{ii9} \geq 2 \Rightarrow \sum_{i \in N} x_{ii5} \leq 3$$

which we model with the three conditions

$$\sum_{i \in N} x_{ii9} \geq 2 \Rightarrow \delta = 1 \tag{1}$$
$$\delta = 1 \Rightarrow \sum_{i \in N} x_{ii5} \geq 3 \tag{2}$$
$$\delta = 1 \Rightarrow \sum_{i \in N} x_{ii5} \leq 3. \tag{3}$$

For (1), we need to do the contrapositive. So instead we model

$$\delta = 0 \Rightarrow \sum_{i \in N} x_{ii9} \leq 1. \tag{4}$$

The trick for $\delta = 0 \Rightarrow \sum_{j \in N} a_j x_j \leq b$ is $\sum_{j \in N} a_j x_j \leq b + M\delta$. For (4), $M = 9 - 1 = 8$, so we add the constraint

$$\sum_{i \in N} x_{ii9} - 8\delta \leq 1. \tag{5}$$

For (2), we need the trick for $\delta = 1 \Rightarrow \sum_{j \in N} a_j x_j \geq b$, which is $\sum_{j \in N} a_j x_j + m\delta \geq m + b$. In the case of (2), we have that $m = -3$, so the constraint to add is

$$\sum_{i \in N} x_{ii5} - 3\delta \geq 0. \tag{6}$$

Finally, for (3), we need the trick for $\delta = 1 \Rightarrow \sum_{j \in N} a_j x_j \leq b$, which is $\sum_{j \in N} a_j x_j + M\delta \leq M + b$. In the case of (3), we get that $M = 9 - 3 = 6$, so the constraint becomes

$$\sum_{i \in N} x_{ii5} + 6\delta \leq 9. \tag{7}$$

Adding the constraints (5) (6), (7), and $\delta \in \{0,1\}$ will accomplish the required constraints stated in the problem.

2 Integer Programming is Smarter than a Fourth Grader?

This was my son's homework assignment when he was in 4th grade. I tried to make him use integer programming to figure out the answer. He told me to buzz off, but since you hopefully have more respect for your professor than my son does for me, you will be ***happy*** to use integer programming to solve the following mind-bender:

Maxine, Mabel, Mavis, Millie, and Martha were grandmothers. Their daughters were named Carla, Carol, Cindy, Cathy, and Caren, and the daugthers were married to John, Jake, Jack, Joe, and Jason. Each of the couples had one son. The names of the grandsons were Tom, Tex, Tim, Tip, and Tab.

You also know the following facts/clues:

1. Maxine's Daughter was not Carla
2. Mavis' son-in-law was not named Jack, and Catchy was married to Joe, but their son was not named Tab
3. Tim's mother was not named either Carol or Carla, because Tim's mother was Jake's wife Cindy
4. Mabel, Millie, and Martha did not have daughters named Carla or Carol, and Martha's son-in-law was not John because John was married to Caren, and their son was named Tom
5. Millie was terrible with names, but she knew that her son-in-law was named either Joe or Jason, and her grandson was named either Tip or Tab.
6. Tab did not have a grandmother named Mavis.

Who was Maxine's grandson? In fact, we can determine everyone's grandson. This is a bit hard. Here are some hints.

- You will need at least the following binary variables in your model
  - $\mathsf{xD}_{g,d}$ if Grandma $g \in G$ has Daughter $d \in D$
  - $\mathsf{xH}_{g,h}$ if Grandma $g \in G$ has Son-In-Law $h \in H$. (That is, the daughter of Grandma g has husband h)
  - $\mathsf{xS}_{g,s}$ if Grandma $g \in G$ has Grandson $s \in S$
- Write the constraints that every grandmother has one daughter, and every daughter has one grandmother
- Same for son-in-law and grandson.
- Then implement each of the rules above. (Some of the bullets will require multiple constraints).
- **CHECK TO SEE IF YOU HAVE AN ANSWER AFTER IMPLEMENTING EACH CONSTRAINT**. The instance is not infeasible, so if you are getting infeasible, you have implemented a constraint incorrectly.

2-1 Problem Write a math model whose solution will determine for each grandmother, who her daughter, son-in-law, and grandson is.

Answer:

- $\mathsf{xD}_{g,d}$ if Grandma $g \in G$ has Daughter $d \in D$
- $\mathsf{xH}_{g,h}$ if Grandma $g \in G$ has Son-In-Law $h \in H$. (That is, the daughter of Grandma g has husband h)
- $\mathsf{xS}_{g,s}$ if Grandma $g \in G$ has Grandson $s \in S$
- Every grandmother has one daughter:
$$\sum_{d \in D} \mathsf{xD}_{gd} = 1 \quad \forall g \in G$$
- Every daughter has one grandmother:
$$\sum_{g \in G} \mathsf{xD}_{gd} = 1 \quad \forall d \in D$$
- Every grandmother has one son-in-law:
$$\sum_{h \in H} \mathsf{xH}_{gh} = 1 \quad \forall g \in G$$
- Every son-in-law has one grandmother:
$$\sum_{g \in G} \mathsf{xH}_{gh} = 1 \quad \forall h \in H$$
- Every grandmother has one grandson:
$$\sum_{s \in S} \mathsf{xS}_{gs} = 1 \quad \forall g \in G$$
- Every grandson has one grandmother:
$$\sum_{g \in G} \mathsf{xS}_{gs} = 1 \quad \forall s \in S$$
- Maxine's daughter was not Carla:
$$\mathsf{xD}_{\mathsf{maxine,carla}} = 0$$
- Mavis' son in law was not Jack
$$\mathsf{xH}_{\mathsf{mavis,jack}} = 0$$

- Cathy was married to Joe
$$\mathsf{xD}_{g,\mathsf{cathy}} = \mathsf{xH}_{g,\mathsf{joe}} \quad \forall g \in G$$
- Cathy's son is not named Tab:
$$\mathsf{xD}_{g,\mathsf{cathy}} + \mathsf{xS}_{g,\mathsf{tab}} \leq 1 \quad \forall g \in G$$
- Joe's son is not named Tab:
$$\mathsf{xH}_{g,\mathsf{cathy}} + \mathsf{xS}_{g,\mathsf{tab}} \leq 1 \quad \forall g \in G$$
- Tim's mother is not named Carol: (not really needed)
$$\mathsf{xS}_{g,\mathsf{tim}} + \mathsf{xD}_{g,\mathsf{carol}} \leq 1 \quad \forall g \in G$$
- Tim's mother is not named Carla: (not really needed)
$$\mathsf{xS}_{g,\mathsf{tim}} + \mathsf{xD}_{g,\mathsf{carla}} \leq 1 \quad \forall g \in G$$
- Tim's mother is Cindy:
$$\mathsf{xS}_{g,\mathsf{tim}} = \mathsf{xD}_{g,\mathsf{cindy}} \forall g \in G$$
- Jake's wife was Cindy:
$$\mathsf{xH}_{g,\mathsf{jake}} = \mathsf{xD}_{g,\mathsf{cindy}} \forall g \in G$$
- Mabel, Millie, and Martha did not have daughters named Carla or Carol. Define $M_1 = \{\mathsf{mabel}, \mathsf{millie}, \mathsf{martha}\}$, $C_1 = \{\mathsf{carla}, \mathsf{carol}\}$, then
$$\sum_{g \in M_1} \sum_{d \in C_1} \mathsf{xD}_{gd} = 0$$
- Martha's son in law was not John:
$$\mathsf{xH}_{\mathsf{martha},\mathsf{john}} = 0$$
- John was married to Caran:
$$\mathsf{xH}_{g,\mathsf{john}} = \mathsf{xD}_{g,\mathsf{caren}} \forall g \in G$$
- Son of John was Tom:
$$\mathsf{xH}_{g,\mathsf{john}} = \mathsf{xS}_{g,\mathsf{tom}} \forall g \in G$$
- Son of Caren was Tom:
$$\mathsf{xD}_{g,\mathsf{caren}} = \mathsf{xS}_{g,\mathsf{tom}} \forall g \in G$$
- Mille knew her son in law was Joe or Jason
$$\mathsf{xH}_{\mathsf{millie},\mathsf{joe}} + \mathsf{xH}_{\mathsf{millie},\mathsf{jason}} = 1$$

- Mille's grandson was either Tip or Tab

$$\mathsf{xS}_{\mathsf{millie,tip}} + \mathsf{xS}_{\mathsf{millie,tab}} = 1$$

- Tab's grandma was not Mavis

$$\mathsf{xS}_{\mathsf{mavis,tab}} = 0$$

2-2 Problem Implement your model from 2-1 and print out each grandmothers, daughter, son-in-law, and grandson. Here is some helpful code.

Julia Code

```julia
G = [:Maxine, :Mabel, :Mavis, :Millie, :Martha]
D = [:Carla, :Carol, :Cindy, :Cathy, :Caren ]
H = [:John, :Jake, :Jack, :Joe, :Jason ]
S = [:Tom, :Tex, :Tim, :Tip, :Tab ]

@variable(m, xD[G,D], Bin) # 1 iff grandma g has daughter d
@variable(m, xH[G,H], Bin) # 1 iff grandma g has (son-in-law) (daughter's husband) h
@variable(m, xS[G,S], Bin) # 1 iff grandma g has grandson s


function printGrandmaSolution(xD, xH, xS)
    for g in G
        grandma = g
        daughter = :Unknown
        son_in_law = :Unknown
        grandson = :Unknown

        for d in D
            if value(xD[g,d]) > 0.5
                daughter = d
            end
        end
        for h in H
            if value(xH[g,h]) > 0.5
                son_in_law = h
            end
        end
        for s in S
            if value(xS[g,s]) > 0.5
                grandson = s
            end
        end
        println("Grandma ", g, " has daughter ", daughter, " son-in-law ", son_in_law,
        " and grandson ", grandson)
    end
end;
```

Answer:

Julia Code

```julia
using JuMP, HiGHS
m = Model(HiGHS.Optimizer)
set_silent(m)

@variable(m, xD[G,D], Bin) # 1 iff grandma g has daughter d
@variable(m, xH[G,H], Bin) # 1 iff grandma g has (son-in-law) (daughter’s husband) h
@variable(m, xS[G,S], Bin) # 1 iff grandma g has grandson s

# Everyone grandmother has one daughter
@constraint(m, [g in G], sum(xD[g,d] for d in D) == 1)

# Everyone daughter has one grandmother
@constraint(m, [d in D], sum(xD[g,d] for g in G) == 1)

# Everyone grandmother has one son-in-law
@constraint(m, [g in G], sum(xH[g,h] for h in H) == 1)

# Everyone son-in-law has one grandmother
@constraint(m, [h in H], sum(xH[g,h] for g in G) == 1)

# Everyone grandmother has one grandson
@constraint(m, [g in G], sum(xS[g,s] for s in S) == 1)

# Everyone grandson has one grandmother
@constraint(m, [s in S], sum(xS[g,s] for g in G) == 1)

# Now put all the rules

#1. Maxime’s daughter was not Carla
@constraint(m, xD[:Maxine,:Carla] == 0)

#2a. Mavis’ son in law was not Jack
@constraint(m, xH[:Mavis,:Jack] == 0)

#2b. Cathy was married to Joe
@constraint(m, [g in G], xD[g,:Cathy] == xH[g,:Joe])

#2c. Cathy’s Son is not named Tab
@constraint(m, [g in G], xD[g,:Cathy] + xS[g,:Tab] <= 1)

#2d. Joe’s Son is not named Tab
@constraint(m, [g in G], xH[g,:Joe] + xS[g,:Tab] <= 1)

#3a. Tim’s mother is not named Carol
@constraint(m, [g in G], xS[g,:Tim] + xD[g,:Carol] <= 1)

#3b. Tim’s mother is not named Carla
@constraint(m, [g in G], xS[g,:Tim] + xD[g,:Carla] <= 1)

#3c. Tim’s mother was Cindy
@constraint(m, [g in G], xS[g,:Tim] == xD[g,:Cindy])
```

```
#3d Jake's wife was Cindy
@constraint(m, [g in G], xH[g,:Jake] == xD[g,:Cindy])

#4a. Mabel, Millie, and Martha did not have daughters named Carla or Carol
MMM = [:Mabel, :Millie, :Martha]
CC = [:Carla, :Carol]
@constraint(m, sum(xD[g,d] for g in MMM, d in CC) <= 0)

#4b. Martha's son in law was not John
@constraint(m, xH[:Martha,:John] == 0)

#4c John was married to Caren
@constraint(m, [g in G], xH[g,:John] == xD[g,:Caren])

#4d Son of John was Tom
@constraint(m, [g in G], xH[g,:John] == xS[g,:Tom])

#4e Son of Caren was Tom
@constraint(m, [g in G], xD[g,:Caren] == xS[g,:Tom])
# 4c redundant...

# 5a Mille knew her son in law was Joe or Jason
@constraint(m, xH[:Millie,:Joe] + xH[:Millie,:Jason] == 1)

# 5b Mille's grandson was either Tip or Tab
@constraint(m, xS[:Millie,:Tip] + xS[:Millie,:Tab] == 1)

#6 Tab's grandma was not Mavis
@constraint(m, xS[:Mavis,:Tab] == 0)


optimize!(m)
@assert is_solved_and_feasible(m)

printGrandmaSolution(xD, xH, xS)
```

Julia Answer

```
Grandma Maxine has daughter Carol son-in-law Jack and grandson Tab
Grandma Mabel has daughter Caren son-in-law John and grandson Tom
Grandma Mavis has daughter Carla son-in-law Jason and grandson Tex
Grandma Millie has daughter Cathy son-in-law Joe and grandson Tip
Grandma Martha has daughter Cindy son-in-law Jake and grandson Tim
```

3 An Auror is Near

He-Who-Shall-Not-Be Named is back, and all wizards need protection by aurors. The wizarding world has been divided into eight districts. The time (in seconds) required to travel from one district to another via the floo network is shown in the table below:

	1	2	3	4	5	6	7	8
1		3	4	6	1	9	8	10
2	3		5	4	8	6	1	9
3	4	5		2	2	3	5	7
4	6	4	2		3	2	5	4
5	1	8	2	3		2	2	4
6	9	6	3	2	2		3	2
7	8	1	5	5	2	3		2
8	10	9	7	4	4	2	2	

The wizard population in district i (in thousands) is given by p_i for $i = 1, \ldots, 8$. In this instance, $p = (40, 30, 35, 20, 15, 50, 45, 60)$, so the population of district 1 is 40 thousand wizards, etc. The Ministry of Magic has declared that aurors will be located at 3 locations.

3-1 Problem Write an integer programming model that will determine the locations of the aurors that maximize the number of people who live within two seconds of an auror.

Hint: Your solution will need two sets of binary decision variables: one set to determine whether or not each district is assigned as an auror location, and the other set to determine for each district if it has an auror assigned to a district within 2 seconds.

Answer:
Define the following decision variables:

1. x_i = binary variable, with $x_i = 1$ indicates district i is selected as an auror location and $x_i = 0$ otherwise, for $i = 1, \ldots, 8$
2. y_i = binary variable, with $y_i = 1$ ensuaring that an auror is assigned to a district within two seconds of district i, and $y_i = 0$ otherwise, for $i = 1, \ldots, 8$

The objective is to maxmize the protected population:

$$\max \sum_{i=1}^{8} p_i y_i$$

The first constraint states that we can choose only 3 auror locations:

$$\sum_{i=1}^{8} x_i = 3$$

Each district can be considered covered only if a district that is within 2 seconds of that district is selected. For district 1, the districts within 2 seconds are just district 1 and 5. Thus, we must enforce the constraint that $y_1 = 1$ should imply $x_1 + x_5 \geq 1$ (or, equivalently, via the contrapositive, if $x_1 + x_5 \leq 0$ then $y_1 = 0$). This can be formulated as:

$$x_1 + x_5 \geq y_1.$$

Similar logic for the other districts yields the inequalities:

$$
\begin{aligned}
x_2 + x_7 &\geq y_2 \\
x_3 + x_4 + x_5 &\geq y_3 \\
x_3 + x_4 + x_6 &\geq y_4 \\
x_3 + x_5 + x_6 + x_7 &\geq y_5 \\
x_4 + x_5 + x_6 + x_8 &\geq y_6 \\
x_2 + x_5 + x_7 + x_8 &\geq y_7 \\
x_6 + x_7 + x_8 &\geq y_8
\end{aligned}
$$

Finally, all variables are binary:

$$x_i, y_i \text{ is binary}, \quad i = 1, \ldots, 8$$

Note: It is also true by the definition of our decision variables that we want, e.g., $x_1 + x_5 \geq 1$ to imply $y_1 = 1$ (so we get credit for covering district 1 if it is covered). This is the reverse of the logic that we modeled above. However, we do not need to add constraints to model this direction because due to the the objective function, an an optimal solution y_1 will be set equal to 1 unless it is constrained otherwise. So, the important direction in this logic is just to enforce that if we do *not* have district 1 covered ($x_1 + x_5 \leq 0$) then $y_1 = 0$ (so we don't take credit in the objective for covering a district we did not actually cover).

3-2 Problem Implement your model from Problem 3-1 in Julia and JuMP to determine the best three locations for aurors.

Answer:

Julia Code

```julia
N = 8

# parameters
t = [ 0 3 4 6 1 9 8 10
 3 0 5 4 8 6 1 9
 4 5 0 2 2 3 4 7
 6 4 2 0 3 2 5 4
 1 8 2 3 0 2 2 4
 9 6 3 2 2 0 3 2
 8 1 5 5 2 3 0 2
 10 9 7 4 4 2 2 0
]

p = [40 30 35 20 15 50 45 60]

#variables
m = Model(HiGHS.Optimizer)
@variable(m, x[1:N], Bin)
@variable(m, y[1:N], Bin)

```

```
    # Maximize covered population
    @objective(m, Max, sum(p[i]*y[i] for i in 1:N))

    #Open three facilities
    @constraint(m, sum(x[i] for i in 1:N) == 3)

    # if i covered, then it is within 2 min of some opened facility
    @constraint(m, [i in 1:N], sum(x[j] for j in 1:N if t[j,i] <= 2) >= y[i])

    optimize!(m)

    println("Covered population: ", objective_value(m))
    openfac = [i for i in 1:N if value(x[i]) > 0.1]
    covered = [i for i in 1:N if value(y[i]) > 0.1]
    println("Open facilities : ", openfac)
    println("Covered areas : ", covered)
```

Julia Answer

```
    Covered population: 295.0
Open facilities : [3, 5, 7]
Covered areas : [1, 2, 3, 4, 5, 6, 7, 8]
```

So by putting aurors in locations 3, 5, and 7, you can cover ALL THE WIZARDS.

4 Routing Apple Deliveries

A local apple orchard needs to deliver orders for its apples to six customers. The time (in minutes) between the orchard (O) and each customer, and also between each pair of customers, is given in the table below. The table also gives the size of each customer's order (in crates). (The time between locations is the same in both directions, so for each pair of locations the time is only reported once.)

	Time						
	O	C1	C2	C3	C4	C5	C6
O	-	18	9	11	14	21	12
C1	-	-	9	16	4	5	10
C2	-	-	-	9	6	13	4
C3	-	-	-	-	14	21	22
C4	-	-	-	-	-	7	9
C5	-	-	-	-	-	-	12

	C1	C2	C3	C4	C5	C6
Order size (crates)	60	55	30	45	25	50

The orchard has one truck that can carry at most 100 crates of apples. (Therefore, the truck will have to make multiple trips to make all the deliveries) Each trip can visit more than one customer, as long as the *total* number of crates on the truck at the start of the trip does not exceed the capacity. However, each order must be delivered all at once (i.e., a customer's order should not be split into two trips.)

4-1 Problem Formulate an integer program to help the orchard assign customers to the trips so that the total time the truck has to travel per day is minimized. (Hint: model the problem as a set covering problem.)

Answer:
The first step is to create a list of the possible routes a truck can travel, and record which customers it covers and what is the time of the route. A group of customers can be put together on a route if the total number of crates is less than or equal to the capacity of 100. It's also possible that a route could consist of a single customer. The set of all possible routes, and their total times are given in the table below.

Route j	Customers Covered	Time (t_j)
1	C1	18+18=36
2	C2	9+9=18
3	C3	11+11=22
4	C4	14+14=28
5	C5	21+21=42
6	C6	12+12=24
7	C1,C3	18+16+11=45
8	C1,C5	18+5+21=44
9	C2,C3	9+9+11=29
10	C2,C4	9+6+14=29
11	C2,C5	9+13+21=43
12	C3,C4	11+14+14=39
13	C3,C5	11+21+21=56
14	C3,C6	11+22+12=45
15	C4,C5	14+7+21=42
16	C4,C6	14+9+12=35
17	C5,C6	21+12+12=45
18	C3,C4,C5	11+14+7+21=53

For the first five routes, the time is just the time to travel to the customer and then return to the orchard. For other routes with the exception of route 18 (each with 2 customers), the total time is computed by adding the time to travel to the first customer, then the time to travel from the first to the second customer, and finally the time to travel from the second customer back to the orchard. For the last route in our list, there are 3 customers, so we must choose sequence to visit them with minimum time (alternatively we could define a "route" for each possible sequence, but the solution would never choose one that is not minimum time). The unique possible sequences are $C3 \rightarrow C4 \rightarrow C5$ (cost of 11+14+7+21=53), $C3 \rightarrow C5 \rightarrow C4$ (cost of 11+21+7+21=60), and $C4 \rightarrow C3 \rightarrow C5$ (cost of 14+14+21+21=70) so we use the minimum length sequence of 53. (The other three possible sequences are reverses of these three, so do not need to be separately considered.)

We then let the decision variable be:

$x_j = 1$ if route j is used, and $x_j = 0$ otherwise, for $j = 1, \ldots, 18$.

The objective is to minimize the total time:

$$\min \sum_{j=1}^{18} t_j x_j$$

The constraints then are to make sure each customer is visited once:

$$\begin{aligned}
x_1 + x_7 + x_8 &\geq 1 \quad \text{(Customer 1)} \\
x_2 + x_9 + x_{10} + x_{11} &\geq 1 \quad \text{(Customer 2)} \\
x_3 + x_7 + x_9 + x_{12} + x_{13} + x_{14} + x_{18} &\geq 1 \quad \text{(Customer 3)} \\
x_4 + x_{10} + x_{12} + x_{15} + x_{16} + x_{18} &\geq 1 \quad \text{(Customer 4)} \\
x_5 + x_8 + x_{11} + x_{13} + x_{15} + x_{17} + x_{18} &\geq 1 \quad \text{(Customer 5)} \\
x_6 + x_{14} + x_{16} + x_{17} &\geq 1 \quad \text{(Customer 6)}
\end{aligned}$$

Finally, we must specify that the variables are binary:

$$x_j \text{ is binary}, j = 1, \ldots, 14$$

4-2 Problem Implement and solve the model using the JuMP package in Julia, and explain the solution in terms of the problem.

Answer:

I'll give two answers for this one, the write-it-out way and the matrix way

Julia Code

```julia
using JuMP, HiGHS

N = 18
t = [36, 18, 22, 28, 42, 24, 45, 44, 29, 29, 43, 39, 56, 45, 42, 35, 45, 53]

m = Model(HiGHS.Optimizer)
@variable(m, x[1:N], Bin)
@objective(m, Min, sum(t[j]*x[j] for j in 1:N))

@constraint(m, x[1] + x[7] + x[8] >= 1)
@constraint(m, x[2] + x[9] + x[10] + x[11] >= 1)
@constraint(m, x[3] + x[7] + x[9] + x[12] + x[13] + x[14] + x[18] >= 1)
@constraint(m, x[4] + x[10] + x[12] + x[15] + x[16] + x[18] >= 1)
@constraint(m, x[5] + x[8] + x[11] + x[13] + x[15] + x[17] + x[18] >= 1)
@constraint(m, x[6] + x[14] + x[16] + x[17] >= 1)
optimize!(m)
for j in 1:N
    if value(x[j]) > 0.1
        println("Do route ", j)
    end
end
println("Total time: ", objective_value(m))
```

The second way

Julia Code
```
N = 18
t = [36, 18, 22, 28, 42, 24, 45, 44, 29, 29, 43, 39, 56, 45, 42, 35, 45, 53]
A = [
    1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0
    0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0
    0 0 1 0 0 0 1 0 1 0 0 1 1 1 0 0 0 1
    0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 1 0 1
    0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 1 1
    0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0
]

m = Model(HiGHS.Optimizer)
@variable(m, x[1:N], Bin)
@objective(m, Min, t'x)
@constraint(m, A*x .>= 1)
optimize!(m)
for j in 1:N
    if value(x[j]) > 0.1
        println("Do route ", j)
    end
end
println("Total time: ", objective_value(m))
```

Either gives answer:

Julia Answer
```
    Do route 8
    Do route 9
    Do route 16
    Total time: 108.0
```

5 Paint Production

As part of its weekly production, a paint company produces five batches of paints, always the same, for some big clients who have a stable demand. Every paint batch is produced in a single production process, all in the same blender that needs to be cleaned between each batch. The durations of blending paint batches 1 to 5 are 40, 35, 45, 32 and 50 minutes respectively. The cleaning times depend of the colors and the paint types. For example, a long cleaning period is required if an oil-based paint is produced after a water-based paint, or to produce white paint after a dark color. The times are given in minutes in the following matrix A where A_{ij} denotes the cleaning time after batch i if it is followed by batch j.

$$A = \begin{bmatrix} 0 & 11 & 7 & 13 & 11 \\ 5 & 0 & 13 & 15 & 15 \\ 13 & 15 & 0 & 23 & 11 \\ 9 & 13 & 5 & 0 & 3 \\ 3 & 7 & 7 & 7 & 0 \end{bmatrix}$$

Since the company has other activities, it wishes to deal with this weekly production in the shortest possible time (blending and cleaning).

5-1 Problem Write an integer programming model to minimize the time for weekly production and to determine the corresponding order of paint batches. The order will be applied every week, so the cleaning time between the last batch of one week and the first of the following week needs to be accounted for in the total duration of cleaning.

Answer:

This is a traveling salesman problem. We will implement it with Miller-Tucker-Zemlin (MTZ) constraints.

We have $n = 5$ jobs. The total time to do all of the batches will be the sum of the blending times plus the cleaning times. The blending times are just a constant, so we are only trying to minimize the sum of the cleaning times (which depend on the order of processing). Let A_{ij} be the time to clean the blender if paint i precedes paint j.

Let binary decision variables x_{ij} that take the value 1 if job i immediately precedes job j in the ordering, and let continuous variable u_j be the order of job j in the

$$\min \sum_{i=1}^{5} \sum_{j=1}^{5} A_{ij} x_{ij}.$$

$$\begin{aligned} \sum_{j=1}^{5} x_{ij} &= 1 \qquad \forall i = 1, \ldots, 5 \\ \sum_{i=1}^{5} x_{ij} &= 1 \qquad \forall j = 1, \ldots, 5 \\ u_i - u_j + 5x_j &\le 4 \qquad \forall i = 1, \ldots, 5, \forall j = 2, \ldots, 5 \\ x_{ij} &\in \{0, 1\} \qquad \forall i = 1, \ldots, 5, \forall j = 1, \ldots, 5 \\ u_j &\ge 0 \qquad \forall j = 1, \ldots, 5 \end{aligned}$$

5-2 Problem Implement your model from Problem 5-1 in Julia and JuMP to determine the minimum time and order of production

Answer:

Julia Code

```julia
# This is a traveling salesman problem!
# Cities correspond to paint batches, and the time it takes to travel
# from one city to the next is the time it takes to clean the blender
#in between two particular batches.

# Because the cleaning time between the last batch of one week and the first batch
# of the following week needs to be accounted for,
# this is EXACTLY a traveling salesman problem!

# A[i,j] is the time it takes to travel i --> j
A = [  0  11   7  13  11
       5   0  13  15  15
      13  15   0  23  11
       9  13   5   0   3
       3   7   7   7   0 ]

# time it takes to blend a particular batch. This information isn’t important,
# because we need to blend each batch
# every week no matter what order they’re blended in. So this is a fixed cost!
b = [ 40, 35, 45, 32, 50 ]

using JuMP, HiGHS
m = Model(HiGHS.Optimizer)

n = length(b)  # number of paints

# x[i,j] = 1 if we blend j immediately after having blended i.
@variable(m, x[1:n,1:n], Bin)

# can’t flow from a node to itself
@constraint(m, nzdiag[i=1:n], x[i,i] == 0 )

# flow constraints (exactly one flow out of each )
@constraint(m,  inflows[i=1:n], sum( x[i,j] for j=1:n ) == 1 )
@constraint(m, outflows[j=1:n], sum( x[i,j] for i=1:n ) == 1 )

# additional variables and constraints for MTZ formulation
@variable(m, u[1:n] >= 0)
@constraint(m, MTZ[i=1:n, j=2:n], u[i] - u[j] + n*x[i,j] <= n-1)

@objective(m, Min, sum( x[i,j]*A[i,j] for i=1:n, j=1:n ))

optimize!(m)

X = value.(x);
for i = 1:n
    for j=1:n
        if X[i,j] >= .99
            println(" blend ",i," is followed by blend ",j," requiring cleanup time ",A[i,j])
        end
    end
```

```
end

println("and the minimum cleanup time is: ", objective_value(m), " min, (plus ", sum(b), " min of blending)")
```

Gives output:

Julia Answer

```
blend 2 is followed by blend 1 requiring cleanup time 5
 blend 3 is followed by blend 5 requiring cleanup time 11
 blend 4 is followed by blend 3 requiring cleanup time 5
 blend 5 is followed by blend 2 requiring cleanup time 7
and the minimum cleanup time is:
41.0 min, (plus 202 min of blending)
```