DM871 – Linear and Integer Programming

Answers to the Take-home Assignment, Spring 2021

ahoth19

Insert a \newpage before every Subtask.

Template for figure inclusion (see latex source):

Template for source code inclusion:

```
import numpy as np
```

Task 1

Subtask 1.a

We pick the row of the simplex where $x_0 = 1$, which is the third row. In terms of the Chvatal Gomory cut we have

$$\sum_{j \in N} (\bar{a}_{uj} - \lfloor \bar{a}_{uj} \rfloor) x_j \geq \bar{b}_u - \lfloor \bar{b}_u \rfloor,$$

where $N = \{2, 4\}$, we have

$$\bar{b}_u = 10/9$$

$$\bar{a}_{u2} = 1/3 \quad \bar{a}_{u4} = 2/9.$$

By substituting the numbers we get the cut

$$1/3x_2 + 2/9x_4 \geq 1/9.$$

We can write it in terms of the original variables x_0 and x_1. x_2 and x_4 can be written from our original variables, from isolating the slack variables in our constraints

$$x_2 = 2 - 2x_1 - x_0$$

$$x_4 = 2 + 3x_1 - 3x_0$$

We insert these values for x_2 and x_4 into our cut

$$1/3(2 - 2x_1 - x_0) + 2/9(2 + 3x_1 - 3x_0) \geq 1/9$$

$$x_0 \leq 1$$

We have found our Chvatal Gomory cutting plane and written it in terms of the original variables

Subtask 1.b

The optimal solution of the linear relaxation that was previously found has value $z = 32/9$, where $x_0 = 10/9$ and $x_1 = 4/9$. With our new cut we have the constraint

$$x_0 \leq 1$$

This is no longer satisfied since the following inequality is is not satisfied.

$$10/9 \leq 1$$

Hence our previous optimal solution is no longer feasible with the new cut.

Task 2

At each node we relax the problem using linear relaxation and then solve it using zweigmedia.com simplex solver.[1] The initial explanation is explained in node A.

Node A:

We find a relaxed solution to the root by relaxing the problem using linear relaxation. The linear relaxation for binary variables is made by restricting the variables to a value between 0 and 1 (both inclusive).

$$\begin{aligned} \max \quad & z = 5x_1 + 5x_2 + 8x_3 - 2x_4 - 4x_5 \\ \text{s.t.} \quad & -3x_1 + 6x_2 - 7x_3 + 9x_4 + 9x_5 \geq 10 \\ & x_1 + 2x_2 - x_4 - 3x_5 \leq 0 \\ & x_i \geq 0, \quad i = 1, \dots, 5 \\ & x_i \leq 1, \quad i = 1, \dots, 5 \end{aligned}$$

Solving this linear problem gives us the objective function and the solution of the LP, which is a dual bound of our ILP.

$$db = 121/9 \qquad x = [1, 1, 1, 5/6, 13/18]$$

Using the most fractional variable rule for branching, the most fractional variable is x_5.

$$\arg\max_{j \in C} \min\{f_j, 1 - f_j\} = \arg\max_{j \in C}\{0, 0, 0, 0.166, 0.722\} = 5$$

We branch with x_5, where $x_5 = 0$ or $x_5 = 1$. We first expand the greater-or-equal branch (or in our case, =1), meaning we go to node C

Node C:

When solving the linear relaxation of node C, we can set $x_5 = 1$. This gives us the dual bound of C.

$$db = 116/9 \qquad x = [1, 1, 1, 5/9, 1]$$

Again we use the most fractional rule for branching, and now branch on x_4. We branch with $x_4 = 0$ or $x_4 = 1$. We first expand with greater or equal branch, so we go to node G.

Node G: - incumbent solution

Solving the linear relaxation of node G, where $x_5 = 1$ and $x_4 = 1$, we have the following dual bound

$$db = z = 12 \qquad x = [1, 1, 1, 1, 1]$$

This is a feasible solution to our ILP since all variables are binary, this solution is the current incumbent. We now go to the sibling node of G, node F.

Node F:

Solving the linear relaxation of node F, where $x_5 = 1$ and $x_4 = 0$, we have the following solution

$$db = 58/7 \qquad x = [1, 1, 2/7, 0, 1]$$

Since this provides us with a worse dual bound than our current incumbent solution we can prune its sub-problems. We return from our depth-first search to node B

[1]://www.zweigmedia.com/simplex/simplex.php

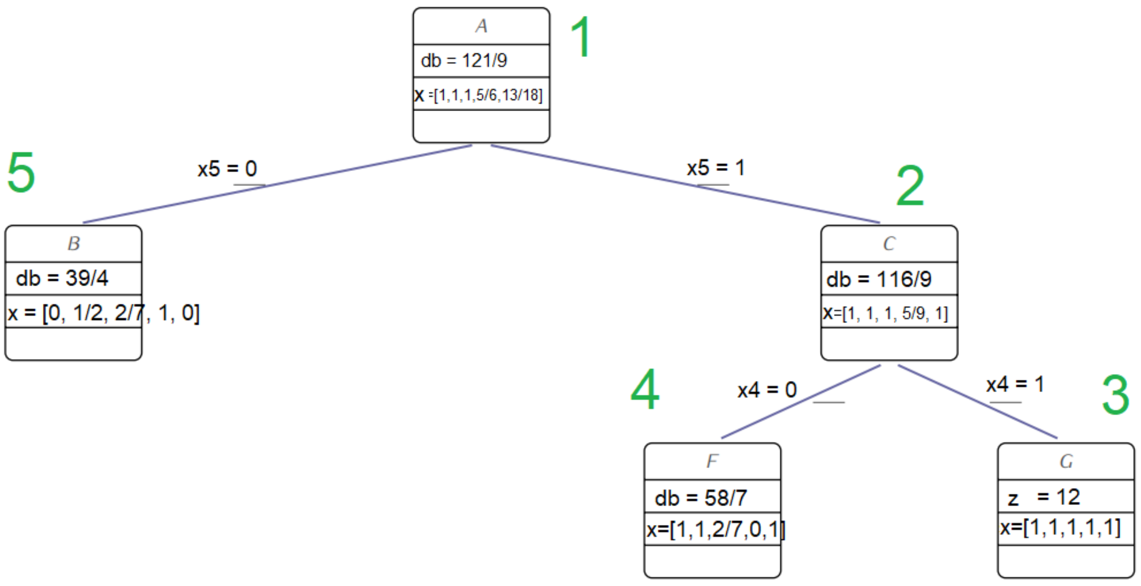
Node B:

Solving the linear relaxation of node B, where $x_5 = 0$, we have the following solution

$$db = 39/14 \qquad x = [0, 1/2, 2/7, 1, 0]$$

Since this provides us with a worse dual bound than our current incumbent solution we can prune its sub-problems.

Since there are no more nodes to search, our incumbent solution is the optimal solution. The solution is presented below in the branch and bound tree.



The optimal solution to the ILP is then $x = [1, 1, 1, 1, 1]$ with value $z = 12$ as found in node G.

Task 3

Subtask 3.a

We introduce an integer variable x_{hij} that describes the amount of beer of type $h \in H$, shipped from city $j \in J$ to city $i \in I$. Where,

$$H = \{'\text{Lager}','\text{Ale}'\}$$

$$J = \{'\text{Odense}','\text{Aarhus}'\}$$

$$I = \{'\text{Copenhagen}','\text{Roskilde}','\text{Aalborg}'\}.$$

We also have the constants $inflow_{hi}$, $cost_{ij}$, $capacity_{ij}$. We model this problem in mathematical programming terms.

$$\begin{aligned}
\min \quad & c = \sum_{i \in I} \sum_{j \in J} (cost_{ij} \cdot \sum_{h \in H} x_{hij}) \\
\text{s.t.} \quad & \sum_{h \in H} x_{hij} \leq capacity_{ij} \quad \forall i \in I, j \in J \\
& \sum_{j \in J} x_{hij} = inflow_{hi} \qquad \forall i \in I, h \in H \\
& x_{hij} \in \mathbb{Z}^+ \qquad \forall h \in H, i \in I, j \in J
\end{aligned}$$

The objective function is to minimize the cost of the beers shipped, depending on the costs of shipping from the cities. This is constrained by two groups of constraints.
The first group of constraint is that the sum amount of beer of the two types shipped from any two cities should be less than or equal to the capacity between those two cities. Meaning we can not ship more than the capacity allows us.
The second group of constraints is that the amount of beer of both types sent to a particular city must satisfy the inflow required of that type of beer h for the given city i.

This is a integer programming problem since the inflow is measured in "number of indivisible cases". This is further restricted to be the case of an integer problem if we cannot send a fractional value of the same type of beer from the two cities j to one of the cities i. For example sending 1/3 a case of Ale from Aarhus to Copenhagen and 2/3 a case of Ale from Odense to Copenhagen would be a valid decision if the variable was not restricted to integer values.

Subtask 3.b

The code below constructs the model in Gurobi, adding the variables, constraints and then solving it and printing the solutions. The optimal solution to the problem was found, and the optimal cost is $c = 5500$ (5.500e+03), where the shipped beers are

50 lager from Odense to Copenhagen
50 lager from Aarhus to Roskilde
10 lager from Aarhus to Aalborg

30 ale from Odense to Copenhagen
30 ale from Odense to Roskilde
10 ale from Aarhus to Copenhagen
30 ale from Aarhus to Aalborg

There is added another group of constraints in subtask 3.b. Different from the two in subtask 3.a. Here the output from the two cities; Odense and Aarhus per beer cannot exceed a specific amount. Odense has to ship 50 lager and 60 ale. And Aarhus has to ship 60 lager and 40 ale.

```
import gurobipy as gp
from gurobipy import GRB
commodities = ["Lager", "Ale"]
nodes = ["Odense", "Aarhus", "Copenhagen", "Roskilde", "Aalborg"]
arcs, capacity = gp.multidict({
    ("Odense", "Copenhagen"): 100,
    ("Odense", "Roskilde"): 80,
    ("Odense", "Aalborg"): 120,
    ("Aarhus", "Copenhagen"): 120,
    ("Aarhus", "Roskilde"): 120,
    ("Aarhus", "Aalborg"): 120
})

cost = {
    ("Lager", "Odense", "Copenhagen"): 10,
    ("Lager", "Odense", "Roskilde"): 20,
    ("Lager", "Odense", "Aalborg"): 60,
    ("Lager", "Aarhus", "Copenhagen"): 40,
    ("Lager", "Aarhus", "Roskilde"): 40,
    ("Lager", "Aarhus", "Aalborg"): 30,
    ("Ale", "Odense", "Copenhagen"): 20,
    ("Ale", "Odense", "Roskilde"): 20,
    ("Ale", "Odense", "Aalborg"): 80,
    ("Ale", "Aarhus", "Copenhagen"): 60,
    ("Ale", "Aarhus", "Roskilde"): 70,
    ("Ale", "Aarhus", "Aalborg"): 30
}

inflow = {
    ("Lager", "Odense"): 50,
    ("Lager", "Aarhus"): 60,
    ("Lager", "Copenhagen"): -50,
    ("Lager", "Roskilde"): -50,
    ("Lager", "Aalborg"): -10,
    ("Ale", "Odense"): 60,
    ("Ale", "Aarhus"): 40,
    ("Ale", "Copenhagen"): -40,
    ("Ale", "Roskilde"): -30,
    ("Ale", "Aalborg"): -30
}

## Model and misc.
m = gp.Model()
```

```
commodities = ["Lager", "Ale"]
nodes_from = ["Odense", "Aarhus"]
nodes_to = ["Copenhagen", "Roskilde", "Aalborg"]

A = cost.keys()
coefficients = [cost[a] for a in A]

## Vars
x = m.addVars(A, obj=coefficients, lb=0.0, vtype=GRB.CONTINUOUS, name="x")

## Constraints

# Capacity
for i in nodes_to:
    for j in nodes_from:
        m.addConstr(gp.quicksum([x[h, j, i] for h in commodities]), GRB.LESS_EQUAL,
            capacity[j, i])

# Inflow in to-cities
for i in nodes_to:
    for h in commodities:
        m.addConstr(gp.quicksum([x[h, j, i] for j in nodes_from]), GRB.EQUAL, -inflow[h
            , i])

# Outflow in from-cities
for j in nodes_from:
    for h in commodities:
        m.addConstr(gp.quicksum([x[h, j, i] for i in nodes_to]), GRB.EQUAL, inflow[h, j
            ])

# Solve model
m.write("exam3.lp")
m.optimize()
m.display()
sol = m.getAttr("x", x)
for s in sol:
    print (s, sol[s])
```