Report on Project 1: Developing Optimized Work Schedule for Metal Co

by

Md Hishamur Rahman

Student ID: 1400077

Umail: hisham.rahman@utah.edu

University of Utah

20th September, 2022

Contents

Executive Summary

Metal Co produces different metal products in a factory that require an optimized weekly work schedule and inventory policy for the plant. This report developed a mathematical model based on linear programming that can be solved optimally using open-source software to produce the required work schedule the plant. Experiment based on the data provided from the plant revealed that the model developed in this report can produce annually $13.33 million additional profit to the company.

Introduction

Metal Co manufactures various metal products from raw steel, i.e., nails, screws, pipes, flashing, rebar, and conduit, which require an optimized work schedule for the plant over a week. The company has resource limitation in terms of how much raw steel it can process to manufacture the metal products and it has daily demand requirements for each metal product that has to be fulfilled. The current profit of the company is 2 million per week based on the current revenue and costs and the optimized work schedule is expected to make more profit than the current baseline.

The abovementioned problem of the company is addressed by modeling their process mathematically using a linear programming approach. The model considered various constraints based on the manufacturing process, resource limitation of the company, and the demand requirement of the clients. The objective of the mathematical model is to maximize the profit considering the revenue from selling the products and costs related to production, inventory, and backorder of the products. The resulting mathematical model is solved optimally using Pulp, a Python package for solving linear programs. Experiments based on the weekly data provided from the factory shows that the proposed mathematical model can produce a savings of $2.26 million per week.

Data and Assumptions

The data provided for this project includes the weekly demand data for each metal, selling price and manufacturing cost for each metal, raw material cost, inventory cost, and backorder cost. The company also informed that it can process up to 500 tons of steel per week to produce the metals. It is assumed that there is no starting inventory and backorder.

The factory is assumed to have the following demand (in tons) for the metals over the next week:

Demand in days	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Nails	50	40	60	40	50	30	30
Screws	90	70	80	70	70	40	40
Pipe	90	60	70	60	80	40	30
Flashing	70	60	80	40	120	60	30
Rebar	250	75	75	75	300	75	75
Conduit	150	0	0	0	150	0	0
Total	700	305	365	285	770	245	205

It is noteworthy to mention that the company has more demand than it could produce on Monday and Friday. This means that on Monday it would have some backorder since the starting inventory is zero. However, the excess demand on Friday can be fulfilled from the inventory.

The metal products are assumed to have the following selling cost and manufacturing cost over the next week:

Products	Selling price ($) per ton	Manufacturing cost ($) per ton
Nails	3000	100
Screws	3500	300
Pipe	2800	200
Flashing	4000	600
Rebar	2400	50

Conduit	3000	250

However, the manufacturing cost does not include the raw material cost for steel. The raw material cost for raw steel is assumed to be $2000 per ton. Furthermore, the inventory cost for a metal is $20 per day. To cover the expected loss of customer due to backorder, any item on backorder is charged 2% of its selling price every day.

Mathematical Model

The mathematical model for this project is developed based on linear programming approach. The model includes a set of decision variables that needs to be solved, a set of constraints to meet the limits and requirements of the factory, and an objective function that involves profit maximization. They are described below with equation along with the Python code snippets.

Decision Variables

The decision variables for this model are as follows:

Production of metal i on day j → Pij
Inventory balance of metal i on day j → Iij
Backorder of metal i on day j → Bij
Number of items sold for metal i on day j → Sij

The above decision variables are coded in Pulp as follows:

```
prod= pulp.LpVariable.dicts("production", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
invent= pulp.LpVariable.dicts("inventory", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
back= pulp.LpVariable.dicts("backorder", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
sold= pulp.LpVariable.dicts("sold", ((i, j) for i in range(numproducts) for j in range(numdays)), lowBound=0, cat = 'Continuous')
```

Constraints

The company can process up to 500 tons of steel per day to produce metals, which indicates that the total production of metals on each day j is constrained to 500 tons. This can be expressed as follows:

$$\sum_{i} Pij \leq 500 \quad \forall j$$

```
for j in range (numdays):
    my_lp_problem += pulp.lpSum([prod[i, j] for i in range(numproducts)]) <=500
```

The number of items sold in any day j for a product i should meet its corresponding demand (dij) for that day; however, if the production is less than demand then sometimes it is also possible to sell less than demand. This can be expressed as follows:

$$Sij \leq dij \quad \forall i, \forall j$$

```
for i in range (numproducts):
    for j in range (numdays):
        my_lp_problem += sold[i, j] <= demand[i][j]
```

The sum of production and backorder for a metal i on a day j and its inventory balance from the previous day should be equal to sum of the items sold for that metal in that day, the inventory balance of that metal on that day, and the backorder of the previous day. This can be expressed as:

$$Pij + Ii,j-1 + Bij \leq Sij + Iij + Bi,j-1 \quad \forall i \geq 0, \forall j \geq 1$$

```
for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += prod[i, j] + invent[i,j-1] + back[i,j] == sold[i,j] +  invent[i,j] + back[i,j-1]
```

Since the above constraint is applied from 2nd day of the week (j≥1), we need specify the constraint for the first day separately (j=0). The first day will not have any inventory balance and backorder from the previous day and they are taken to be zero. Therefore, the resulting constraint for the first day can be formulated and coded as follows:

$$Pi,0 + Bi,0 \leq Si,0 + Ii,0 \quad \forall i$$

```
for i in range (numproducts):
    my_lp_problem += prod[i, 0] + back[i,0] == sold[i,0] +  invent[i,0]
```

Finally, the company would want their final inventory balance and backorder at the end of week to be zero so that they can minimize their cost. This can be expressed as follows:

$$Ii,6 = 0$$

$$Bi,6 = 0$$

```
for i in range(numproducts):
    my_lp_problem += invent[i,numdays-1] ==0
    my_lp_problem += back[i,numdays-1] ==0
```

Objective Function

The objective of the mathematical model is to maximize the profit; i.e., ∑revenue - ∑cost, which can be expressed as follows:

$$\sum_{ij} Ri \times Sij - \sum_{ij} 2000 \times Pij - \sum_{ij} Mi \times Pij - \sum_{ij} 20 \times Iij - \sum_{ij} 0.02 \times Ri \times Bij$$

where, Ri is the selling price for metal i; 2000 is the cost of raw material; Mi is the manufacturing cost for metal i; 20 is the inventory cost; and 0.02 is the percentage charge on backorder items.

```
my_lp_problem += pulp.lpSum([sellingprice[i]*sold[i, j] for i in range(numproducts) for j in range(numdays)])
               - pulp.lpSum([2000*prod[i, j] for i in range(numproducts) for j in range(numdays)])
               - pulp.lpSum([manucost[i]*prod[i, j] for i in range(numproducts) for j in range(numdays)])
               - pulp.lpSum([20*invent[i, j] for i in range(numproducts) for j in range(numdays)])
               - pulp.lpSum([0.02*sellingprice[i]*back[i, j] for i in range(numproducts) for j in range(numdays)])
```

The Python code for the mathematical model is provided in the Appendix.

Analysis

The developed mathematical model is optimized and solved using Pulp package in Python. The optimum solution revealed that it is possible to save $2.26 million in a week using the provided mathematical model. The solution also provides the optimum schedule for production, inventory balance, backorder, and number of items sold for each metal over the week, which are tabulated as follows:

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Items
Production								
	50	40	60	90	0	30	30	Nails
	90	70	80	70	70	40	40	Screws
	90	60	125	5	80	40	30	Pipe
	70	60	80	160	0	60	30	Flashing
	50	270	80	175	200	75	75	Rebar
	150	0	0	0	150	0	0	Conduit
Inventory								
	0	0	0	50	0	0	0	Nails
	0	0	0	0	0	0	0	Screws
	0	0	55	0	0	0	0	Pipe
	0	0	0	120	0	0	0	Flashing
	0	0	0	100	0	0	0	Rebar
	0	0	0	0	0	0	0	Conduit
Backorder								
	0	0	0	0	0	0	0	Nails
	0	0	0	0	0	0	0	Screws
	0	0	0	0	0	0	0	Pipe
	0	0	0	0	0	0	0	Flashing
	200	5	0	0	0	0	0	Rebar
	0	0	0	0	0	0	0	Conduit
Sold								
	50	40	60	40	50	30	30	Nails
	90	70	80	70	70	40	40	Screws
	90	60	70	60	80	40	30	Pipe
	70	60	80	40	120	60	30	Flashing
	250	75	75	75	300	75	75	Rebar
	150	0	0	0	150	0	0	Conduit

Some important policy decisions can be made based on the abovementioned work schedule. They are as follows:

1. It is observed that there is a high demand (700 tons) of metals products on the first day of the week (Monday) due to greater demand of rebar and conduit. However, there is no

inventory balance on the first day of the week and there is a limit in the total production due to raw material processing capacity (500 tons/day). As a result, a large part of the demand of rebar on the first day has to be fulfilled as a backorder on the next day (highlighted in red).

2. The demand is also on Friday (770 tons) mainly because of higher demand of rebar, conduit, and flashing. However, this demand can be tackled using the previous day inventory of nails, flashing, and rebar. Because, of the inventory for these items, the company can focus more on producing the greater demand items.

Conclusion

This report provided a mathematical model for optimizing the work schedule of metal products at Metal Co factory. The mathematical model used a linear programming approach that was coded and solved using the Pulp package in Python. The analysis on data of the factory for a week showed that the mathematical model is capable of producing $2.26 million per week, which is a 13% increase over the previous profit baseline.

There are a few recommendations and scope for future work that might be useful for generating further profits. The company can reduce the cost if they can use an inventory for the first day of the week since the charge for backorder is more than the inventory storage cost. Furthermore, the company can reduce the inventory cost if they can process more raw steel to increase the production of flashing and rebar. In future, further analysis can be done on a dataset over larger periods to assess the profitability trend. Additionally, the model improvement can be tested by imposing additional constraints on inventory and backorder.

Appendix

```
import pulp
import openpyxl
import random


def readexcelfile(path, demand, manucost, sellingprice, products, daynames,numdays,
numproducts):

    wb = openpyxl.load_workbook(filename = path)
    sheetnames = wb.sheetnames
    wsheet =wb[sheetnames[0]]
    for i in range (2, 2+numproducts):

        names.append(wsheet[i][0].value)
        sellingprice.append(float(wsheet[i][1].value))
        manucost.append(float(wsheet[i][2].value))

    for j in range (1, 1+numdays):
       daynames.append(wsheet[10][j].value)

    for i in range (11, 11+numproducts):
        temp=[]
        for j in range (1, 1+numdays):
            temp.append(float(wsheet[i][j].value))
        demand.append(temp)


numproducts = 6
numdays = 7
path ="project1data.xlsx"
demand=[]
names=[]
daynames=[]
manucost=[]
sellingprice=[]
totdemand = [700,305,365,285,770,245,205]



readexcelfile(path, demand, manucost, sellingprice,names, daynames,numdays,
numproducts)

print ("demand is \n",demand)
print ("Products are \n",names)
print("Day names are \n",daynames)
print ("Manufacturing costs are \n",manucost)
print("selling prices are \n",sellingprice)

#Then instantiate a problem class, we'll name it "My LP problem" and we're looking
for an optimal maximum so we use LpMaximize
```

```
my_lp_problem = pulp.LpProblem("My LP Problem", pulp.LpMaximize)

prod= pulp.LpVariable.dicts("production", ((i, j) for i in range(numproducts) for j
in range(numdays)), lowBound=0, cat = 'Continuous')
invent= pulp.LpVariable.dicts("inventory", ((i, j) for i in range(numproducts) for j
in range(numdays)), lowBound=0, cat = 'Continuous')
back= pulp.LpVariable.dicts("backorder", ((i, j) for i in range(numproducts) for j
in range(numdays)), lowBound=0, cat = 'Continuous')
sold= pulp.LpVariable.dicts("sold", ((i, j) for i in range(numproducts) for j in
range(numdays)), lowBound=0, cat = 'Continuous')

# obj and constr are added using += to the my_lp_problem class.

# Pulp assumes that the the objective function is alsyws given first.

my_lp_problem += pulp.lpSum([sellingprice[i]*sold[i, j] for i in range(numproducts)
for j in range(numdays)]) - pulp.lpSum([2000*prod[i, j] for i in range(numproducts)
for j in range(numdays)]) - pulp.lpSum([manucost[i]*prod[i, j] for i in
range(numproducts) for j in range(numdays)]) - pulp.lpSum([20*invent[i, j] for i in
range(numproducts) for j in range(numdays)]) -
pulp.lpSum([0.02*sellingprice[i]*back[i, j] for i in range(numproducts) for j in
range(numdays)])

# constraints
for j in range (numdays):
    my_lp_problem += pulp.lpSum([prod[i, j] for i in range(numproducts)]) <=500

for i in range (numproducts):
    for j in range (numdays):
        my_lp_problem += sold[i, j] <= demand[i][j]

for i in range (numproducts):
    for j in range (1,numdays):
        my_lp_problem += prod[i, j] + invent[i,j-1] + back[i,j] == sold[i,j] +
invent[i,j] + back[i,j-1]

for i in range (numproducts):
    my_lp_problem += prod[i, 0] + back[i,0] == sold[i,0] +  invent[i,0]

for i in range(numproducts):
    my_lp_problem += invent[i,numdays-1] ==0
    my_lp_problem += back[i,numdays-1] ==0

print (my_lp_problem)

status=my_lp_problem.solve()
if pulp.LpStatus[my_lp_problem.status] =='Infeasible':
    print ('INFEASIBLE ************************************')
```

```
else:
    # To write out to an excel file


    wbr = openpyxl.Workbook()

    #Sheets start at 0

    wbr.create_sheet(index = 0, title = "solution")
    sheetnames = wbr.sheetnames

    wsheet = wbr[sheetnames[0]]

    #Writing to individual cells
    # Cells start at 1, 1

    #set column widths
    letters=['A','B','C','D','E','F','G','H','I','J','K','L']
    widths = [12, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8 ]
    for i in range (0,10):
        wsheet.column_dimensions[letters[i]].width = widths[i]

    #I am using row to increase as I write out stuff and then columns.

    row =1
    wsheet.cell(row,1).value ='Money Made'
    wsheet.cell(row,2).value =pulp.value(my_lp_problem.objective)
    row+=1

    for i in range (0, len(daynames)):
        wsheet.cell(row , i+2).value = daynames[i]
    row+=1
    wsheet.cell(row,1).value ='Production'
    row+=1
    for i in range (0,len(demand)):
        for j in range (0,len(demand[i])):
            wsheet.cell(row, j+2).value =prod[i,j].varValue
        wsheet.cell(row, j+3).value =names[i]
        row+=1

    wsheet.cell(row,1).value ='Inventory'
    row+=1
    for i in range (0,len(demand)):
        for j in range (0,len(demand[i])):
            wsheet.cell(row, j+2).value =invent[i,j].varValue
        wsheet.cell(row, j+3).value =names[i]
        row+=1

    wsheet.cell(row,1).value ='Backorder'
    row+=1
```

```
for i in range (0,len(demand)):
    for j in range (0,len(demand[i])):
        wsheet.cell(row, j+2).value =back[i,j].varValue
    wsheet.cell(row, j+3).value =names[i]
    row+=1

wsheet.cell(row,1).value ='Sold'
row+=1
for i in range (0,len(demand)):
    for j in range (0,len(demand[i])):
        wsheet.cell(row, j+2).value =sold[i,j].varValue
    wsheet.cell(row, j+3).value =names[i]
    row+=1

wbr.save('readinoutputexcel.xlsx')
```