ISYE 6669 Deterministic Optimization

Homework 10

March 28th, 2021

1.

This is the generic RMP formulation:

$$\min \sum_{j \in I} x_j$$

$$s.t \sum_{j \in I} A_j x_j = b$$

$$x_j \geq 0, \forall j \in I$$

In our case, we're selecting columns 1-3 to form the initial subset of patterns to start from, to create the following formulation:

$$\min x_1 + x_2 + x_3$$

$$s.t$$

$$\begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 \\ 7 \\ 0 \end{bmatrix} x_2 + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} x_3 = \begin{bmatrix} 15 \\ 30 \\ 20 \end{bmatrix}$$

$$x_1, x_2, x_3 \geq 0$$

$$Optimal\ Solution: \ \overline{x_1} = 1.5,\ \overline{x_2} = \frac{30}{7},\ \overline{x_3} = 4$$

$$= 9.79$$

$$Optimal\ Basis:$$

$$B = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 5 \end{bmatrix} c_B = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix},$$

$$\hat{y} = c_B^T B^{-1}$$

$$= [1, 1, 1] \begin{bmatrix} \frac{1}{10} & 0 & 0 \\ 0 & \frac{1}{7} & 0 \\ 0 & 0 & \frac{1}{5} \end{bmatrix}$$

$$= \left[\frac{1}{10}, \frac{1}{7}, \frac{1}{5} \right]$$

2.

3.

Iteration 1

$$\widehat{Z} = max\ \frac{1}{10}a_1 + \frac{1}{7}a_2 + \frac{1}{5}a_3$$
$$s.t$$
$$7a_1 + 11a_2 + 16a_3 \leq 80$$
$$a_1, a_2, a_3 \geq 0$$

In this case our optimal solution is :

$$a_1 = 11, a_2 = 0, a_3 = 0$$
$$\widehat{Z} = 11 * \frac{1}{10}$$
$$= 1.1$$

Therefore, the minimum reduced cost is

$\frac{1}{5} - \widehat{Z} = -\frac{9}{10} < 0,$ *which means the current solution is not optimal and* A_1 *enter the list I*

4-5

```
import cvxpy as cp
import numpy as np


def solve_rmp(A, n):
    """[Solve reduced master problem]

    Args:
        A ([type]): [A matrix containing weights for each roll
type]
        n ([type]): [Number of decision variables for x]
        c ([type]): [Cost vector with reduced costs for each
iteration]
        b ([type]): [Bounds]
    """
    x = cp.Variable((n,1))

    b = np.array([15, 30, 20]).reshape((3,1))
    c = np.ones((3,1))
    # defining objective
    objective = cp.Minimize(cp.sum(x))
```

```
    # defining constraints
    constraints = [A@x==b,
                   x>=0]

    rmp = cp.Problem(objective, constraints)
    rmp.solve()
    # printing outputs
    print("\nThe optimal value for knapsack objective function
is:", round(rmp.value, 2))
    print("The values for variable a's in knapsack are:",
rmp.variables()[0].value)

    return rmp


def calculate_reduced_costs(B):
    """[Calculate reduced costs for direction vector]

    Args:
        y ([np.array]): [Cost vector]
        B ([np.array]): [Basis matrix]

    Returns:
        [type]: [Direction db vector with weights to determine if
we're reached our optimal solution,
        if all weights > 0, we've reached optimal, if any weights
< 0 we select the minimum index to enter
        enter the basis]
    """
    #Cb
    c = np.ones((3,1))
    #Get inverse of basis matrix
    B_inv = np.linalg.inv(B)
    #Get reduced costs for simplex method step
    y = B_inv.dot(c)
    #Return index of minimum value
    return y
```

```
def solve_knapsack_problem(y):
    #Knapsack problem
    a = cp.Variable((3,1), integer=True)
    w = np.array([7, 11, 16]).reshape((3,1))

    knapsack_objective = cp.Maximize(cp.sum(cp.multiply(a,y)))
    # defining constraints
    knap_sack_constraints = [cp.sum(cp.multiply(w,a))<=80,
                a>=0]
    knapsack_prob = cp.Problem(knapsack_objective,
knap_sack_constraints)
    knapsack_prob.solve()

    # printing outputs
    print("\nThe optimal value for knapsack objective function
is:", round(knapsack_prob.value, 2))
    print("The values for variable a's in knapsack are:",
knapsack_prob.variables()[0].value)
    return knapsack_prob


if __name__ == '__main__':
    #RM_Step 1
    A_1 = np.array([[10, 0, 0],
                [0, 7, 0],
                [0, 0, 5]])
    A_2 = np.array([[11,10, 0, 0],
                    [0,0, 7, 0],
                    [0,0, 0, 5]])

    # defining objective
    rmp = solve_rmp(A_1,3)

    #Determine which variables are non-negative from the optimal
solution, to create basis matrix
    basis = A_1[:,np.where(rmp.variables()[0].value >= 1e-5)[0]]
    y_reduced_costs = calculate_reduced_costs(basis)

    #Calculate knapsack problem
```

```
    knapsack_sol = solve_knapsack_problem(y_reduced_costs)

    # defining objective
    rmp_step2 = solve_rmp(A_2,4)

    #Determine which variables are non-negative from the optimal
solution, to create basis matrix
    basis_2 = A_2[:,np.where(rmp_step2.variables()[0].value >= 1e-
5)[0]]
    y_reduced_costs_2 = calculate_reduced_costs(basis_2)

    #Calculate knapsack problem
    knapsack_sol2 = solve_knapsack_problem(y_reduced_costs)
```

Q4 Results:

The optimal value for the RMP objective function is: 9.79

The values for variable x's in RMP are: [[1.5]
[4.28571429]
[4.]]

The reduced costs are: [[0.1]
[0.14285714]
[0.2]]
Long-step dual simplex will be used

The optimal value for knapsack objective function is: 1.1

The values for variable a's in knapsack are: [[11.]
[0.]
[0.]]

The optimal value for the RMP objective function is: 9.65

Since 1 - z ⩽ 0 for all reduced costs, we need to use the knapsack problem to add the generated column.

Meaning in this case the index in A corresponding to the optimal solution [11,0,0], now enter the basis. Meaning our new pattern is what we see in A_2 =

$$\begin{pmatrix} 11 & 10 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}$$

Q5 (results) :

The values for variable x's in RMP are: [[1.36363635e+00]
 [1.72665593e-08]
 [4.28571429e+00]
 [4.00000000e+00]]

The reduced costs are: [[0.09090909]
 [0.14285714]
 [0.2   ]]
Long-step dual simplex will be used

The optimal value for knapsack objective function is: 1.04

The values for variable a's in knapsack are: [[2.]
 [6.]
 [0.]]

Since 1 - z ⩽ 0 for all reduced costs, we need to use the knapsack problem to add the generated column.

Therefore we would continue until the optimal solution is determined.

6. Not enough time.