42114: Integer Programming

Solutions

Week 2

Updated: September 6, 2022

Week 2: Formulations

NASA Capital Budget Model

1. Each of the five-year periods is actually a knapsack capacity constraint. If a mission has any cost in a given five-year period it will be part of the knapsack constraint for that period. E.g. the constraint for the first period:

$$6x_1 + 2x_2 + 3x_3 + x_7 + 4x_9 + 5x_12 \leq 10$$

Since mission 1, 2, 3, 7, 9 and 12 are the only missions with a cost during the first five-year period.
The incompatible missions can be handled using set packing constraints. E.g. we cannot choose both mission 4 and 5:

$$x_4 + x_5 \leq 1$$

Finally the dependencies can be modelled using inequalities as so:

$$x_4 \leq x_3$$

Hence, mission 4 cannot be carried out unless mission 3 is also carried out.
The objective function is a sum of the mission values.
The full model is then:

$$\begin{aligned}
\max \quad & 200x_1 + 3x_2 + 20x_3 + 50x_4 + 70x_5 + 20x_6 + 5x_7 + 10x_8 + \\
& 200x_9 + 150x_{10} + 18x_{11} + 8x_{12} + 300x_{13} + 185x_{14}
\end{aligned}$$

$$\begin{aligned}
\text{s.t.} \quad & 6x_1 + 2x_2 + 3x_3 + x_7 + 4x_9 + 5x_{12} && \leq && 10 \\
& 3x_2 + 5x_3 + 5x_5 + 8x_7 + 5x_9 + 8x_{10} + 7x_{12} + x_{13} + 4x_{14} && \leq && 12 \\
& 8x_5 + x_6 + 4x_{10} + 2x_{11} + 4x_{13} + 5x_{14} && \leq && 14 \\
& 8x_6 + 5x_8 + 7x_{11} + x_{13} + 3x_{14} && \leq && 14 \\
& 10x_4 + 4x_6 + x_{13} + 3x_{14} && \leq && 14 \\
& x_4 + x_5 && \leq && 1 \\
& x_8 + x_9 && \leq && 1 \\
& x_{11} + x_{14} && \leq && 1 \\
& x_{11} && \leq && x_2 \\
& x_4 && \leq && x_3 \\
& x_5 && \leq && x_3 \\
& x_6 && \leq && x_3 \\
& x_7 && \leq && x_3 \\
& x_i && \in && \{0,1\} \quad \forall i = 1...14
\end{aligned}$$

The first five constraints are the knapsack constraints for each five-year period, the following three are the incompatibility constraints and finally the dependency constraints.

2. The model in inputted into Julia (see at the end of the document) and solved. The found optimal solution is: $x_1 = x_9 = x_{13} = x_{14} = 1$ and the rest of the variables are 0. The optimal solution value is 885.

A Container Problem

1. We note that there are two decisions we have to make when solving the Container Problem. Do we want to use a container at the end or not, and if not then where do we pour the content of the container. To reflect this we need two binary decision variables:

$$y_i = \begin{cases} 1 & \text{if we use container } i \text{ in the end} \\ 0 & \text{otherwise} \end{cases}$$

for each container i and

$$x_{ij} = \begin{cases} 1 & \text{if the contents of container } i \text{ is poured into container } j \\ 0 & \text{otherwise} \end{cases}$$

for each pair i, j of containers.
As we want to use as few containers as possible, the objective function will be:

$$\min \sum_{i=1}^{n} y_i$$

We need to ensure that the total volume of a container is never exceeded when pouring more content into a container:

$$\sum_{i=1, i\neq j}^{n} (a_i x_{ij}) + a_j y_j \leq v_j y_j \qquad \forall j = 1...n$$

Furthermore, this ensures that we do not pour any content into a container which is not used at the end.
We also need to ensure that we do not pour content from a container we use in the end **and** that the liquid is not split between multiple containers when we pour from an unused container:

$$y_i + \sum_{j=1, i\neq j}^{n} x_{ij} = 1 \qquad \forall i = 1...n$$

Thus we get the model:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} y_i \\
\text{s.t.} \quad & a_j y_j + \sum_{\substack{i=1 \\ i\neq j}}^{n} a_i x_{ij} \leq v_j y_j && \forall j = 1...n \\
& y_i + \sum_{\substack{j=1 \\ i\neq j}}^{n} x_{ij} = 1 && \forall i = 1...n \\
& x_{ij} \in \{0,1\} && \forall i = 1...n, \forall j = 1...n \\
& y_i \in \{0,1\} && \forall i = 1...n
\end{aligned}$$

As we do not really use x_{ii} for any i, we might use x_{ii} as y_i and discard the y-variables. This is not very "beautiful" but brings us down to only one set of variables (in total n^2). Doing it this

way, the model looks like this:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n} x_{ii} & & & \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_i x_{ij} & \leq & v_j x_{jj} & \forall j = 1...n \\
& \sum_{j=1}^{n} x_{ij} & = & 1 & \forall i = 1...n \\
& x_{ij} & \in & \{0,1\} & \forall i = 1...n, \forall j = 1...n
\end{aligned}$$

where the $x_{ii} = y_i$ parts in the constraints can now be included in the sums.

2. The model is inputted into Julia (see at the end of the document) with the given values.
   By making all the variables continuous, the problem becomes a linear programming problem. The optimal LP solution has an objective value of $2\frac{2}{3}$ using containers 1 and 2 and $\frac{2}{3}$ of container 4. You may run the continuous version of the Julia code to get the full solution. The optimal solution of the integer programming problem has an objective value of 3 using containers 1, 2 and 4. Containers 3,5 and 6 are emptied by taking the contents of container 3 and putting it into container 2, taking the contents of container 5 and putting it into container 4 and taking the contents of container 6 and putting it into container 1.
   Notice: There might be more than one optimal solution where only three containers are used. In this case they are all equally good seen from the Integer Program's point-of-view, as they all minimize the objective function.
   Is it a surprise that the linear programming optimum is smaller than the integer programming optimum?

IP Exam 2015: Question 2 (15%)

Question 2.1

Let y_t^A and y_t^B be the binary variables that indicates whether we produce product A and B on day t or not. Let x_t^A and x_t^B be the inventory of product A and B on day t. Finally, let p_t^A and p_t^B be the quantity produced on day t of product A and B.

Since the machine can only produce one product on any given day, and we assume that the initial inventory of both products are 0 units, the demands of both products cannot be strictly positive on the first day. hence, we make this assumption.

We wish to minimize the cost of the production schedule. Hence we minimize the sum of set-up costs and storage costs:

$$\min \sum_{t=1}^{5} (f_A y_t^A + f_B y_t^B + s_t^A x_t^A + s_t^B x_t^B)$$

Balancing constraints for the inventories or constraints for the conservation of inventory are needed for all days $t = 1...5$ and for both products:

$$p_t^A + x_{t-1}^A = d_t^A + x_t^A$$

$$p_t^B + x_{t-1}^B = d_t^B + x_t^B$$

The initial and final inventories are decided beforehand through assumptions:

$$x_0^A = x_0^B = x_5^A = x_5^B = 0$$

Production limitations on both products need to be enforced for each day $t = 1...5$:

$$p_t^A \leq l_A y_t^A$$

$$p_t^B \leq l_B y_t^B$$

Finally we make sure that only one product is produced on any day $t = 1...5$:

$$y_t^A + y_t^B \leq 1$$

Hence, we have the entire model:

$$\min \quad \sum_{t=1}^{5} (f_A y_t^A + f_B y_t^B + s_t^A x_t^A + s_t^B x_t^B)$$

$$\begin{array}{lllll}
\text{s.t.} & p_t^A + x_{t-1}^A & = & d_t^A + x_t^A & \\
& p_t^B + x_{t-1}^B & = & d_t^B + x_t^B & \\
& p_t^A & \leq & l_A y_t^A & \forall t = 1...5 \\
& p_t^B & \leq & l_B y_t^B & \forall t = 1...5 \\
& y_t^A + y_t^B & \leq & 1 & \forall t = 1...5 \\
& x_0^A, x_0^B, x_5^A, x_5^B & = & 0 & \\
& y_t^A, y_t^B & \in & \{0, 1\} & \forall t = 1...5 \\
& x_t^A, x_t^B, p_t^A, p_t^B & \geq & 0 & \forall t = 1...5
\end{array}$$

Question 2.2

What we need to detect is when y_t^A goes from 0 to 1 (going from producing product B to producing product A) and when y_t^B goes from 0 to 1 (going from producing product B to producing product A). To do this we introduce the binary variable z_t^A. It is 1 if we produce product A for the first time after having produced product B. This is modelled as:

$$z_t^A \geq y_t^A - y_{t-1}^A$$
$$z_t^A \geq 0$$

Likewise we introduce the binary variable z_t^B and model:

$$z_t^B \geq y_t^B - y_{t-1}^B$$
$$z_t^B \geq 0$$

Since the set-up cost on the first day is always paid and we only need to pay setup costs when changing between the products the objective function is rewritten as:

$$\sum_{t=1}^{5} (s_t^A x_t^A + s_t^B x_t^B) + f_A y_1^A + f_B y_1^B + \sum_{t=2}^{5} (f_A z_t^A + f_B z_t^B)$$

Then the new full model is:

$$\min \quad \sum_{t=1}^{5}(s_t^A x_t^A + s_t^B x_t^B) + f_A y_1^A + f_B y_1^B + \sum_{t=2}^{5}(f_A z_t^A + f_B z_t^B)$$

$$\begin{array}{lllll}
\text{s.t.} & p_t^A + x_{t-1}^A & = & d_t^A + x_t^A & \\
& p_t^B + x_{t-1}^B & = & d_t^B + x_t^B & \\
& p_t^A & \leq & l_A y_t^A & \forall t = 1...5 \\
& p_t^B & \leq & l_B y_t^B & \forall t = 1...5 \\
& y_t^A + y_t^B & \leq & 1 & \forall t = 1...5 \\
& z_t^A & \geq & y_t^A - y_{t-1}^A & \forall t = 2...5 \\
& z_t^B & \geq & y_t^B - y_{t-1}^B & \forall t = 2...5 \\
& x_0^A, x_0^B, x_5^A, x_5^B & = & 0 & \\
& y_t^A, y_t^B & \in & \{0,1\} & \forall t = 1...5 \\
& x_t^A, x_t^B, p_t^A, p_t^B, z_t^A, z_t^B & \geq & 0 & \forall t = 1...5
\end{array}$$

Since it is a minimization problem and z_t^A and z_t^B both have a lower bound of either 0 or 1 according to the constraints, it is enough to define it as a positive continuous variable instead of a binary variable.

IP Exam 2013: Question 3 (15%)

Question 3.2

In order to check if $K^C \subseteq K'$ or vice versa, we write up the formulations:

$$\begin{array}{rlll}
K' = \{ & x \in [0,1]^4 & : & 6x_1 + 6x_2 + 6x_3 + 5x_4 \leq 16\} \\
K^C = \{ & x \in [0,1]^4 & : & x_1 + x_2 + x_3 \leq 2, \\
& & & x_1 + x_2 + x_4 \leq 2, \\
& & & x_1 + x_3 + x_4 \leq 2, \\
& & & x_2 + x_3 + x_4 \leq 2\}
\end{array}$$

By adding the constraints in K^C we get:

$$3x_1 + 3x_2 + 3x_3 + 3x_4 \leq 8$$

Multiplying this by 2 we get:

$$6x_1 + 6x_2 + 6x_3 + 6x_4 \leq 16$$

We see that this is very similar to the single constraint in K'.

Since the coefficients are equal or higher (in the case of x_4) in K^C, and thus are "punished" as much or more in the constraints than in K', any feasible solution of K^C must also be feasible in K', hence $K^C \subseteq K'$.

In other words, for any feasible solution $(x_1, x_2, x_3, x_4) \in K^C$ the constraints in K^C are obeyed. But then:

$$6x_1 + 6x_2 + 6x_3 + 5x_4 \leq 6x_1 + 6x_2 + 6x_3 + 6x_4 \leq 16$$

which means the constraint in K' is also obeyed and it is therefore in K'.

Question 3.3

To show that K^C is a better formulation than K', we must show that $K^C \subset K'$. We have already proven that $K^C \subseteq K'$ and only need to show that there is at least one feasible solution in K' which is not in K^C.
There are quite a few of these solutions. Letting $x_4 = 1$ and then having any combination where $x_1 + x_2 + x_3 = 1\frac{5}{6}$ will give:

$$6x_1 + 6x_2 + 6x_3 + 5x_4 = 16 \leq 16$$

in K', but in K^C not all constraints will hold:

$$x_1 + x_2 + x_4 = 2\frac{5}{6} \not\leq 2 \text{ or } x_1 + x_3 + x_4 = 2\frac{5}{6} \not\leq 2 \text{ or } x_2 + x_3 + x_4 = 2\frac{5}{6} \not\leq 2$$

Therefore there is a feasible solution (e.g. $(\frac{5}{6}, 1, 0, 1)$) in K', which is not in K^C, meaning that K^C is a better formulation than K'.

Julia Code

Week 1: Manpower Planning Problem

```julia
## Packages needed for solving MIPs
using JuMP, GLPK

m = Model(GLPK.Optimizer) # Defining model

@variable(m,x[1:7] >= 0, Int) # Defining integer variables

n = length(x)                          # Defining the number of variables
b = [10 5 10 5 10 5 10]                # Used for solving Problem 2
#b = [8 8 8 8 8 8 7]                    # Used for solving Problem 3

# Setting up the objective function
@objective(m, Min, sum( x[i] for i=1:n))

# Defining the constraints
@constraint(m, x[1] + x[4] + x[5] + x[6] + x[7] >= b[1]) # Monday
@constraint(m, x[1] + x[2] + x[5] + x[6] + x[7] >= b[2]) # Tuesday
@constraint(m, x[1] + x[2] + x[3] + x[6] + x[7] >= b[3]) # Wednesday
@constraint(m, x[1] + x[2] + x[3] + x[4] + x[7] >= b[4]) # Thursday
@constraint(m, x[1] + x[2] + x[3] + x[4] + x[5] >= b[5]) # Friday
@constraint(m, x[2] + x[3] + x[4] + x[5] + x[6] >= b[6]) # Saturday
@constraint(m, x[3] + x[4] + x[5] + x[6] + x[7] >= b[7]) # Sunday

JuMP.optimize!(m) # Solving the model

# Printing the objective value
println("Objective: ", JuMP.objective_value(m))

for i in 1:n
    # Printing solution as individual items
    println("x$i = ",JuMP.value(x[i]), " ")
end

println("X = ", JuMP.value.(x)) # Printing solution as a vector

z = objective_value(m) # Saving the objective value
# Computing the number unused man days for Problem 4
println(" Number of unused man day = ", 5*z- sum(b))
```

Week 2: NASA Capital Budget Model

```julia
## Packages needed for solving MIPs
using JuMP, GLPK

m = Model(GLPK.Optimizer) # Setting up the model

# Defining the parameters
profit = [200 3 20 50 70 20 5 10 200 150 18 8 300 185]
weights = [6 0 0 0 0; 2 3 0 0 0; 3 5 0 0 0; 0 0 0 0 10; 0 5 8 0 0;
                   0 0 1 8 4; 1 8 0 0 0; 0 0 0 5 0; 4 5 0 0 0; 0 8 4 0 0;
                   0 0 2 7 0; 5 7 0 0 0; 0 1 4 1 1; 0 4 5 3 3]
budgets = [10 12 14 14 14]

n=14 # Number of missions
b=5 # Number of five-year time periods

# Defining the binary variables
@variable(m, x[1:n], Bin)

# Defining the objective function
@objective(m, Max, sum(profit[i]*x[i] for i=1:n))

# Defining the constraints
@constraint(m, [j=1:b], sum(weights[i,j]*x[i] for i=1:n) <= budgets[j])
@constraint(m, x[4]+x[5]<=1)
@constraint(m, x[8]+x[9]<=1)
@constraint(m, x[11]+x[14]<=1)
@constraint(m, x[11]<=x[2])
@constraint(m, x[4]<=x[3])
@constraint(m, x[5]<=x[3])
@constraint(m, x[6]<=x[3])
@constraint(m, x[7]<=x[3])

JuMP.optimize!(m) # Solving the model

# Printing the solution
println("Objective Value: ", objective_value(m))
println("Variable values: ", value.(x))
```

Week 2: A Container Problem

```julia
## Packages needed for solving MIPs
using JuMP, GLPK

m = Model(GLPK.Optimizer) # Setting up the model

# Defining variables as binary
@variable(m,y[1:6],Bin)
@variable(m,x[1:6,1:6],Bin)

# Solved as a LP relaxation
#@variable(m,y[1:6] >= 0)
#@variable(m,x[1:6,1:6] >= 0)

n = length(y) # Defining the number of variables

v = (500,500,300,300,200,100) # Size of the containers
a = (350,300,200,150,100,100) # Volume of the containers

# Defining the objective function
@objective(m, Min, sum( y[i] for i=1:n))

# Defining the constraints
@constraint(m, [j = 1:n], sum(a[i]*x[i,j]  for i = 1:n if i != j) +
    a[j]*y[j]<= v[j]*y[j])
@constraint(m, [i = 1:n], sum(x[i,j] for j = 1:n if j != i) + y[i]  == 1)

JuMP.optimize!(m) # Solving the model

# Printing the solution
println("Objective value = ", objective_value(m))

# Remember a dot for printing a vector
println("y = ", value.(y))

println("X = ")
for i in 1:n
    for j in 1:n
        print(value(x[i,j]) , " ")
    end
    println()
end

println("Solver status: ", termination_status(m))
println("Solution status: ", primal_status(m))
```

Contents