Gerald Jones

Home Work 7: Gepbab Corporation with sensitivity analysis

ISE522 Spg 22

Problem Description:

> Gepbab Corporation produces three products at two different plants. The cost of producing a unit at each plant is shown on the next slide. Each plant can produce a total of 10,000 units.
>
> At least 6,000 units of product 1, 8,000 units of product 2, and 5,000 units of product 3 must be produced.
>
> Formulate an LP to minimize cost. Solve your LP and use the solution to answer questions on slides 4 and 5.

Questions: See bottom of pdf for after solution discussion section

Module imports and data loading

In [1]:
```
from _GUROBI_TOOLS_.GUROBI_MODEL_BUILDING_TOOLS import *
from _NOTE_BOOK_UTILS import *
```

Data Display

In [2]:
```
# display data for problem
data_df = pd.DataFrame({
    "cost-1": [5, 8],
    "cost-2": [6, 7],
    "cost-3": [8, 10],
}, index = ["plant-1", "plant-2"])

display(data_df)
```

	cost-1	cost-2	cost-3
plant-1	5	6	8
plant-2	8	7	10

Model Formulation

Parameters and Sets:

S Set of Plants or sources of products, $\{1,2\} s \in S$

I Set of Products/Items, $\{1,2,3\}$ i $\in I$

Variables:

$X_{s,i}$ amount from plant s of item i produced

Parameters:

$C_{s,i}$ cost per unit for plant s to produce item i

Constraints:

Production Constraints

$$\sum_{i=1}^{|I|} X_{s,i} \leq 10\text{K}; \forall s \in S$$

Product 1 Constraint

$$\sum_{s=1}^{|S|} X_{s,i} \geq 6\text{K for } i = 1$$

Product 2 Constraint

$$\sum_{s=1}^{|S|} X_{s,i} \geq 8\text{K for } i = 2$$

Product 3 Constraint

$$\sum_{s=1}^{|S|} X_{s,i} \geq 5\text{K for } i = 3$$

Objective:

Minimize:

$$\min(\sum_{s=1}^{|S|} \sum_{i=1}^{|I|} (X_{s,i} \cdot C_{s,i}))$$

Method Definitions

In [3]:
```
# generate objective
def generate_production_cost_expression(Xsi, Csi, S, I, verbose=False):
    expr = 0
    for s in range(S):
        for i in range(I):
            plant = "plant-{}".format(s+1)
            if verbose:
                print("The cost of product {:d} is {} for {}".format(i, Csi.iloc[s, i], plant))

            expr += Xsi[s,i] * Csi.iloc[s, i]
        print()
    return expr

def process_required_list(pl, S, verbose=False):
    if not isinstance(pl, list):
        if verbose:
            print("given non string forming string")
        if isinstance(pl, (int, float)):
            pl = list([pl]*S)
            if verbose:
                print("product limits are now:")
    else:
        if verbose:
            print("product limits are")
    if verbose:
        print(pl)
    return pl

# generate constraints
def production_constraints(model, Xsi, production_limits, S, I, verbose=False, name="capacity constraint plant
    # process given production limits
    # if given a list will just return the list
    # if given some form of number it will be converted to a list with that value
    # repeated S times
    production_limits = process_required_list(production_limits, S, verbose)
    # iterate through each plant
    for s in range(S):
        # for each plant (s) sum the amount of products it produces (X{s,i})
        # and constrain it to the limit
        expr = 0
        for i in range(I):
            expr += Xsi[s, i]
        idx = s%len(production_limits)
        if verbose:
            print("index: ", idx)
            print("Limit for plant {:d}: {:d}".format(s, production_limits[idx]))
            print(name.format(s))
        model.addConstr(expr <= production_limits[idx], name.format(s))
    return

def product_constraints(model, Xsi, product_minimums, S, I, verbose=False, name="product {:d} constraint"):
    # process given production limits
    # if given a list will just return the list
    # if given some form of number it will be converted to a list with that value
    # repeated S times
    product_minimums = process_required_list(product_minimums, I, verbose)
    # iterate through each product
    for i in range(I):
        # for each product (i) sum the amount produced by each plant (s) and
        # constraint it to be above the lower bound
        expr = 0
        for s in range(S):
            expr += Xsi[s, i]
            # make sure the index does not go out of bounds
        idx = i%len(product_minimums)

        if verbose:
            print("index: ", idx)
            print("Limit for product {:d}: {:d}".format(i, product_minimums[idx]))
            print(name.format(s))
        model.addConstr(expr >= product_minimums[idx], name.format(i+1))
    return
```

Gurobi Implementation and Solution

In [4]:
```
try:
    # instantiate model object
    m = gp.Model("Gepbab Corporation")
    S = data_df.shape[0]
    I = data_df.shape[1]
    verbose = False
    ####################################################################################
    ################################# Parameters set up ################################
    ####################################################################################
    product_limits = [6,           # product 1 lower bound on production
                      8,           # product 2 lower bound on production
                      5,           # product 3 lower bound on production
                     ]

    ####################################################################################
    ################################# Variables set up #################################
    ####################################################################################
    Xsi = m.addVars(S, I, vtype=GRB.CONTINUOUS, name="Xsi", lb=0)       # X1, X2 creation and >= 0


#     m.addConstrs(  for s in range(S) for i in range)
    ####################################################################################
    ################################# Objective set up #################################
    ####################################################################################
    objective_expression = generate_production_cost_expression(Xsi, data_df, S, I, verbose=verbose)
    m.setObjective(objective_expression, GRB.MINIMIZE)
    print()
    ####################################################################################
    ################################# Constraint set up ################################
    ####################################################################################
    production_constraints(m, Xsi, 10, S, I, verbose=verbose)
    print()
    product_constraints(m, Xsi, product_limits, S, I, verbose=verbose)


    ####################################################################################
    ################################# SOLVE:OPTIMIZE ###################################
    ####################################################################################


    m.optimize()

    ####################################################################################
    ################################# Display Results ##################################
    ####################################################################################
    print("Optimized Variables: ")
    #displayDecisionVars(m, end_sentinel="2")

    print("\n-------------Optimal Objective value: Does it make sense?----------------------")
    print('Obj: {:.2f}K'.format(m.ObjVal))

    print("\n----------------------Solution, object weight, reduced costs, objective weight ranges")
    for v in m.getVars():
        print("{:s} => {:.2f},\nobj: {},\nreduced cost: {},\nrange:{:.3f}--{:.3f}\n\n".format(v.VarName,
                                                                                              v.X, v.obj,
                                                                                              v.RC,
                                                                                              v.SAObjUp, v.SAObjLow))


    print("\n----------------------Shadow Prices and ranges:")
    for c in m.getConstrs():
        lhsval = m.getRow(c).getValue()
        outputstr = "constraint: {}\n\tlhs: {}\n\tshadowPrice: {}\n\tRHS: {}\n\tRanges: {} -- {}\n\n"
        print(outputstr.format(c.ConstrName, lhsval, c.Pi, c.RHS, c.SARHSUp, c.SARHSLow))





# catch some math errors
except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

```
Restricted license - for non-production use only - expires 2023-10-25



Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (win64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 5 rows, 6 columns and 12 nonzeros
Model fingerprint: 0x697a04d9
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [5e+00, 1e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [5e+00, 1e+01]
Presolve time: 0.00s
Presolved: 5 rows, 6 columns, 12 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.900000e+01   0.000000e+00      0s
       5    1.2800000e+02   0.000000e+00   0.000000e+00      0s

Solved in 5 iterations and 0.01 seconds (0.00 work units)
Optimal objective  1.280000000e+02
Optimized Variables:

-------------Optimal Objective value: Does it make sense?----------------------
Obj: 128.00K

----------------------Solution, object weight, reduced costs, objective weight ranges
Xsi[0,0] => 6.00,
obj: 5.0,
reduced cost: 0.0,
range:6.000---2.000


Xsi[0,1] => 0.00,
obj: 6.0,
reduced cost: 1.0,
range:inf--5.000


Xsi[0,2] => 4.00,
obj: 8.0,
reduced cost: 0.0,
range:9.000--7.000


Xsi[1,0] => 0.00,
obj: 8.0,
reduced cost: 1.0,
range:inf--7.000


Xsi[1,1] => 8.00,
obj: 7.0,
reduced cost: 0.0,
range:8.000--0.000


Xsi[1,2] => 1.00,
obj: 10.0,
reduced cost: 0.0,
range:11.000--9.000



----------------------Shadow Prices and ranges:
constraint: capacity constraint plant: 0
        lhs: 10.0
        shadowPrice: -2.0
        RHS: 10.0
        Ranges: 11.0 -- 9.0


constraint: capacity constraint plant: 1
        lhs: 9.0
        shadowPrice: 0.0
        RHS: 10.0
        Ranges: inf -- 9.0


constraint: product 1 constraint
        lhs: 6.0
        shadowPrice: 7.0
        RHS: 6.0
        Ranges: 7.0 -- 5.0


constraint: product 2 constraint
        lhs: 8.0
        shadowPrice: 7.0
        RHS: 8.0
        Ranges: 9.0 -- -0.0


constraint: product 3 constraint
        lhs: 5.0
        shadowPrice: 10.0
        RHS: 5.0
        Ranges: 6.0 -- 4.0
```

Solution Discussion

> **The solution suggest to produce 6K and 4K units of products 1 and 3 repsectively at plant 1, and 8K and 1K units of products 2 and 3 respectively at plant 2. The solution shows that you should produce the products where their costs are lower. The costs of producing products 1 and 3 are cheaper at plant 1, but since each plant can only produce 10K units each and there must be at least 5k units of product 3 produced the last unit is produced at plant 2. Since the cost of producing product 2 at plant 2 is lower than at plant 1 it makes sense to make all of the required 8k units at plant 2. This solution leads to an optimal cost of 128K.**

Questions:

Question 1:

- What would the cost of producing product 2 at plant 1 have to be for Gepbab to start producing there?
  - **Answer:** The reduced cost for producing product 2 at plant 1 (Xsc[0,1]) is 1. From this the **price would need to be reduced by 1, i.e. 6-1 = 5** so that this option will be see a positive coefficient in the solution.

Question 2:

- What would the total cost be if plant 1 had 9k units of capacity?
  - **Answer:** The shadow price of the capacity constraint for plant one is -2 indicating that increasing the capacity by one unit would decrease the cost by two. From this it seems that decreasing the capacity by 1 from 10K to 9k would increase the cost by 2 from 128K to 130K

Question 3:

- If it cost $9 to produce a unit of product 3 at plant 1, then what would be the new optimal solution
  - **Answer:** The reduced cost for producing product 3 at plant 1 (Xsc[0,2]) is 0 with an objective coefficient range from 9 to 7. This means that increasing it up to 9 will not change the solution so the solution would be the same.

In [5]:
```
# save the notebook as a pdf
# to_PDF("notebook_title")
```