DSA/ISE 5113 Advanced Analytics and Metaheuristics

Homework #5 Group 5

Yashasvi Musku, Sai Abhinov Chowdary Katragadda, Vivek Satya Sai Veera Venkata Talluri, Vignesh Murugan

Question 1: Boomer Sooner Air Services

a)

1) Sets :

There is a need to declare sets for flight plans

set fltplan; #The Airports are KCID1 KACK KMMU KBNA KTUL KCID2

2) Parameters :

The parameters for this model are as follows:

FuelCost :

The fuel cost at each airport in dollars.

RampFees:

The ramp fees at each airport in dollars.

MinGaltoWaivFees:

The minimum gallons to be purchased to waive the ramp fees off at each airport.

FuelBurn:

The fuel burned by Aircraft for each leg.

passengers:

The number of passengers with which the aircraft takesoff at each airport.

MaxRampWeight:

The maximum Ramp weight limit set by FAA.

MaxLandingWeight:

The maximum landing weight limit set by FAA.

BOW:

The basic operating weight of aircraft.

FuelTankCapacity:

The fuel tank capacity of the aircraft.

MinLandingFuel:

The minimum fuel with which the aircraft is to be landed in case of emergency.

ReturnFuelLimit:

The default fuel level of the aircraft at KCID.

M = FuelTankCapacity – MinLandingFuel:

The M constant for limiting fuel purchase to tank capacity

3) Decision Variables :

The decision variables are:

var x{ fltplan} binary; :

The value of x is 1 if the airport is chosen for filling fuel otherwise it is zero. Here decision variable x is either 0 or 1.

var y {fltplan} binary; :

The value of y is 1 if the ramp fees of the airport is waived off due to purchase of fuel above the prescribed limit at that airport, otherwise it is zero.

var FuelLeft {fltplan} >= 0; :

The fuel left in the aircraft at the arrival destination is stored in this variable for that destination.

var FuelBefore {fltplan} >= 0;:

The fuel in the aircraft before takeoff from airport is stored in this variable for the airport from which it takes off.

var FuelAdd {fltplan} >= 0;:

The fuel bought for the aircraft at the particular airport .

var LandingWeight {fltplan} >= 0;:

The landing weight of the aircraft at each destination airport.

var RampWeight {fltplan} >= 0;:

The ramp weight of the aircraft at each airport.

var Cost {fltplan} >= 0;:

The total cost of fuel being bought and the ramp fees at each airport.

4) Objective Function :

The objective function is to plan the five leg flight travel with minimum total cost.

It is declared in the AMPL code as follows:

minimize TotFuelCost: sum {i in fltplan} Cost[i];

5) Constraints :

The constraints for the given problem are as follows:

1.The fuel filling constraint at each airport:

subject to FuelFilling {i in fltplan}: FuelBefore[i] = FuelLeft[i] + FuelAdd[i];:

The sum of fuel left in aircraft at arrival and the fuel bought at airport gives the fuel before takeoff at each airport.

2. The fuel tank capacity limit constraint:

subject to FuelTankLimit {i in fltplan}: FuelBefore[i] <= FuelTankCapacity;:

The total fuel in aircraft should not exceed the total tank capacity.

3.The Emergency Fuel limit constraint:

subject to EmergencyFuelLimit {i in fltplan}: FuelLeft[i] >= MinLandingFuel;:

The minimum extra fuel to be stocked in the aircraft for emergency purposes on arrival at destination , in case of change in destination airport due to bad weather or other emergency conditions.

4. Fuel in tank before and after the whole trip constraints:

subject to FuelatBegin : FuelLeft['KCID1'] = ReturnFuelLimit;:

The fuel with which the aircraft takes off at Cedar Rapids.

subject to FuelatEnd: FuelBefore['KCID2'] >= ReturnFuelLimit;:

The fuel tank level at which the aircraft is to be stocked on arrival at Cedar Rapids.

5.The Ramp Fees waive Constraint:

subject to RmpFees {i in fltplan}: y[i] <= (FuelAdd[i]/(6.7*MinGaltoWaivFees[i]));:

The decision of ramp fees waive off is made based on the amout of fuel purchased at each airport. It is mathematicall represented as:

$$y[i] \; \forall \; i \in fltplan \leq \frac{(Fuel\ bought\ at\ airport\ in\ pounds)}{(6.7 * Minimum\ fuel\ to\ be\ bought\ for\ ramp\ fees\ waive)}$$

6. The total Cost Calculation Constraint:

subject to Total_Cost {i in fltplan}: Cost[i] = RampFees[i] *(1-y[i]) +FuelCost[i]*FuelAdd[i]/6.7;:

The cost inccurring at each airport for fuel purchase and ramp fees are calculaed using this constraint. It is mathematically represented as follows:

$$Cost[i] \;\; = RampFees[i] * (1 - y[i]) + \; FuelCost[i] * FuelAdd[i]/6.7 \;\; \forall \; i \; \in fltplan$$

7.The limitation of fuel purchase constraint

subject to FuelLimit {i in fltplan}: FuelAdd[i] <= M * x[i] ;

subject to Fuelpurch {i in fltplan}: FuelAdd[i] >= 0.1*x[i];:

The limitation of fuel to be bought at each airport based on the fuel tank capacity of aircraft.The second constraint here limits the value of x[i] to zero if value of FuelAdd[i] is zero.

8.The Landing weight at each destination constraint:

subject to LWKCID1: LandingWeight['KCID1'] = FuelLeft['KCID1'] + BOW + 200*0;
subject to LWKACK: LandingWeight['KACK'] = FuelLeft['KACK'] + BOW + 200 * passengers['KCID1'] ;
subject to LWKMMU:LandingWeight['KMMU']=FuelLeft['KMMU']+BOW+ 200*passengers['KACK'] ;
subject to LWKBNA:LandingWeight['KBNA']= FuelLeft['KBNA']+BOW + 200*passengers['KMMU'] ;
subject to LWKTUL: LandingWeight['KTUL'] = FuelLeft['KTUL'] + BOW + 200 * passengers['KBNA'] ;
subject to LWKCID2: LandingWeight['KCID2'] = FuelLeft['KCID2'] + BOW + 200*passengers['KTUL'] ;

The above constraints represent the calculations of the landing weight of aircraft which is the sum of the weight of fuel left at the time of landing, basic operating weight of aircraft and additional 200 pounds for each passenger in the plane during landing which is always equal to the number of passengers during takeoff at previous airport.

9.The Ramp weight at each airport constraint:

subject to RWKCID1: RampWeight['KCID1'] = FuelLeft['KCID1'] + FuelAdd['KCID1'] + BOW + 200 * 0 ;
subject to RWKACK: RampWeight['KACK'] =FuelLeft['KACK']+FuelAdd['KACK'] +BOW + 200*passengers['KACK'] ;
subject to RWKMMU :RampWeight['KMMU'] = FuelLeft['KMMU'] + FuelAdd['KMMU'] + BOW + 200 * passengers['KMMU'] ;

subject to RWKBNA: RampWeight['KBNA'] = FuelLeft['KBNA'] + FuelAdd['KBNA'] + BOW + 200 * passengers['KBNA'] ;
subject to RWKTUL: RampWeight['KTUL'] = FuelLeft['KTUL'] + FuelAdd['KTUL'] + BOW + 200 * passengers['KTUL'];
subject to RWKCID2: RampWeight['KCID2'] = FuelLeft['KCID2'] + FuelAdd['KCID2'] + BOW + 200 * passengers['KCID2'];

The above constraints represent the calculations of the ramp weight of aircraft which is the sum of the weight of fuel left at the time of landing, basic operating weight of aircraft, fuel loaded(bought) at airport and additional 200 pounds for each passenger in the plane ready for takeoff from that airport, in the aircraft.

10.The maximum landing weight constraint:

subject to MaxLW {i in fltplan}: LandingWeight[i] <= MaxLandingWeight;

The maximum landing weight constraint for each leg should not exceed the FAA norms.

11.The maximum ramping weight constraint:

subject to MaxRW {i in fltplan}: RampWeight[i] <= MaxRampWeight;:

The maximum ramping weight constraint for flight at any airport should not exceed the FAA norms.

12. The fuel consumption in each leg constraint:

subject to Leg1_KACK: FuelLeft['KACK'] = FuelBefore['KCID1']-FuelBurn['KCID1'];

subject to Leg2_KMMU: FuelLeft['KMMU'] = FuelBefore['KACK']-FuelBurn['KACK'];

subject to Leg3_KBNA: FuelLeft['KBNA'] = FuelBefore['KMMU']-FuelBurn['KMMU'];

subject to Leg4_KTUL: FuelLeft['KTUL'] = FuelBefore['KBNA']-FuelBurn['KBNA'];

subject to Leg5_KCID2: FuelLeft['KCID2'] = FuelBefore['KTUL']-FuelBurn['KTUL'];

The above constraints represent the calculations of the fuel left in aircraft after each leg which is the difference between the fuel before leg and the fuel consumed during the leg.

The AMPL Code for the above model is as follows:

group5_HW5_p1ai.mod | group5_HW5_p1a.dat | group5_HW5_p1aii.mod

```
#AMPL HW_5 _1a DSA/ISE5113
#Authors  Sai Abhinov Chowdary Katragadda, Yashasvi Mususku,
#         Vivek Satya Sai Veera Venkata Talluri, Vignesh Murugan
#Date: 03/16/2024
# Problem Description:
# This AMPL script models flight plan decision problem where the objective is to
# minimize fuel cost by tankering
# Reset AMPL environment and specify solver options:
reset;
option solver cplex;
option cplex_options 'sensitivity';

#sets
set fltplan;   #The flight destinations in order

# parameters

param FuelCost {fltplan};               # fuel cost at each destination in dollars
param RampFees {fltplan};               # ramp fees at each destination in dollars
param MinGaltoWaivFees {fltplan};       # Mininmum fuel purchase in gallons to waive Rampfees
param FuelBurn {fltplan};               # Fuel consumed in each trip
param passengers {fltplan};             # defining the number of paasengers

param MaxRampWeight;
param MaxLandingWeight;
param BOW;
param FuelTankCapacity;
param MinLandingFuel;
param ReturnFuelLimit;

param M = FuelTankCapacity - MinLandingFuel;   # Fuel tank Capacity - Min Landing Fuel

# Decision Variables

var x {fltplan} binary;   # airport chosen for filling
var y {fltplan} binary;   # airport ramp fee waive off

var FuelLeft {fltplan} >= 0;
var FuelBefore {fltplan} >= 0;
var FuelAdd {fltplan} >= 0;

var LandingWeight {fltplan} >= 0;
var RampWeight {fltplan} >= 0;
```

group5_HW5_p1ai.mod | group5_HW5_p1a.dat | group5_HW5_p1aii.mod

```
# Decision Variables

var x {fltplan} binary;   # airport chosen for filling
var y {fltplan} binary;   # airport ramp fee waive off

var FuelLeft {fltplan} >= 0;
var FuelBefore {fltplan} >= 0;
var FuelAdd {fltplan} >= 0;

var LandingWeight {fltplan} >= 0;
var RampWeight {fltplan} >= 0;
var Cost {fltplan} >= 0;


# Objective_Function

minimize TotFuelCost: sum {i in fltplan} Cost[i];


# Constraints

# 1.fuel filling at airport
subject to FuelFilling {i in fltplan}: FuelBefore[i] = FuelLeft[i] + FuelAdd[i];


# 2.fuel tank capacity limit
subject to FuelTankLimit {i in fltplan}: FuelBefore[i] <= FuelTankCapacity;


# 3.Emergency Fuel limit
subject to EmergencyFuelLimit {i in fltplan}: FuelLeft[i] >= MinLandingFuel;


# 4.Fuel in Tank before and after Trips constraint
subject to FuelatBegin : FuelLeft['KCID1'] = ReturnFuelLimit;
subject to FuelatEnd: FuelBefore['KCID2'] >= ReturnFuelLimit;

#5. Ramp Fees waive const
subject to RmpFees {i in fltplan}: y[i] <= (FuelAdd[i]/(6.7*MinGaltoWaivFees[i]));

# 6.calculating the Total cost (= fuel + ramp fees)
subject to Total_Cost {i in fltplan}: Cost[i] = RampFees[i] *(1-y[i]) + FuelCost[i]*FuelAdd[i]/6.7;
```

group5_HW5_p1ai.mod | group5_HW5_p1a.dat | group5_HW5_p1aii.mod | group5_HW5_p1b.mod

```
#7. Limitation on Fuel Purchase
subject to FuelLimit {i in fltplan}: FuelAdd[i] <= M * x[i] ;
subject to Fuelpurch {i in fltplan}: FuelAdd[i] >= 0.1*x[i];
 # the Fuelpurch is an auxilary constraint to get perfect x decision variable values

# 8.Landing weight at each destination
subject to LWKCID1: LandingWeight['KCID1'] = FuelLeft['KCID1'] + BOW + 200*0;
subject to LWKACK: LandingWeight['KACK'] = FuelLeft['KACK'] + BOW + 200 * passengers['KCID1'] ;
subject to LWKMMU: LandingWeight['KMMU'] = FuelLeft['KMMU'] + BOW + 200 * passengers['KACK'] ;
subject to LWKBNA: LandingWeight['KBNA'] = FuelLeft['KBNA'] + BOW + 200 * passengers['KMMU'] ;
subject to LWKTUL: LandingWeight['KTUL'] = FuelLeft['KTUL'] + BOW + 200 * passengers['KBNA'] ;
subject to LWKCID2: LandingWeight['KCID2'] = FuelLeft['KCID2'] + BOW + 200 * passengers['KTUL'] ;

# 9.Ramp weight at each airport
subject to RWKCID1: RampWeight['KCID1'] = FuelLeft['KCID1'] + FuelAdd['KCID1'] + BOW + 200 * 0 ;
subject to RWKACK: RampWeight['KACK'] = FuelLeft['KACK'] + FuelAdd['KACK'] + BOW + 200 * passengers['KACK']  ;
subject to RWKMMU: RampWeight['KMMU'] = FuelLeft['KMMU'] + FuelAdd['KMMU'] + BOW + 200 * passengers['KMMU'] ;
subject to RWKBNA: RampWeight['KBNA'] = FuelLeft['KBNA'] + FuelAdd['KBNA'] + BOW + 200 * passengers['KBNA'] ;
subject to RWKTUL: RampWeight['KTUL'] = FuelLeft['KTUL'] + FuelAdd['KTUL'] + BOW + 200 * passengers['KTUL'];
subject to RWKCID2: RampWeight['KCID2'] = FuelLeft['KCID2'] + FuelAdd['KCID2'] + BOW + 200 * passengers['KCID2'];

# 10.maximum landing weight constraint
subject to MaxLW {i in fltplan}: LandingWeight[i] <= MaxLandingWeight;

# 11.maximum ramping weight constraint
subject to MaxRW {i in fltplan}: RampWeight[i] <= MaxRampWeight;

# 12.fuel consumption in each leg
subject to Leg1_KACK: FuelLeft['KACK'] = FuelBefore['KCID1']-FuelBurn['KCID1'];
subject to Leg2_KMMU: FuelLeft['KMMU'] = FuelBefore['KACK']-FuelBurn['KACK'];
subject to Leg3_KBNA: FuelLeft['KBNA'] = FuelBefore['KMMU']-FuelBurn['KMMU'];
subject to Leg4_KTUL: FuelLeft['KTUL'] = FuelBefore['KBNA']-FuelBurn['KBNA'];
subject to Leg5_KCID2: FuelLeft['KCID2'] = FuelBefore['KTUL']-FuelBurn['KTUL'];


#Commands-------------------------------------------------------------------------------------------
data group5_HW5_p1a.dat;
solve;

display TotFuelCost;
display x,y, FuelLeft, FuelBefore, FuelAdd;
```

The AMPL .DAT file is as follows:

group5_HW5_p1ai.mod | group5_HW5_p1a.dat | group5_HW5_p1aii.mod

```
data;
set fltplan := KCID1 KACK KMMU KBNA KTUL KCID2;


param:     FuelCost    RampFees    MinGaltoWaivFees    FuelBurn    passengers :=
    KCID1     4           0           infinity          5100          2
    KACK     8.32        800           600             2200          4
    KMMU     8.99        750           500             4700          8
    KBNA     6.48        600           650             3800          8
    KTUL     9.27        800           500             3600          8
    KCID2    4           0           infinity            0           0;

param MaxRampWeight = 36400;
param MaxLandingWeight = 31800;
param BOW = 22800;
param FuelTankCapacity = 14000;
param MinLandingFuel = 2500;
param ReturnFuelLimit = 7000;
```

The ouput for this model is as follows:

```
ampl: model group5_HW5_p1ai.mod
CPLEX 20.1.0.0: sensitivity
CPLEX 20.1.0.0: optimal integer solution; objective 17251.49254
6 MIP simplex iterations
0 branch-and-bound nodes
absmipgap = 3.63798e-12, relmipgap = 2.10879e-16

suffix up OUT;
suffix down OUT;
suffix current OUT;
TotFuelCost = 17251.5

:       x   y FuelLeft FuelBefore FuelAdd    :=
KACK    1   0    8500      9400      900
KBNA    1   1    2500      9900     7400
KCID1   1   0    7000     13600     6600
KCID2   1   0    2500      7000     4500
KMMU    0   0    7200      7200        0
KTUL    0   0    6100      6100        0
;

ampl:
```

The above output can be represented as :

Airport	Fuel Left after landing in	Fuel Add	Fuel Before takeoff from	x	y
KCID1	7000	6600	13600	1	0
KACK	8,500	900	9,400	1	0
KMMU	7200	0	7200	0	0
KBNA	2500	7400	9900	1	1
KTUL	6100	0	6100	0	0
KCID2	2500	4500	7000	1	0

The optimal Total Fuel Cost according to the above plan is **17,251.5** dollars where the fuel values are in pounds. The ramp fees at Nasville airport is waived due to fuel purchase.

For the fueling plan without tankering we have to change constraint 3 as follows:

```
# 3.Emergency Fuel limit without tankering
subject to EmerFuelLimit1: FuelLeft['KACK'] = MinLandingFuel;
subject to EmerFuelLimit2: FuelLeft['KMMU'] = MinLandingFuel;
subject to EmerFuelLimit3: FuelLeft['KBNA'] = MinLandingFuel;
subject to EmerFuelLimit4: FuelLeft['KTUL'] = MinLandingFuel;
subject to EmerFuelLimit5: FuelLeft['KCID2'] >= MinLandingFuel;
```

The output for this change in model is as follows:

```
ampl: model group5_HW5_p1aii.mod
CPLEX 20.1.0.0: sensitivity
CPLEX 20.1.0.0: optimal integer solution; objective 22139.25373
0 MIP simplex iterations
0 branch-and-bound nodes

suffix up OUT;
suffix down OUT;
suffix current OUT;
TotFuelCost = 22139.3

:       x   y FuelLeft FuelBefore FuelAdd    :=
KACK    1   0    2500      4700     2200
KBNA    1   0    2500      6300     3800
KCID1   1   0    7000      7600      600
KCID2   1   0    2500      7000     4500
KMMU    1   1    2500      7200     4700
KTUL    1   1    2500      6100     3600
;
```

The above output can be represented as :

Airport	Fuel Left after landing in	Fuel Add	Fuel Before takeoff from	x	y
KCID1	7000	600	7600	1	0
KACK	2,500	2200	4,700	1	0
KMMU	2500	4700	7200	1	1
KBNA	2500	3800	6300	1	0
KTUL	2500	3600	6100	1	1
KCID2	2500	4500	7000	1	0

The Total Fuel Cost for no tankering plan is **22,139.3** dollars where the fuel values are in pounds. The ramp fees at Morriston airport and Tulsa airport is waived due to fuel purchase.

CONCLUSION:

Thus the Optimal Fuel cost and no tankering cost vary by:

$22139.3 – $17251.5 = **$4887.8**

Thus by tankering plan the pilot can save $4887.8 for this travel.

Question 1: Boomer Sooner Air Services

b)

To modify the model such that if gas is to be bought it should be more than 200 gallons, then the following change in constraint will update the model to the new goal:

```
#7. Limitation on Fuel Purchase
subject to FuelLimit {i in fltplan}: FuelAdd[i] <= M * x[i] ;
subject to Fuelpurch {i in fltplan}: FuelAdd[i] >= 200*6.7*x[i];
```

The output for the above changed model is as follows:

```
ampl: model group5_HW5_p1b.mod
CPLEX 20.1.0.0: sensitivity
CPLEX 20.1.0.0: optimal integer solution; objective 17372.32836
12 MIP simplex iterations
0 branch-and-bound nodes

suffix up OUT;
suffix down OUT;
suffix current OUT;
TotFuelCost = 17372.3

:       x   y FuelLeft FuelBefore FuelAdd    :=
KACK    1   0    8500      9840     1340
KBNA    1   1    2940      9900     6960
KCID1   1   0    7000     13600     6600
KCID2   1   0    2500      7000     4500
KMMU    0   0    7640      7640        0
KTUL    0   0    6100      6100        0
;

ampl:
```

The above output can be represented as follows:

Airport	Fuel Left after landing in	Fuel Add	Fuel Before takeoff from	x	y
KCID1	7000	6600	13600	1	0
KACK	8,500	1340	9,840	1	0
KMMU	7640	0	7640	0	0
KBNA	2940	6960	9900	1	1
KTUL	6100	0	6100	0	0
KCID2	2500	4500	7000	1	0

Thus the optimal Total cost increased to **$17372.3** from **$17251.5** due to the additional purchase of 440 pounds of fuel bought at Nantucket Airport to meet the new constraint which is deducted from the fuel purchased at Nashville airport (7400-440=6960).

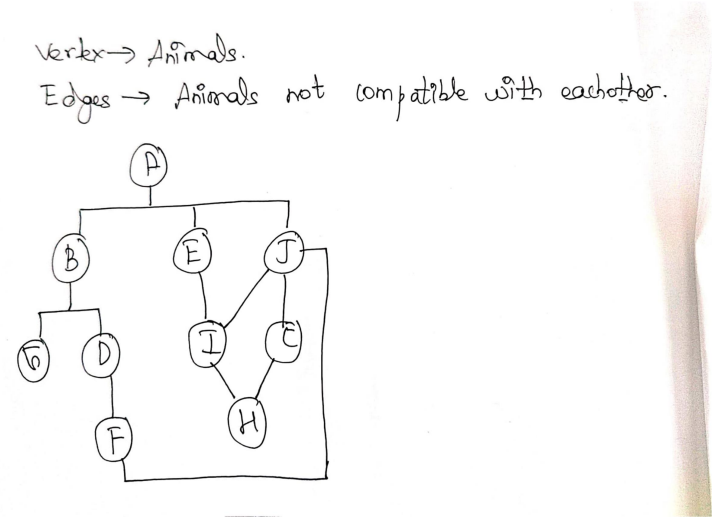
= > $17251.5 + $(8.32-6.48)*(440/6.7) = $17251.5 + $120.83 = **$17372.3**

CONCLUSION:

Thus there is a slight change in the optimal cost due to the new added constraint.

Question 2: OKC Zoo

Let us consider animals to be the vertices(nodes) of the model where the animals which are not compatible with each other are connected by edges(branches). Thus our objective is to make sure that each vertex connected to other through the above edges shouldnot have same color and also the colouring of vertices should be done using minimum number of colors. To build this model the below:



1) Sets :

There is a need to declare sets for nodes and branches for this model

set Vertices; #The Animals a b c d e f g h i j:

The set of nodes is created each representing an animal.

set Edges within (Vertices cross Vertices);

The set of branches is created between the animals which are not compatible with each other that is defined in the .dat file.

2) Parameters :

The parameters for this model are as follows:

TotalColors :

This parameter store the maximum number of possible colors for this model which is 10.

3) Decision Variables :

The decision variables are:

var UseColor{1..TotalColors} binary; :

The binary variable that indiacates whether a color is used to colour vertex.

var AssignColor{Vertices,1..TotalColors} binary; :

The binary variable for assigning colors to vertices

4) Objective Function :

The objective function is to color the vertices with minimum number of colors.

minimize TotalUsedColors: sum{c in 1..TotalColors} UseColor[c];

5) Constraints :

The constraints for the given problem are as follows:

1.Ensuring each vertex is assigned one color:

subject to AssignExactlyOneColor{v in Vertices}:

sum{c in 1..TotalColors} AssignColor[v,c] = 1;

This ensures that each vertex is assigned only one color.

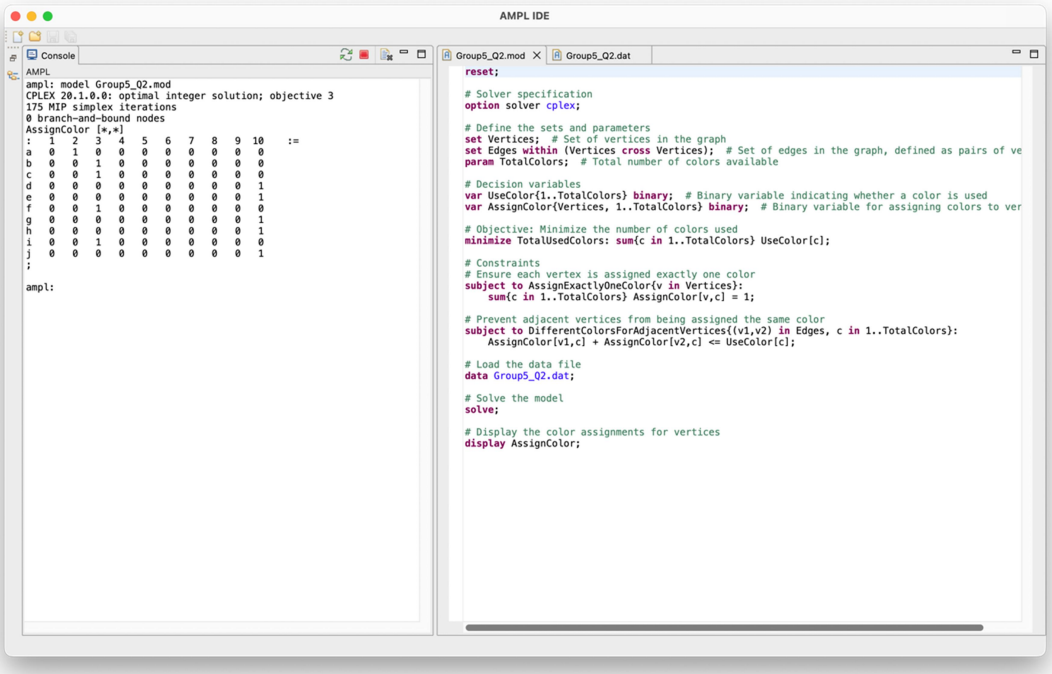
2.Preventing adjacent vertices from being assigned same color:

subject to DifferentColorsForAdjacentVertices{(v1,v2) in Edges, c in 1..TotalColors}:

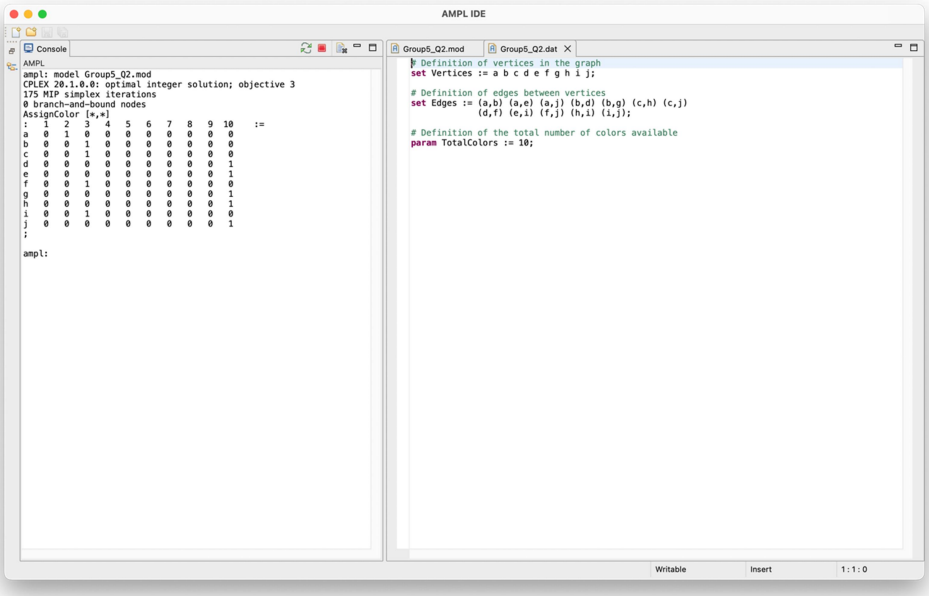
AssignColor[v1,c] + AssignColor[v2,c] <= UseColor[c];

This ensures that adjacent vertices are assigned different colors.

Thus the AMPL CODE is as follows:



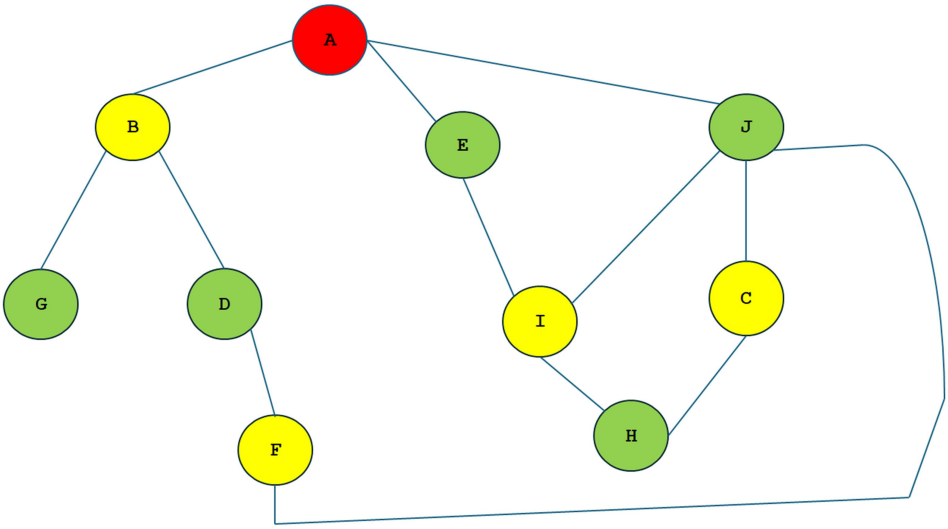
Thus the .DAT file and output is as follows:



CONCLUSION:

From the results, it's determined that three separate enclosures are necessary to accommodate 10 animals, ensuring specific adjacency conditions are met. The allocation is as follows:

- Enclosure 1 houses animal 'a'. (Red 2)
- Enclosure 2 is designated for animals 'b', 'c', 'f', and 'i'. (Yellow 3)
- Enclosure 3 encompasses animals 'd', 'e', 'g', 'h', and 'j'.(Green 10)



Question 3: We Got Gas!

Sets and Parameters:

There are 5 types of gasoline to be stored in 8 tanks, numbered from 1 to 8. The set gas consist of types A,B,C,D,E and set tanknum consists of 1,2,3,4,5,6,7,8.

The below quantities and storage capacities are the parameters.

param g_q{Gas};:

Quantity of Gas Type A = 75000 litres

Quantity of Gas Type B = 50000 litres

Quantity of Gas Type C = 25000 litres

Quantity of Gas Type D = 80000 litres

Quantity of Gas Type E = 20000 litres

param t_c{Tanknum};:

Storage capacity of tank 1 = 25000 litres

Storage capacity of tank 2 = 25000 litres

Storage capacity of tank 3 = 30000 litres

Storage capacity of tank 4 = 60000 litres

Storage capacity of tank 5 = 80000 litres

Storage capacity of tank 6 = 85000 litres

Storage capacity of tank 7 = 100000 litres

Storage capacity of tank 8 = 50000 litres

Each tank has got some pumping cost per 1000 litres for storing the gases, represented by ***param t_cost{Gas,Tanknum};.***

Decision Variables:

Let y_{ij} represent the amount of gasoline type i to be stored in tank j, where i represents the gas type and j represents the storage tank number.

Let x_{ij} represent the selection of a tank for particular gas matrix, i.e., if the value of x for gas I and tank number j is 1 then the gas i is stored in tank number j, otherwise it's value is 0.

Objective Function:

The objective of this problem is to minimize the total pumping cost of storing the gasoline in each storage tank.

Mathematically,

Total Pumping Cost = $\sum_{i=1}^{5} \sum_{j=1}^{8} y_{ij}\, c_{ij}$

where c_{ij} is the pumping cost per 1000 litres of gas i stored in tank j.

There are some constraints to be followed,

Constraints:

There are some constraints to be followed,

i) Each storage tank should not exceed its capacity

$\sum_{i=1}^{5} y_{ij}$ ≤ Capacity for individual tanks from 1 to 8

ii) All types of gas must be stored

$\sum_{j=1}^{8} y_{ij}$ = Gasoline of types A,B,C,D,E

iii) Individual gas must be stored in at least one tank

$\sum_{j=1}^{8} x_j \geq 1$

iv) The tank should have only one type gas stored

$\sum_{i=1}^{5} x_i \leq 1$

v) Gas should be in atleast one tank and restricted to one gas per tank

y_{ij} <= g_q[i]*x[i,j];

AMPL code:

```
reset;
option solver cplex;
#Sets
set Gas;
set Tanknum;
#Parameters
param g_q{Gas};
param t_c{Tanknum};
param t_cost{Gas,Tanknum};
#binary variable to show whether the gas a is stored in tank b
var x{Gas,Tanknum} binary;
var y{Gas,Tanknum} >= 0;
#Objective function
#The objective is to minimize the total pumping cost of gas storage in tanks
minimize Totalcost: sum{a in Gas,b in Tanknum} y[a,b]*t_cost[a,b]/1000;
#Constraints
#capacity of each storage tank should not exceed
subject to T_capacity{b in Tanknum}: sum{a in Gas} y[a,b] <= t_c[b];
#supply of each gas should be met
subject to G_supply{a in Gas}: sum{b in Tanknum} y[a,b] = g_q[a];
#each gas type should be stored in at least one tank
subject to min_gas_store{a in Gas}: sum{b in Tanknum} x[a,b] >= 1;
#only one gas should be in a tank
subject to 1_g_t{b in Tanknum}: sum{a in Gas} x[a,b] <= 1;
# amount if gas should be in atleast one tank and restricted to only one gas per tank
subject to Gas_T{a in Gas, b in Tanknum}: y[a,b] <= g_q[a]*x[a,b];
data group5_HW5_q3.dat;
solve;
printf "Minimum Pumping Cost: %g\n", Totalcost;
display y;
```

AMPL .DAT file:

```
data;

set Gas := A B C D E; #Gas Type
set Tanknum := 1 2 3 4 5 6 7 8; #Tank number

param g_q := A 75000      #Quantity of Gas in litres
             B 50000
             C 25000
             D 80000
             E 20000;

param t_c := 1 25000      #Capacity of tank storage in litres
             2 25000
             3 30000
             4 60000
             5 80000
             6 85000
             7 100000
             8 50000;

param t_cost : 1 2 3 4 5 6 7 8 :=  #Cost of tank storage
            A  1 2 2 1 4 4 5 3
            B  2 3 3 3 1 4 5 2
            C  3 4 1 2 1 4 5 1
            D  1 1 2 2 3 4 5 2
            E  1 1 1 1 1 1 5 5;
```

Output:

```
CPLEX 20.1.0.0: optimal integer solution; objective 320
95 MIP simplex iterations
0 branch-and-bound nodes
Minimum Pumping Cost: 320
y [*,*] (tr)
:     A       B       C       D       E      :=
1   25000       0       0       0       0
2       0       0       0   25000       0
3       0       0   25000       0       0
4   50000       0       0       0       0
5       0   50000       0       0       0
6       0       0       0       0   20000
7       0       0       0    5000       0
8       0       0       0   50000       0
;
```

After performing the optimization, the minimum cost to be spent on storing the gases in different tanks is determined to be **$320.**

The minmum cost storage plan is as follows:

GAS TYPE	TANK FOR GAS TO BE STORED			Total
A	Tank 1 = 25000	Tank 4 = 50000		75000
B	Tank 5 = 50000			50000
C	Tank 3 = 25000			25000
D	Tank 2 =25000	Tank 7 = 5000	Tank 8 = 50000	80000
E	Tank 6 = 20000			20000

Question 4: Galaxy Industries Revisited

Reformulating the Galaxy Industries problem using piecewise linear cost function:

Given the space rays generate 8$ revenue per unit.
Here we are initializing the first set of space rays as sr1 which are from 0 to 125 units,
Costs $1.50
Here we are initializing the second set of space rays as sr2 which are from 125to 225 units,
Costs $1.05.
Here we are initializing the third set of space rays as sr3 which are from 225 to 375 units,
Costs $0.95.
Here we are initializing the first set of space rays as sr4which are from 375 and above units,
Costs $0.75.

Here we are initializing the first set of zappers as zp1 which are from 0 to 50 units
Costs $1.05.
Here we are initializing the first set of space rays as zp2 which are from 50 to 125 units
Costs $0.75 .
Here we are initializing the first set of space rays as zp3 which are from 125 units and above and costs $1.50.

Here we are denoting the sum of all space rays and zappers with the variables x1 and x2 respectively. And initializing the M, where M is the upper limit and assigning it to the largest integer.

For the piecewise linear function, we have to initialize the binary variables for the different set of space rays and zappers. The variables srb1, srb2,srb3 are the binary variables for space rays. The variables zpb1 and zpb2 are the binary variables for zappers.

AMPL Code:

Group5_Q2.mod | group5_HW5_q4.mod

```
reset;
option solver cplex;

# decision variables

var sr1 >= 0 integer;   # Total costs for 1st set of space rays: $1.50 for 0 to 125 units
var sr2 >= 0 integer;   # Total costs for 2nd set of space rays: $1.05 for the next 100 units
var sr3 >= 0 integer;   # Total costs for 3rd set of space rays: $0.95 for the next 150 units
var sr4 >= 0 integer;   # Total costs for last set of space rays: $0.75 for any more

var zp1 >= 0 integer;   # Total costs for 1st set of zappers: $1.05 for 0 to 50 units
var zp2 >= 0 integer;   # Total costs for 2nd set of zappers: $0.75 for the next 75 units
var zp3 >= 0 integer;   # Total costs for 3rd set of zappers: $1.50 for any more units

var x1 = sr1+sr2+sr3+sr4;   # sum of all of space rays
var x2 = zp1+zp2+zp3;       # sum of all of zappers

param M := 100000;  # The value M is set as the upper limit
#for the piecewise linear function we are considering the binary variables  are used
#to indicate which set of costs applies for the production

var srb1 binary; #variable srb1 is a space rays binary variable
var srb2 binary; #variable srb2 is a space rays binary variable
var srb3 binary; #variable srb3 is a space rays binary variable
var zpb1 binary; #variable zpb1 is a zappers binary variable
var zpb2 binary; #variable zpb2 is a zappers binary variable

# optimal solution is maximizing the profit
maximize Solution: 8*x1+5*x2-(1.50*sr1+1.05*sr2+0.95*sr3+0.75*sr4+1.05*zp1+0.75*zp2+1.50*zp3);

# constraints
subject to plastic: 2*x1+x2<=1000;      # special plastic
subject to prodTime: 3*x1+4*x2<=2400;   # production time per week
subject to totalProd: x1+x2<=700;       # Total production cannot exceed 700 units
subject to mix: x1-x2<=350;             # Number of units of Space Rays cannot exceed Zappers by more than 350

# piecewise linear cost function and integer solution for space rays
subject to sr1c1: 125*srb1 <= sr1;
subject to sr1c2: sr1 <= 125;
subject to sr2c1: 100*srb2 <= sr2;
subject to sr2c2: sr2 <= 100*srb1;
subject to sr3c1: 150*srb3 <= sr3;
```

```
# piecewise linear cost function and integer solution for space rays
subject to sr1c1: 125*srb1 <= sr1;
subject to sr1c2: sr1 <= 125;
subject to sr2c1: 100*srb2 <= sr2;
subject to sr2c2: sr2 <= 100*srb1;
subject to sr3c1: 150*srb3 <= sr3;
subject to sr3c2: sr3 <= 150*srb2;
subject to sr4c:  sr4 <= M*srb3;

# piecewise linear cost function and integer solution for zappers
subject to zp1c1: 50*zpb1 <= zp1;
subject to zp1c2: zp1 <= 50;
subject to zp2c1: 75*zpb1 <= zp2;#zp1 b2
subject to zp2c2: zp2 <= 75;#zp1
subject to zp3c:  zp3 <= M*zpb2;#zp2

solve;
display Solution;
display x1;
display x2;
display sr1, sr2, sr3, sr4;
display zp1, zp2, zp3;
display x1+x2;
```

Writable | Insert | 21 : 1 : 856

The objective function is to maximize the profit. With the given equation we can maximize the profit.

```
maximize Solution: 8*x1+5*x2-(1.50*sr1+1.05*sr2+0.95*sr3+0.75*sr4+1.05*zp1+0.75*zp2+1.50*zp3);
```

which has 8$ for space rays and 5$ for zappers and subtracting this from the all the set of space rays and zappers and their corresponding costs.

And the constraints are defined as given in the question.

```
# constraints
subject to plastic: 2*x1+x2<=1000;      # special plastic
subject to prodTime: 3*x1+4*x2<=2400;   # production time per week
subject to totalProd: x1+x2<=700;       # Total production cannot exceed 700 units
subject to mix: x1-x2<=350;             # Number of units of Space Rays cannot exceed Zappers by more than 350
```

The piecewise linear cost function and integer solution for the space rays are

```
# piecewise linear cost function and integer solution for space rays
subject to sr1c1: 125*srb1 <= sr1;
subject to sr1c2: sr1 <= 125;
subject to sr2c1: 100*srb2 <= sr2;
subject to sr2c2: sr2 <= 100*srb1;
subject to sr3c1: 150*srb3 <= sr3;
subject to sr3c2: sr3 <= 150*srb2;
subject to sr4c:  sr4 <= M*srb3;
```

These constraints define the piecewise linear cost functions and integer solutions for both Space Rays and Zappers:

Piecewise Linear Cost Function and Integer Solution for Space Rays:

Constraints for Space Rays:

1. ***sr1c1: (125 * srb1 <=sr1):*** This constraint ensures that the total cost (sr1) of producing Space Rays does not exceed (125) units within the first cost range.
2. ***sr1c2: (sr1 <=125):*** This constraint sets an upper bound on the total cost (sr1) of producing Space Rays within the first cost range.
3. ***sr2c1:(100 * srb2 <= sr2):*** This constraint sets an upper bound on the total cost (sr2) of producing Space Rays within the second cost range.
4. ***sr2c2: (sr2 <=100 * srb1):*** This constraint ensures that the total cost (sr2) of producing Space Rays does not exceed (100) units within the second cost range.
5. ***sr3c1: (150 * srb3 <= sr3):*** This constraint sets an upper bound on the total cost (sr3) of producing Space Rays within the third cost range.
6. ***sr3c2: (sr3 <=150 * srb2):*** This constraint ensures that the total cost (sr3) of producing Space Rays does not exceed (150) units within the third cost range.
7. ***sr4c:(sr4 <= M *srb3):*** This constraint sets an upper bound on the total cost (sr4) of producing Space Rays beyond the third cost range.

Piecewise Linear Cost Function and Integer Solution for Zappers:

Constraints for Zappers:

1. zp1c1: (50 *zpb1 <= zp1): This constraint ensures that the total cost (zp1) of producing Zappers does not exceed (50) units within the first cost range.

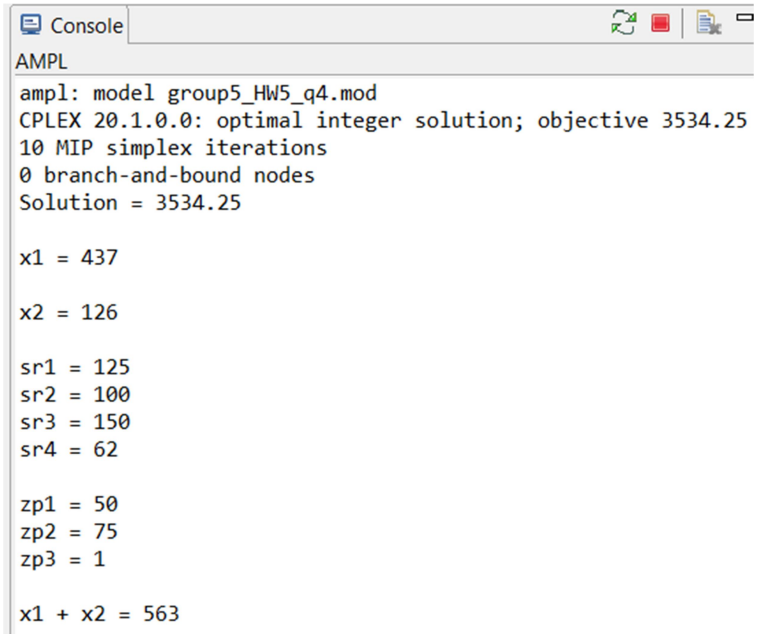
2. zp1c2: (zp1 <= 50): This constraint sets an upper bound on the total cost (zp1) of producing Zappers within the first cost range.

3. zp2c1: (75 * zpb1 <= zp2): This constraint sets an upper bound on the total cost (zp2) of producing Zappers within the second cost range.

4. zp2c2: (zp2 <= 75): This constraint ensures that the total cost (zp2) of producing Zappers does not exceed (75) units within the second cost range.

5. zp3c: (zp3 <=M * zpb2): This constraint sets an upper bound on the total cost (zp3) of producing Zappers beyond the second cost range.

The output of the solution is given below.



CONCLUSION:

The total number of units are 563 out of which 437 are space rays and 126 of them are zappers.

From **437 units of space rays** ,126 units are produced with a cost of $1.50 each and 100 units are produced with a cost of $1.05 each and 150 are produced with a cost of $0.95 and 62 are produced with a cost of $0.75.

From **126 units of zappers** ,50 units are produced at a cost of $1.05 and 75 units are produced at a cost of $0.75 and 1 unit is produced at a cost of $1.50.

Hence the total number of space rays and zappers produced is equal to the 563.

The optimal integral solution for the objective is **$3534.25.**

Question 5: Binary Programming

Decision Variables:
Let x1,x2,x3 and x4 be the decision variables based on the problem given.
In the problem, decision variables are constrained to be in the range 0 and 1 included.

Objective Function:
The objective function for the problem is as follows:
maximize ObjectiveFunction: 90*x1 + 55*x2 + 63*x3 + 47*x4;
Here maximizing the linear combination of decision variables is the objective function.

Constraints:
1. Limit on the combination of resources used by x1, x2, x3, and x4:
subject to ResourceConstraint: 7*x1 + 2*x2 + 8*x3 + 3*x4 <= 10;

2. Either x3 or x4 is 1 constraint:
subject to PairwiseLimit: x3 + x4 <= 1;

3. Ensures x3 is 1 if and only if x1 is 1:
subject to PositivityConstraint1: -x1 + x3 <= 0;

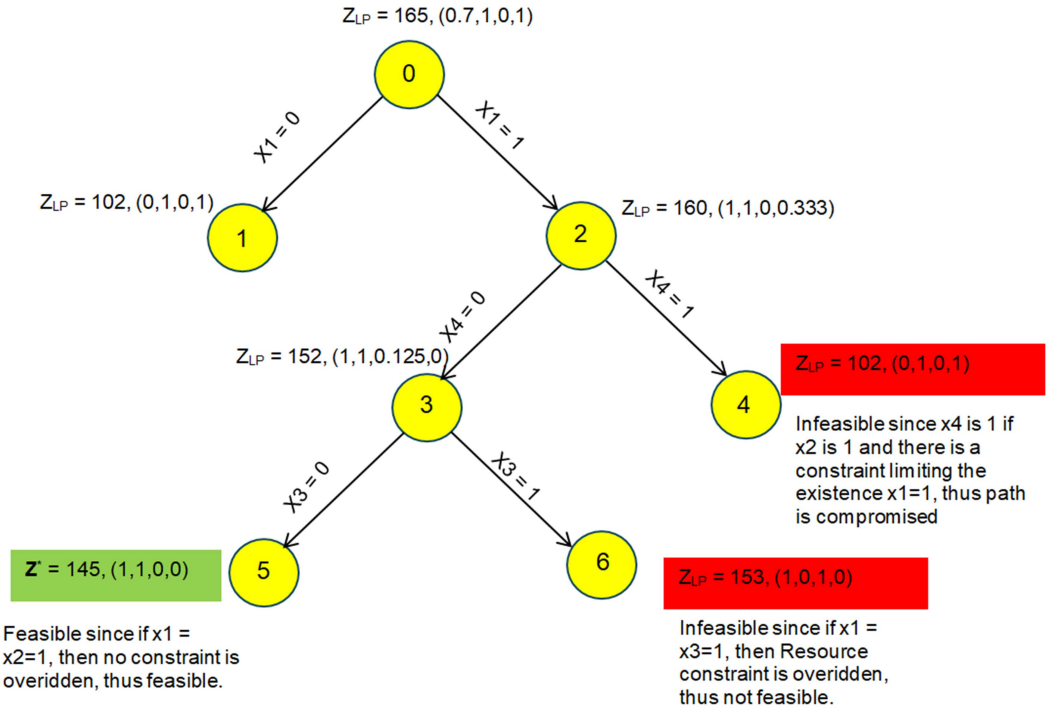
4. Ensures x4 is 1 if and only if x2 is 1:
subject to PositivityConstraint2: -x2 + x4 <= 0;

AMPL Code:

group5_HW5_q5.mod | group5_HW5_q1ai.mod | group5_HW5_q5sam.mod

```
#AMPL HW_5 _5 DSA/ISE5113
#Authors  Sai Abhinov Chowdary Katragadda, Yashasvi Mususku,
#         Vivek Satya Sai Veera Venkata Talluri, Vignesh Murugan
#Date: 03/16/2024
# Problem Description:
# This AMPL script models the Branching and Bounding problem and computes for optimal solution
# Reset AMPL environment and specify solver options:

reset;

# Setting the solver to CPLEX
option solver cplex;

# Definition of decision variables with their bounds
var x1 >=0,<=1;  # Decision variable x1
var x2 >=0,<=1;  # Decision variable x2
var x3 >=0,<=1;  # Decision variable x3
var x4 >=0,<=1;  # Decision variable x4


# Objective function to maximize
maximize ObjectiveFunction: 90*x1 + 55*x2 + 63*x3 + 47*x4;
# Maximizing the linear combination of decision variables

# Constraints
subject to ResourceConstraint: 7*x1 + 2*x2 + 8*x3 + 3*x4 <= 10;
# Limit on the combination of resources used by x1, x2, x3, and x4
subject to PairwiseLimit: x3 + x4 <= 1;
# Constraint that the sum of x3 and x4 must not exceed 1

subject to PositivityConstraint1: -x1 + x3 <= 0;  # Ensures x1 is less than or equal to x3
subject to PositivityConstraint2: -x2 + x4 <= 0;  # Ensures x2 is less than or equal to x4

subject to x1const: x1=1;
subject to x4const: x4=1;
#subject to x3const: x3=1;
# Solve the optimization model
solve;

# Display results
display ObjectiveFunction;  # Display the value of the objective function
display x1, x2, x3, x4;  # Display the values of decision variables
```

The Branching and bounding is as follows:



Node 0:

The result 166.28 is obtained with x1 = 0.7 , x2 = 1 , x3 = 0 , x4 =1 , as x1 does not meet the initial requirement for the problem we create 2 branches , One with x1=0 and x1=1 .

```
ampl: model group5_HW5_q5.mod
CPLEX 20.1.0.0: optimal solution; objective 166.2857143
1 dual simplex iterations (0 in phase I)
ObjectiveFunction = 166.286

x1 = 0.714286
x2 = 1
x3 = 0
x4 = 1
```

Node 1: (Enforced x1=0)

The result 102 is obtained with x1 = 0 , x2 = 1 , x3 = 0 , x4 =1 ,no more branching from this node as we obtained integer solution .

```
ampl: model group5_HW5_q5sam.mod
CPLEX 20.1.0.0: optimal solution; objective 102
0 dual simplex iterations (0 in phase I)
ObjectiveFunction = 102

x1 = 0
x2 = 1
x3 = 0
x4 = 1
```

Node 2: (Enforced x1 = 1)

The result 160.67 is obtained with x1 = 1, x2 = 1 , x3 = 0 , x4 =0.33 , as x4 does not meet the initial requirement for the problem we create 2 branches , One with x4=0 and x4=1 .

```
ampl: model group5_HW5_q5sam.mod
CPLEX 20.1.0.0: optimal solution; objective 160.6666667
1 dual simplex iterations (0 in phase I)
ObjectiveFunction = 160.667

x1 = 1
x2 = 1
x3 = 0
x4 = 0.333333
```

Node 3: (Enforced x1=1 and x4=0)

The result 152.875 is obtained with x1 = 1 , x2 = 1 , x3 = 0.125 , x4 =0 , as x3 does not meet the initial requirement for the problem we create 2 branches , One with x3=0 and x3=1 .

```
ampl: model group5_HW5_q5sam.mod
CPLEX 20.1.0.0: optimal solution; objective 152.875
0 dual simplex iterations (0 in phase I)
ObjectiveFunction = 152.875

x1 = 1
x2 = 1
x3 = 0.125
x4 = 0
```

Node 4: (Enforced x1=1 and x4 = 1)

The result 102 is obtained with x1 = 0 , x2 = 1 , x3 = 0 , x4 =1 ,no more branching from this node as we obtained integer solution, with over rides over enforced constraints as shown below. Thus this solution is in Infeasible region.

```
ampl: model group5_HW5_q5sam.mod
presolve, variable x1:
        impossible deduced bounds: lower = 1, upper = 0.714286;
        difference = 0.285714
ObjectiveFunction = 102

x1 = 0
x2 = 1
x3 = 0
x4 = 1
```

Node 5: (Enforced x1 = 1 and x4 = 0 and x3 = 0)

The result 145 is obtained with x1 = 1 , x2 = 1 , x3 = 0 , x4 =0 ,no more branching from this node as we obtained integer solution .

```
ampl: model group5_HW5_q5sam.mod
CPLEX 20.1.0.0: optimal solution; objective 145
0 dual simplex iterations (0 in phase I)
ObjectiveFunction = 145

x1 = 1
x2 = 1
x3 = 0
x4 = 0
```

Node 6: (Enforced x1 =1 and x4 = 0 and x3 = 1)

The result 153 is obtained with x1 = 1 , x2 = 0 , x3 = 1 , x4 =0 ,no more branching from this node as we obtained integer solution, with over rides enforced constraints as shown below. Thus this solution is in Infeasible region.

```
ampl: model group5_HW5_q5sam.mod
presolve, variable x2:
        impossible deduced bounds: lower = 0, upper = -2.5
ObjectiveFunction = 153

x1 = 1
x2 = 0
x3 = 1
x4 = 0
```

Result:

Thus Output at Node 5 gives the optimal solution, which is 145
for x1 = x2 = 1 and x3 = x4 = 0.

Conclusion:

This can be verified by running the below AMPL code using binary function on all decision variables:

AMPL Code:

group5_HW5_q5.mod | group5_HW5_q1ai.mod

```
#AMPL HW_5 _5 DSA/ISE5113
#Authors  Sai Abhinov Chowdary Katragadda, Yashasvi Mususku,
#         Vivek Satya Sai Veera Venkata Talluri, Vignesh Murugan
#Date: 03/16/2024
# Problem Description:
# This AMPL script models the Branching and Bounding problem and computes for optimal solution
# Reset AMPL environment and specify solver options:

reset;

# Setting the solver to CPLEX
option solver cplex;

# Definition of decision variables with their bounds
var x1 binary;  # Decision variable x1
var x2 binary;  # Decision variable x2
var x3 binary;  # Decision variable x3
var x4 binary;  # Decision variable x4


# Objective function to maximize
maximize ObjectiveFunction: 90*x1 + 55*x2 + 63*x3 + 47*x4;
# Maximizing the linear combination of decision variables

# Constraints
subject to ResourceConstraint: 7*x1 + 2*x2 + 8*x3 + 3*x4 <= 10;
# Limit on the combination of resources used by x1, x2, x3, and x4
subject to PairwiseLimit: x3 + x4 <= 1;
# Constraint that the sum of x3 and x4 must not exceed 1

subject to PositivityConstraint1: -x1 + x3 <= 0;  # Ensures x1 is less than or equal to x3
subject to PositivityConstraint2: -x2 + x4 <= 0;  # Ensures x2 is less than or equal to x4


# Solve the optimization model
solve;

# Display results
display ObjectiveFunction;  # Display the value of the objective function
display x1, x2, x3, x4;  # Display the values of decision variables
```

OUTPUT:

```
ampl: model group5_HW5_q5.mod
CPLEX 20.1.0.0: optimal integer solution; objective 145
0 MIP simplex iterations
0 branch-and-bound nodes
ObjectiveFunction = 145

x1 = 1
x2 = 1
x3 = 0
x4 = 0
```

The computation yields a result of 145, with the specific variable assignments of X1=1, X2=1, X3=0, and X4=0.