MAT 160 - Project 2

Sangwon Yoon, Ryan Tjoa and Ahmed Mahmoud

Problem 1:

(a) In this problem, we are looking for the maximum complete subgraph (maximal clique). The subgraph represents a subset of the criminals which should be complete i.e., each distinct nodes/criminal are adjacent/know one another. We are looking for such clique/subgraph where we can not extend it by including one more adjacent vertex, hence, maximial. We can formulate the problem as an optimization problem as follows. Let x represents the adjacency matrix ($n \, x \, n$ binary matrix) of the graph where $x_{ij} = 1$ if criminal i knows criminal j. Otherwise, $x_{ij} = 0$. Let y be the decision variable (binary vector of length n) where $y_i = 0$ is criminal i is in the maximum clique. The objective function then becomes

$$Objective\ Function: max \sum_{j=1}^{n} y_j$$

The constraints for this problem is to make sure that members of the subgraph knows one another (complete subgraph)

$$Constraints: \; y_i + y_j \leq 1 \; if \; x_{ij} = 0$$

(b) Here we are looking for the minimal set of vertices from which we can reach all other vertices i.e., *dominating set*. The dominating set D is a subset of the graph such that every vertex not in D is adjacent to at least one member of D. We should also make sure that each two vertices in D are not connected i.e., independent set. Thus, we are trying to find the maximal independent set of the graph which is also a minimal dominating set (reference: https://en.wikipedia.org/wiki/Dominating_set#Independent_domination).
We use y as the decision variable where $y_i = 1$ means that node i is in the maximal independent set (paid informant). Thus, the objective function becomes

$$Objective\ Function: min \sum_{j=1}^{n} y_j$$

The constraints are

$$Constraints: \left(\sum_{j=1}^{n} x_{ij} y_j \right) + y_i \geq 1, \; for \; i = 1 \ldots n$$

The first term of the constraint equation (sum over n) makes sure that vertex i is connected to at least of member of the dominating set, where the second term ensures that it is already in the dominating set.

Problem 2:

Let n be the number of vertices of the graph G and is the adjacency matrix of the graph x_{ij}. Let v be the decision variable such that $v_i = 1$ if vertex i is in the vertex-cover. The objective function is

$$Objective\ Function: min \sum_{j=1}^{n} v_j$$

The constraints are

$$Constraints: \ v_i + v_j \geq 1 \ if \ x_{ij} = 1$$

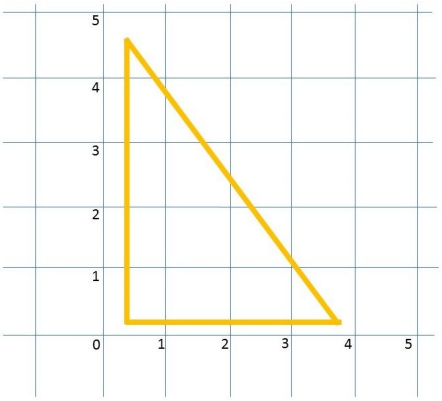
Explanation: v_i is given a value in order to mark whether a vertex is going to be part of the vertex-cover. This will start on an arbitrary vertex. x_{ij} is given the value one if an edge exists between two vertices. The objective function is that we want to minimize the sum of v_i because this will result in the smallest number of vertices in the vertex-cover. The constraint tells us that if there is an edge connecting two vertices, the sum of those two vertices has to be at least one. This guarantees that each edge will be touching a minimum of one vertex in the vertex-cover.

Let M be the maximum matching of G, and let S be the minimum vertex cover of G. Suppose there are m edges in M. Then for each of these m edges in M, there are endpoints on each of these edges in S. Because this is a maximum matching, M does not have any endpoints that are shared. Since these endpoints are not shared, for a vertex cover to exist, there must be at least m vertices to cover these m edges. Therefore, it must hold true that

$$max\{|M| : M \ is \ a \ matching \ of \ G\} \leq min\{|S| : S \ is \ a \ vertex \ cover \ of \ G\}$$

Problem 3:

In linear programming, the constraints in an integer program form a polytope. Feasible set is given by the set of all integer-valued points within the polytope and not the entire polytope. We can generate linear programming relaxation from taking the same objective function and constraints but with the requirement that variables are integers is replaced by appropriate continuous constraints. To show an example of an integer programming problem which has no feasible integer solutions but its LP relaxation has a feasible set in $\Re^2$ we draw the feasible region as shown the figure where non of it is heads lies on the grid (integer). For such feasible region, the minimum as well as the maximum value of the function is not an integer, thus no feasible integer solution. However, relaxing the problem to be in $\Re^2$ will give the optimal solution.



Problem 4:

We can construct a complete weighted graph *G** such that an edge *u*-*v* in *G** will be given a weight of 1 if it exists in the original *G* graph, otherwise the edge will be given a weight of 2. We can then run the TSP algorithm/software on *G** to get the tour of minimum weight. If the total distance/weight travelled is equal *n* where *n* is the total number of vertices ($n = |V|$) and it contains all the vertices, this means that the tour has all the edges of weight 1 which implies that we find the cycle that visits each vertex exactly once. If the total distance travelled is more than *n*, then *G** does not have a TSP solution of weight *n* which means that TSP has to involve at least one edge of weight 2 which implied it is not possible to cycle all the vertices in G once and come back to the starting vertex.

In conclusion, we can use the TSP algorithm/software to get the cycle that visits each vertex exactly once by running the algorithm/software on *G** and verify that the solution we get have a weight/total distance travelled of *n*.

Problem 5:

The problem can be formulated as linear programming and written as follows

$Variables: x_T, x_N, x_P, x_A$

$Maximize: 5000x_T + 8500x_N + 2400x_P + 2800x_A$

$Constraints:$

- $x_P + x_A \geq 5$
- $290x_P + 380x_A \leq 1800$
- $800x_T + 925x_N + 290x_P + 380x_A \leq 8000$
- $0 \leq x_T \leq 12$
- $0 \leq x_N \leq 5$
- $0 \leq x_P \leq 25$
- $0 \leq x_A \leq 20$
- $x_T, x_N, x_P, x_A \in I\ (integer)$

where x_T, x_N, x_P, x_A are the number of ads run per week for TV spot, newspaper, radio (prime time) and radio (afternoon), respectively.

Solving the problem using SCIP, we get the following optimal solution

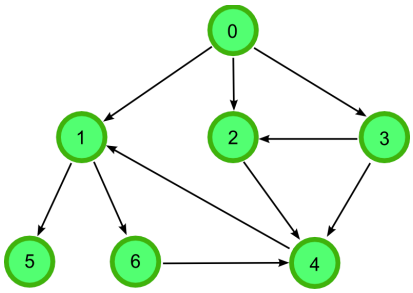
$Objective\ value:\ 66900\ views$ which is the number of audience reach for all ads per week

Solution:

- $x_T = 2$
- $x_N = 5$
- $x_P = 6$
- $x_A = 0$

Problem 6:

(a) Here we try to prove that if a directed graph G has a topological ordering, then G does not contain any directed cycle. We will try to prove this by contradiction. We start by assuming that G has a topological order $y_1, ..., y_n$. Suppose by contradiction that G has a directed cycle such that y_i is the lowest-indexed node in the cycle and y_j is the node before y_i in the cycle, thus we get an edge connecting y_i and y_j. Note that we have chosen i such that $i < j$. We also have y_i and y_j forming an edge and $y_1, ..., y_n$ are in topological order which means $i > j$ which leads to contradiction. Thus, if G has a topological order, then there is no directed cycles in it.



A simple model that detect cycles in a directed graph and returns yes if there is a cycle would rely on Depth First Search (DFS) on the graph. DFS starts with a vertex and keep exploring all the down path starting from this vertex one complete path at a time.
For example, for the graph on the right, starting from vertex 0, a complete path would be 0→2→4. If there is a cycle in the graph, it would lead to coming back to the starting vertex or already-visited vertex while exploring the path. Thus, our model for detecting cycles will work by solving DSF for all vertices. For each path, it will recorder the already-visited vertices in this path. For a newly-visited vertex, it will be checked against already-visited vertices. If it is one of the already-visited vertices, then the graph has a cycle. For example, start with vertex 1, we will have 1 in the already-visited vertices. We then move to 6 and add it to already-visited vertices, and then to 4 and do the same. The vertex 4 will lead us to 1 which is in the already-visited vertices list, and our model reports that there is a cycle.

(b) Using MATLAB, we were able to create a graph for all the math courses offered and their respective (math-only) prerequisites. The prerequisites were obtained from UC Davis 2018-2019 Catalogue website (published on April 30th, 2018) for courses *MATH21A* to *MAT189*. Figure (1) shows the *MathC* graph. A directed edge in the graph from node ***A*** to node ***B*** suggests that course ***A*** is a prerequisite for course ***B***. Optional edges are shown in blue. Since some courses have optional prerequisites that form a group, we created additional nodes for such courses. These additional node are shown with bigger circles in Figure (1). For example courses MAT114, MAT124, MAT128B, MAT128C, MAT135A, MAT135B, MAT141, and MAT115B all require taking either MAT22A or MAT67 in addition to another course. For that, we created a node *MAT22A*|*MAT67* (the vertical bar **|** reads as *or*) that points to all mentioned courses. The wights on the edges are one.
MathC graph has 56 nodes for which 50 nodes represent unique course and 5 nodes represent set of grouped courses. *Math C* graph has 95 edges.

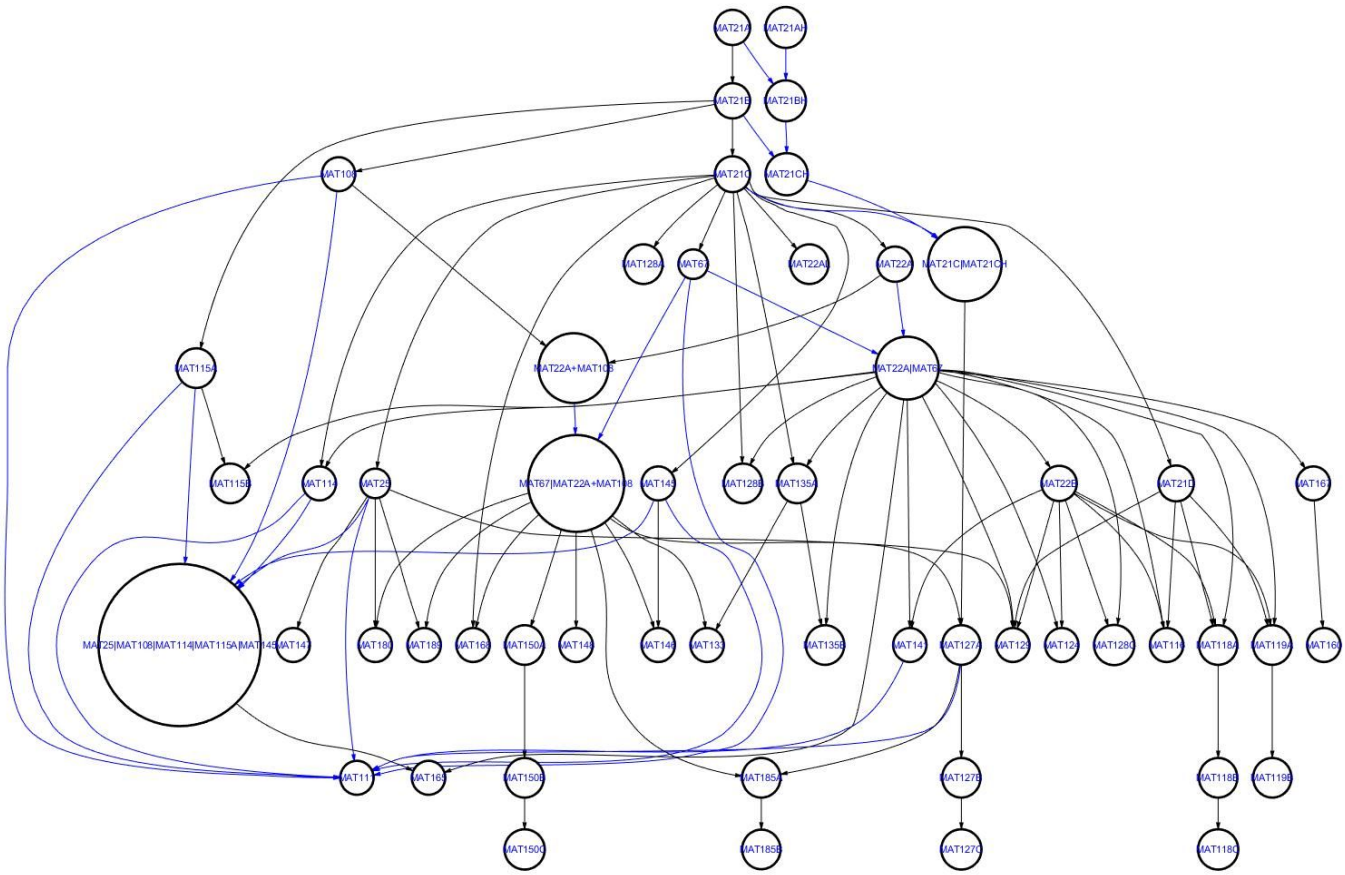


Figure (1): The *MathC* graph showing all the math prerequisites for undergrad offered in the math department starting with MAT21 and beyond. Unique class are shown in small circle

(c) For this problem, our optimization model is as follows
The decision variable is a matrix xij

For this problem, our optimization model is as follows. The decision variable is a matrix $x_{i,j}$ with number of rows equal to number of available courses nC and number of columns equal to the available number of quarter nQ (explained later). The matrix is binary where $x_{i,j} = 1$ means that the student should take course i in quarter number j. The objective function is to minimize the total number of quarters i.e., graduate as soon as possible. This can be written as

$$Objective\ Function: min \sum_{i=1}^{nC} \sum_{j=1}^{nQ} j x_{i,j}$$

We should let the number of quarter be decided by the algorithm, for that we set it up to 20 thus we allow the student to graduate within 5 years which is more than enough.

The constraints for this problem are:

- The course should be take only once: $\sum_{j=1}^{nQ} x_{i,j} <= 1, for\ i: 1, ..., nC$

- No more than three courses to be taken per quarter: $\sum_{i=1}^{nC} x_{i,j} <= 3, for\, j : 1, ..., nQ$

- If a course is required by the major, it must be take: $\sum_{j=1}^{nQ} x_{i,j} = 1, for\, i \varepsilon \{required\ courses\}$

- If the student is required to take one of set of optional courses, the constraint is $\sum_{i=1}^{nOp} \sum_{j=1}^{nQ} x_{i,j} >= 1, nOp = \{set\ of\ optional\ courses\}$

- If there is an edge from vertex *A* to vertex *B* in *mathC* graph with a black edge i.e., course *A* is a prerequisite (non-optional) for course *B*, the constraints is $argmax_j x_{A,j} < argmax_j x_{B,j}$. Since ZIMPL does not allow *agrmax* in the constraints, we can rewrite this as $\sum_{j=1}^{nQ} j x_{A,j} < \sum_{j=1}^{nQ} j x_{B,j}$

- If the student is allowed to take either course A or B as prerequisite for course C, we can enforce such constraint as using three inequalities

$$\sum_{j=1}^{nQ} x_{c,j} <= \sum_{j=1}^{nQ} x_{A,j} + \sum_{j=1}^{nQ} x_{B,j}$$

$$\left(1 - \sum_{j=1}^{nQ} x_{B,j}\right) \sum_{j=1}^{nQ} x_{A,j} \sum_{j=1}^{nQ} j x_{A,j} <= \left(1 - \sum_{j=1}^{nQ} x_{B,j}\right) \sum_{j=1}^{nQ} x_{A,j} \sum_{j=1}^{nQ} j x_{c,j}$$

$$\left(1 - \sum_{j=1}^{nQ} x_{A,j}\right) \sum_{j=1}^{nQ} x_{B,j} \sum_{j=1}^{nQ} j x_{B,j} <= \left(1 - \sum_{j=1}^{nQ} x_{A,j}\right) \sum_{j=1}^{nQ} x_{B,j} \sum_{j=1}^{nQ} j x_{c,j}$$

The first equation is only meaningful if the student wants to take course C or it is required by the major and it ensures that either (or both) course A and B are also take. The second and third equation are symmetric where they ensure if course B is not take (first parenthesis) then course A must be take (first summation after the parenthesis) such that it is taken in a quarter before taking course C (second summation after the parenthesis).

We applied our model on the course requirements for A.B. in secondary teaching. The following table shows the results for what courses to take each quarter. All the math-only requirements are fulfilled within three years and a quarter. Additionally, all the prerequisites imposed by *MathC* graph are satisfied.

Q1	***Q2***	***Q3***	***Q4***	***Q5***	***Q6***	***Q7***	***Q8***	***Q9***	***Q10***
MAT21A	MAT145	MAT128A	MAT108	MAT119A	MAT115B	MAT114	MAT118C	MAT150A	MAT127A
MAT21B	MAT21D	MAT22AL	MAT111	MAT141	MAT118A	MAT116	MAT119B	MAT180	MAT127B
MAT21C	MAT25	MAT67	MAT115A	MAT22B	MAT135A	MAT118B	MAT124	MAT189	MAT127C

Comments of MathC graph: From the plot of the graph we can say that course MAT21C is the most important as it has a lot of out-going edges. This is verified by computing the centrality

betweennessof the graph which is maximum at MAT21C course. The least attractive course if MAT129 which has a lot of prerequisites and it is not required by any other course.

Problem 7:

Basic formulation:

We can implement Sudoku solver as binary integer programming problem. Instead of visualize the problem as a 9-by-9 square, we view is as 9-by-9-by-9 cube where the added layers represent one of the possible solution (1-9) for that square cell. We represent this with the variable $x_{i,j,k}$ where the first two indices (i and j) represents the horizontal and vertical location (from 1 to 9) and the third one represents either the number k assigned to this cell (from 1 to 9).

The constraints:

Since each cell in the square grid should have only one solution while other solution should be zeros, we can impose this by summing over all possible k for this cell. This can be written as

$$\sum_{k=1}^{9} x_{i,j,k} = 1, \text{ for } i = 1,2,...,9 \text{ and } j = 1,2,...,9 \qquad (1)$$

This will gives 9-by-9 equality constraints.

Since each row and column contain just one value from 1 to 9, we can impose this constraint in a similar fashion while changing the indices as follows

$$\sum_{j=1}^{9} x_{i,j,k} = 1, \text{ for } i = 1,2,...,9 \text{ and } k = 1,2,...,9 \qquad (2)$$

$$\sum_{i=1}^{9} x_{i,j,k} = 1, \text{ for } k = 1,2,...,9 \text{ and } j = 1,2,...,9 \qquad (3)$$

The constraint (2) here makes sure that each row has one of the nine values by fixing i and k indices and looping over j . In cooperation with constraint (1), this will ensure that each cell in each row has a unique value (different from other cells in the same row). Same idea is applied using constraint(3) but for columns.

Same idea can capture the unique assignments in the nine partitions $A_1, A_2, ..., A_9$. However, we will need to loop over the partitions i.e., 3-by-3 grid. This can be written as

$$\sum_{i=1}^{3}\sum_{j=1}^{3} x_{i+X,j+Y,k} = 1, \text{ for } k = 1,2,...,9,\ X = 0,3,6 \text{ and } Y = 0,3,6 \qquad (4)$$

X and Y here represents a shift in i and j indices which means we are picking different partition for each X and Y combination which total to 6-by-6 permutation representing all different partitions. In constraint (4), we get 9-by-6-by-6 equality constraints.

The final constraints are the given filled entries. If entry (i,j) is given the number s, we can impose this by constraining the entry $x_{i,j,s} = 1$. Note that constraints (1) will make sure this entry will not change as other k value will be zeroed.

Objective function:

With this setup, we are not optimizing for an objective function. We are just looking for one feasible solution that respect all the given constraints i.e., feasibility problem.

Solution of SUDOKU Number 1

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

Solution of SUDOKU Number 2

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

Solution of SUDOKU Number 3

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

Figure (2): Solution of the SUDOKU game solved using ZIMPL/SCIP and plotted using MATLAB where the gray color indicates the clue cells and white cells are the solution

Solution:

Figure (2) shows the solution as outputted from SCIP. The solution is printed using MATLAB code that feeds into the solution as displayed from SCIP (*'display solution'*) and the clue cells. All files attached with the homework. First SUDOKU (from the left) took *0.01* sec to solve while the second and third took *0.01* and *0.02* sec to solve, respectively. It was hard to determine which one took longer as these timing were not consistent. We experimented, however, with what extreme case where only one clue cell was given and it took *0.39* sec to solve.

The model we have can be used to check if a SUDOKU problem has more than one solution as follows. We run the model/code using the clues we have as input. We store this solution and pick arbitrary cell $x_{i,j}$ that is not part of the clue cells. If s is the number that our model gave for the cell $x_{i,j}$ then we will have $x_{i,j,s} = 1.$ Now, in order to check if there is another solution for the same set of clues, we run the same model/code using the same set of clue and constraints but add another constraints such that $x_{i,j,s} = 0.$ That way we let $x_{i,j}$ take any other value that it might has if there is another solution for the same set of clues. If our models/code showed no solution, we move on to another cell (that is not part of the clues) and repeat. Otherwise, we will have another solution for the same set of clues.

The simplest bad SUDOKU that might have more than one solution is the one that has only one clue. We applied out technique on the input clue of $x_{5,5,5} = 5$ which gave us the solution shown in Figure (3) left. In the next run we restricted $x_{1,1,2} = 0$ which gave use a different solution shown in Figure (3) right.

Solution of SUDOKU only one

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

Solution of SUDOKU only one

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

Figure (3): Two different solution for the same input clue (X[5,5]=5) using our model

After some research, we found out that there is no 16-clue SUDOKU with a unique solution i.e., the minimum number of clues have to be given to ensure the existence of a unique solution is 17 [1].

References:

[1] *McGuire, Gary, Bastian Tugemann, and Gilles Civario. "There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration." Experimental Mathematics 23, no. 2 (2014): 190-217.*