EASE OF LOGISTICS

MINI PROJECT REPORT

Submitted by

AKSSHAYA B (202009002)

AMIRTHAVARSHINI B (202009004)

BHUVANA S (202009008)

MANJU BALA S (202009026)

in

19AD552 – MACHINE LEARNING TECHNIQUES LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

NOVEMBER 2022

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATASCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **“AKSSHAYA B (202009002) , BHUVANA S (202009008), AMIRTHAVARSHINI B (202009004)** and **MANJU BALA S (202009026)”** for the mini project titled **“EASE OF LOGISTICS”** in 19AD552 – Machine Learning Techniques Laboratory during the fifth semester July 2022 – November 2022 under my supervision.

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Asso. Professor (Senior Grade)& Head,
AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Asso. Professor (Senior Grade)& Head,
AI&DS Department
Mepco Schlenk Engg. College, Sivakasi

ABSTRACT

Logistics management deals with the coordination of resources in an organization. Logistics management focuses on the organization as a whole and not on individual units and departments while deciding about the allocation of resources The resources may be in the form of men, machines, materials, money and time. Logistics management helps in the efficient use and deployment of the scarce resources. In absence of effective logistics management, there will be a depletion of various meager resources. Logistics focuses on the movement and storage of items in the supply chain. Supply chain management (SCM) is more comprehensive, covering all of the coordination between partners that have a role in this network, including sourcing, manufacturing, transporting, storing and selling. Therefore, this project is about solving problems encountered in some Supply Chain Analytics and design class assignments, thus minimizing tedious manual work on spreadsheet.

ACKNOWLEDGEMENT

We would like to express my special thanks of gratitude to our project guide **Dr. J. Angela Jennifa Sujana ,** Head of the Department , Artificial Intelligence and Data Science as well as **Mrs. L. Prasika** , Assistant Professor , Artificial Intelligence and Data Science who gave us the wonderful opportunity to do this wonderful project on the topic **“EASE OF LOGISTICS”** , which also helped us in doing a lot of research and we came to know about so many things .

We are really thankful to them.

Secondly , We would also like to thank our parents and staffs who helped us a lot in finishing this project within the limited time.

We were making this project not only for marks but also to gain knowledge.

Thanks all again who helped us.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PROBLEM STATEMENT

The Problem is to design a system which should be able to find the shortest path to reach the destination from the source and also it should be able to design class assignments, to minimize tedious manual work on spreadsheet.

1.2 NEED FOR THE SOLUTION

Logistics entails delivering

- Right Product
- In the Right quantity
- And Right condition
- To the Right Place
- At the Right time
- To the Right Customer
- And at the Right place

1.3 OBJECTIVES

- To improve Supply Chain Efficiency
- Inventory Management
- To fulfill customer requirements
- To mitigate product damage
- To reduce Operational Cost
- Quick Response
- To optimize delivery performance

CHAPTER 2

LITERATURE REVIEW

- Global Logistics Network Modelling and Policy | ScienceDirect
- Logistics and Benefits of Using Mathematical Models of Hydrologic and Water Resource Systems | ScienceDirect
- Transportation Research Part E: Logistics and Transportation Review | Journal | ScienceDirect.com by Elsevier
- Cleaner Logistics and Supply Chain | Journal | ScienceDirect.com by Elsevier
- Logistics Operations and Management | ScienceDirect

CHAPTER 3

SYSTEM DESIGN

3.1 PROPOSED WORK

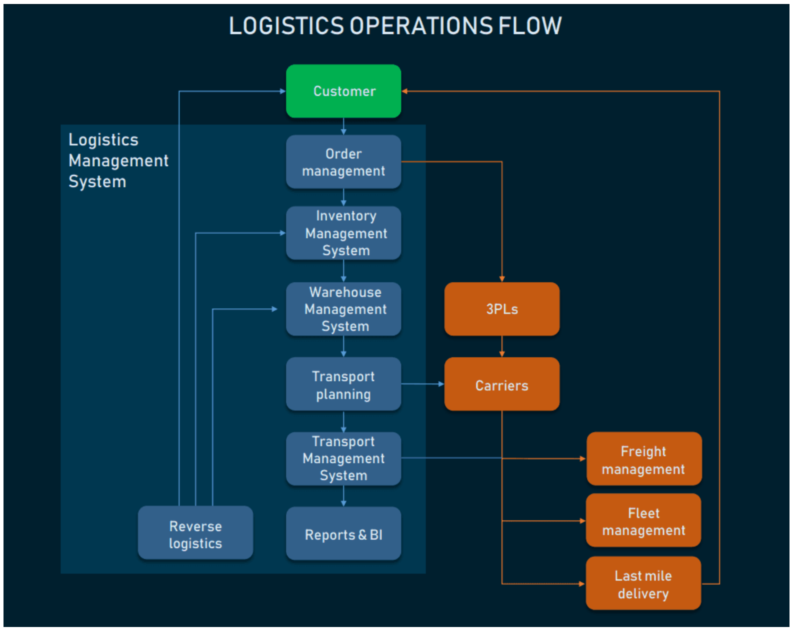

Figure.3.1.1. Work Flow

3.2 SYSTEM ARCHITECTURE DIAGRAM

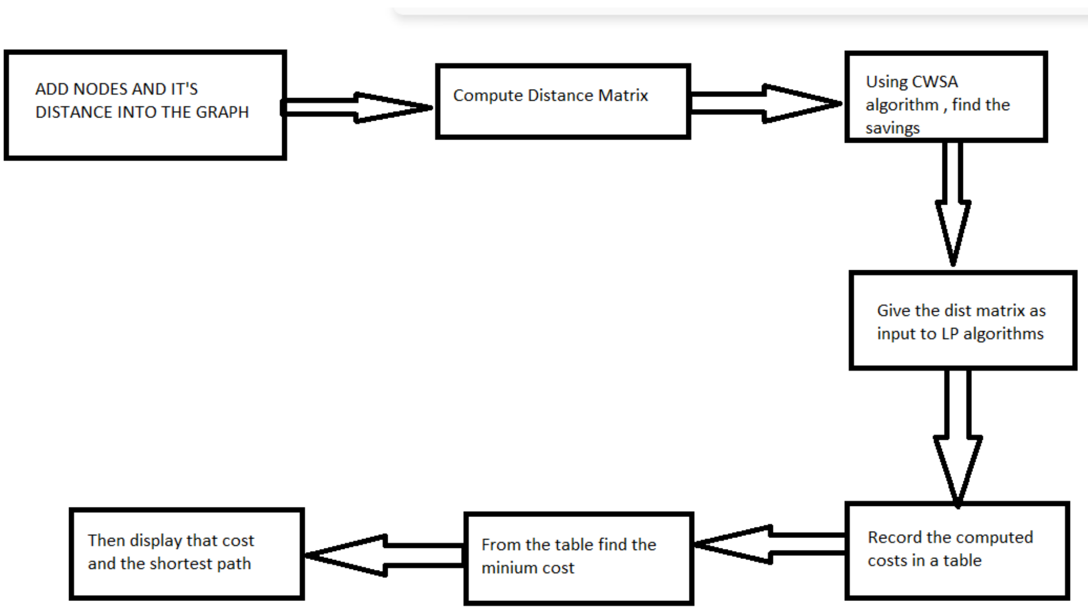

Figure.3.2.1. System Design Architecture

CHAPTER 4

IMPLEMENTATION

4.1. ALGORITHM

- Clarke-Wright Savings Algorithm for Vehicle Routing Problem
- Mixed Integer Linear Programming for Master Production Schedule
- One Time Run for Master Production Schedule
- Lot for Lot (Chase) for Master Production Schedule
- Silver Meal for Master Production Schedule
- Fixed Order Quantity (FOQ) for Master Production Schedule
- Periodic Order Quantity (POQ) for Master Production Schedule

4.2 TOOLS USED

LANGUAGE : PYTHON

PACKAGES : NumPy , Pandas ,PuLP

CHAPTER 5

CODING

5.1. PROJECT FOLDER STRUCTURE

scanalytics.py	04-11-2022 12:44	Python.File	17 KB
MPS MILP.lp	04-11-2022 10:13	LP File	2 KB
Sample Notebook	04-11-2022 10:13	Jupyter Source File	14 KB

scanalytics.py

```
import numpy as np
import operator
from IPython.display import display
import pandas as pd
from pulp import *

#Clarke-Wright Savings Algorithm
class CWSA(object):
    '''
    argument:
    create an object with 'distances' attributes.
    self.distances[(from_node,to_node)] = distance
    output:
    '''
    def __init__(self):
        self.distances = {}

    def add_dist(self, from_node,to_node,distance):
        if from_node != 'DC' and to_node != 'DC':
            if from_node < to_node:
                self.distances[(from_node,to_node)] = distance
            else:
                self.distances[(to_node,from_node)] = distance
        elif from_node == 'DC':
            self.distances[(to_node,from_node)] = distance
        elif to_node == 'DC':
            self.distances[(from_node,to_node)] = distance

def CWSA_dist_matrix(cwsa):
    '''
    argument: cwsa object

    output:
```

```
    CWSA_mtx (numpy array): rows = from_node
                            columns = to_node
                            entries = distance (above diagonal element)
                                 diagonal and below diagonal elements are 0
    '''
    from_list = []
    dist_dict = cwsa.distances
    for from_node,to_node in dist_dict:
        if from_node not in from_list:
            from_list.append(from_node)
    from_list.sort()
    CWSA_mtx = np.zeros((len(from_list),len(from_list)+1))

    for from_node,to_node in dist_dict:
        if to_node != 'DC':
            CWSA_mtx[from_node-1,to_node-1] = dist_dict[(from_node,to_node)]
        else:
            CWSA_mtx[from_node-1,-1] = dist_dict[(from_node,to_node)]
    return CWSA_mtx

def CWSA_savings(cwsa):
    '''
    Given cwsa object, provide savings and distance table of
    argument:
    (object): cwsa object with complete distances attribute added by add_dist
          function
    output:
    CWSA_dict(dataframe): 1st column   = index
                          2nd column   = (from_node,to_node)
                          3rd column   = distance/cost saving for these nodes
    CWSA_mtx (dataframe): distance/cost (above diagonal element) and
                          saving (below diagonal element) of each pair of nodes
    '''
    CWSA_mtx = CWSA_dist_matrix(cwsa)
    CWSA_dict = {}
    for i in range(np.shape(CWSA_mtx)[0]):
        for j in range(i+1,np.shape(CWSA_mtx)[0]):
            saving = CWSA_mtx[i,-1] + CWSA_mtx[j,-1] - CWSA_mtx[i,j]
            CWSA_mtx[j,i] = saving
            CWSA_dict[(i+1,j+1)] = saving
    CWSA_list = sorted(CWSA_dict.items(),key=operator.itemgetter(1),
                    reverse=True)
    CWSA_savings_df = pd.DataFrame(CWSA_list)
    CWSA_df = pd.DataFrame(CWSA_mtx)
    return CWSA_df,CWSA_savings_df
```

```
def MPS_MILP(demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using Mixed Integer Linear Programming
    argument:
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
    output:
    status (string): status of mixed integer linear programming
    inventory (list): inventory for each time period
    prod_schedule (list): production schedule for each time period
    total_cost (float): total cost of manufacturing and inventory
    '''

    #Problem statement
    prob = LpProblem('project',LpMinimize)

    #Variables
    prod_vars = ['Z'+str(i) for i in range(len(demand_forecast))]
    prod_vars_lp = LpVariable.dicts('Var',prod_vars,0,1,LpInteger)
    prod_qty_vars = ['Q'+str(i) for i in range(len(demand_forecast))]
    prod_qty_vars_lp = LpVariable.dicts('Var',prod_qty_vars,0,None)
    inventory_vars = ['I'+str(i) for i in range(len(demand_forecast))]
    inventory_vars_lp = LpVariable.dicts('Var',inventory_vars,0,None)

    # The objective function
    total_setup_cost = []
    for i in range(len(demand_forecast)):
        total_setup_cost.append(prod_vars_lp['Z'+str(i)]*setup_cost)

    total_holding_cost = []
    for i in range(len(demand_forecast)):
        total_holding_cost.append((inventory_vars_lp['I'+str(i)])*holding_cost)

    total_cost = total_setup_cost + total_holding_cost
    prob += lpSum(total_cost), 'Total Cost'

    #Inventory balance
    for i in range(1,len(demand_forecast)):
        prob += lpSum(prod_qty_vars_lp['Q'+str(i)]
                   -demand_forecast[i]
                   +inventory_vars_lp['I'+str(i-1)]
                   -inventory_vars_lp['I'+str(i)]) == 0
```

```
    prob += lpSum(prod_qty_vars_lp['Q0']
                  -demand_forecast[0]
                  +init_inventory
                  -inventory_vars_lp['I0']) == 0

    #Linking constraint
    M = np.sum(demand_forecast)
    for i in range(0,len(demand_forecast)):
        prob += M*prod_vars_lp['Z'+str(i)]-prod_qty_vars_lp['Q'+str(i)] >= 0

    #prob += prod_vars_lp['Z'+str(4)] == 1

    #Demand constraint
    for i in range(1,len(demand_forecast)):
        prob += prod_qty_vars_lp['Q'+str(i)]\
                +inventory_vars_lp['I'+str(i-1)]\
                -demand_forecast[i] >= 0

    prob += prod_qty_vars_lp['Q0']\
            +init_inventory\
            -demand_forecast[0] >= 0

    prob.writeLP('MPS MILP.lp')
    prob.solve()

    inventory = [0 for i in range(len(demand_forecast))]
    prod_schedule = [0 for i in range(len(demand_forecast))]
    for v in prob.variables():
        if v.name[4] == 'I': inventory[int(v.name[5:])] = v.varValue
        if v.name[4] == 'Q': prod_schedule[int(v.name[5:])] = v.varValue

    status = LpStatus[prob.status]
    total_cost = value(prob.objective)
    return status,inventory,prod_schedule,total_cost

def MPS_onetime(demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using One Time strategy
    argument:
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
    output:
```

```
    inventory (list): inventory for each time period
    quantity (list): quantity of product manufactured for each time period
    total_cost (float): total cost of manufacturing and inventory
    '''
    prod_qty = (np.sum(demand_forecast)-init_inventory)
    prod_schedule = [0 for i in range(len(demand_forecast))]
    prod_schedule[0] = prod_qty
    inventory = []
    for time in range(len(demand_forecast)):
        if time == 0: inventory.append(init_inventory+prod_schedule[time]-
demand_forecast[time])
        else:
            inventory.append(inventory[time-1]+prod_schedule[time]-demand_forecast[time])

    total_setup_cost = 0
    for i in prod_schedule:
        if i > 0: total_setup_cost += setup_cost

    total_holding_cost = 0
    for i in inventory:
        if i > 0: total_holding_cost += i*holding_cost

    total_cost = total_setup_cost + total_holding_cost

    return inventory,prod_schedule,total_cost

def MPS_chase(demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using Chase strategy.
    argument:
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
    output:
    inventory (list): inventory for each time period
    quantity (list): quantity of product manufactured for each time period
    total_cost (float): total cost of manufacturing and inventory
    '''

    prod_schedule = []
    inventory = []

    #First check how many time period the current inventory can hold
    init_prod = 0
```

```
    while init_prod in range(len(demand_forecast)):
        if np.sum(demand_forecast[:init_prod + 1]) <= init_inventory:
            init_prod += 1
        else: break

    #Add 0 as the production during this period of using current inventory
    for idx in range(init_prod):
        if idx == 0:
            prod_schedule.append(0)
            inventory.append(init_inventory - demand_forecast[idx])
        else:
            prod_schedule.append(0)
            inventory.append(inventory[idx-1] - demand_forecast[idx])

    for idx in range(init_prod,len(demand_forecast)):
        if not inventory:
            prod_schedule.append(demand_forecast[idx])
            inventory.append(0)
        elif inventory[idx-1] > 0:
            prod_schedule.append(demand_forecast[idx] - inventory[idx-1])
            inventory.append(0)
        else:
            prod_schedule.append(demand_forecast[idx])
            inventory.append(0)

    total_setup_cost = 0
    for i in prod_schedule:
        if i > 0: total_setup_cost += setup_cost

    total_holding_cost = 0
    for i in inventory:
        if i > 0: total_holding_cost += i*holding_cost

    total_cost = total_setup_cost + total_holding_cost

    return inventory,prod_schedule,total_cost

def MPS_silvermeal(demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using Silver Meal strategy.
    argument:
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
```

```
output:
inventory (list): inventory for each time period
quantity (list): quantity of product manufactured for each time period
total_cost (float): total cost of manufacturing and inventory
'''

prod_schedule = []
inventory = []

#First check how many time period the current inventory can hold
init_prod = 0
while init_prod in range(len(demand_forecast)):
    if np.sum(demand_forecast[:init_prod + 1]) <= init_inventory:
        init_prod += 1
    else: break

#Add 0 as the production during this period of using current inventory
for idx in range(init_prod):
    if idx == 0:
        prod_schedule.append(0)
        inventory.append(init_inventory - demand_forecast[idx])
    else:
        prod_schedule.append(0)
        inventory.append(inventory[idx-1] - demand_forecast[idx])

ix = init_prod

while ix in range(init_prod,len(demand_forecast)):
    cost = setup_cost
    if ix+1 == len(demand_forecast):
        prod_schedule.append(demand_forecast[ix])
        inventory.append(0)
        break
    next_cost = (setup_cost + demand_forecast[ix+1]*holding_cost)/2
    inventory_factor = [1*holding_cost]
    ix2 = 1
    while next_cost <= cost and ix+ix2 < len(demand_forecast):
        cost = next_cost
        ix2 += 1
        inventory_factor.append(ix2*holding_cost)
        next_cost = ((setup_cost + sum(i[0] * i[1]
                      for i in zip(demand_forecast[ix+1:ix+ix2+1],inventory_factor)))/(1+ix2))
    ix += ix2
    if not inventory:
        production = sum(demand_forecast[ix-ix2:ix])
```

```
            inventory.append(production-demand_forecast[ix-ix2])
        else:
            production = sum(demand_forecast[ix-ix2:ix])-inventory[-1]
            inventory.append(inventory[-1]+production-demand_forecast[ix-ix2])

        prod_schedule.append(production)

        for i in range(ix2-1):
            prod_schedule.append(0)
            inventory.append(inventory[-1]-demand_forecast[ix-ix2+i+1])

    total_setup_cost = 0
    for i in prod_schedule:
        if i > 0: total_setup_cost += setup_cost

    total_holding_cost = 0
    for i in inventory:
        if i > 0: total_holding_cost += i*holding_cost

    total_cost = total_setup_cost + total_holding_cost

    return inventory,prod_schedule,total_cost

def MPS_FOQ(Q,demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using Fixed Order Quantity strategy.
    argument:
    Q (float): economic order quantity
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
    output:
    inventory (list): inventory for each time period
    quantity (list): quantity of product manufactured for each time period
    total_cost (float): total cost of manufacturing and inventory
    '''
    prod_schedule = []
    inventory = []

    #First check how many time period the current inventory can hold
    init_prod = 0
    while init_prod in range(len(demand_forecast)):
        if np.sum(demand_forecast[:init_prod + 1]) <= init_inventory:
            init_prod += 1
```

```
        else: break

    #Add 0 as the production during this period of using current inventory
    for idx in range(init_prod):
        if idx == 0:
            prod_schedule.append(0)
            inventory.append(init_inventory - demand_forecast[idx])
        else:
            prod_schedule.append(0)
            inventory.append(inventory[idx-1] - demand_forecast[idx])

    prod_schedule.append(Q)
    last_prod_time = init_prod
    if init_prod == 0:
        inventory.append(init_inventory+Q-demand_forecast[init_prod])
    else:
        inventory.append(inventory[-1]+Q-demand_forecast[init_prod])

    for time in range(init_prod+1,len(demand_forecast)):
        if inventory[-1] < demand_forecast[time]:
            prod_schedule.append(Q)
            inventory.append(inventory[-1]+Q-demand_forecast[time])
            last_prod_time = time
        else:
            prod_schedule.append(0)
            inventory.append(inventory[-1]-demand_forecast[time])

    if inventory[-1] > 0:
        prod_schedule[last_prod_time] -= inventory[-1]
        inventory[last_prod_time] = inventory[last_prod_time-1]+Q-inventory[-1]-
demand_forecast[last_prod_time]
        for time in range(last_prod_time+1,len(demand_forecast)):
            inventory[time] = inventory[time-1]-demand_forecast[time]+prod_schedule[time]

    total_setup_cost = 0
    for i in prod_schedule:
        if i > 0: total_setup_cost += setup_cost

    total_holding_cost = 0
    for i in inventory:
        if i > 0: total_holding_cost += i*holding_cost

    total_cost = total_setup_cost + total_holding_cost
    return inventory,prod_schedule,total_cost
```

```
def MPS_POQ(t,demand_forecast,setup_cost,holding_cost,init_inventory):
    '''
    MPS using Periodic Order Quantity strategy.
    argument:
    t (float): interval time period between productions
    demand_forecast (list): demand for each time period
    setup_cost (float): fixed cost of setting up manufacturing
    holding_cost (float): fixed cost of init_inventory
    init_inventory(float): initial inventory
    output:
    inventory (list): inventory for each time period
    quantity (list): quantity of product manufactured for each time period
    total_cost (float): total cost of manufacturing and inventory
    '''

    prod_schedule = []
    inventory = []

    #First check how many time period the current inventory can hold
    init_prod = 0
    while init_prod in range(len(demand_forecast)):
        if np.sum(demand_forecast[:init_prod + 1]) <= init_inventory:
            init_prod += 1
        else: break

    #Add 0 as the production during this period of using current inventory
    for idx in range(init_prod):
        if idx == 0:
            prod_schedule.append(0)
            inventory.append(init_inventory - demand_forecast[idx])
        else:
            prod_schedule.append(0)
            inventory.append(inventory[idx-1] - demand_forecast[idx])

    Q = sum(demand_forecast[init_prod:init_prod+t])
    prod_schedule.append(Q)
    if init_prod == 0:
        inventory.append(init_inventory+Q-demand_forecast[init_prod])
    else:
        inventory.append(inventory[-1]+Q-demand_forecast[init_prod])
    last_prod_time = init_prod

    for time in range(init_prod+1,len(demand_forecast)):
        if time - last_prod_time < t:
            prod_schedule.append(0)
```

```
            inventory.append(inventory[-1]-demand_forecast[time])
        else:
            Q = sum(demand_forecast[time:time+t])
            prod_schedule.append(Q)
            inventory.append(inventory[-1]+Q-demand_forecast[time])
            last_prod_time = time

    if inventory[-1] > 0:
        prod_schedule[last_prod_time] -= inventory[-1]
        inventory[last_prod_time] = inventory[last_prod_time-1]+Q-inventory[-1]-
demand_forecast[last_prod_time]
        for time in range(last_prod_time+1,len(demand_forecast)):
            inventory[time] = inventory.append(inventory[time-1]-demand_forecast[time])

    total_setup_cost = 0
    for i in prod_schedule:
        if i > 0: total_setup_cost += setup_cost

    total_holding_cost = 0
    for i in inventory:
        if i > 0: total_holding_cost += i*holding_cost

    total_cost = total_setup_cost + total_holding_cost
    return inventory,prod_schedule,total_cost
```

SAMPLE NOTEBOOK.ipynb

```
from scanalytics import *
from IPython.display import display
cwsa = CWSA()
cwsa.add_dist(1,2,16.3)
cwsa.add_dist(1,3,16.5)
cwsa.add_dist(1,4,20)
cwsa.add_dist(1,5,19.6)
cwsa.add_dist(1,6,17.9)
cwsa.add_dist(1,7,9.3)
cwsa.add_dist(1,'DC',12.7)
cwsa.add_dist(2,3,7.2)
cwsa.add_dist(2,4,14.9)
cwsa.add_dist(2,5,16.6)
cwsa.add_dist(2,6,16.6)
cwsa.add_dist(2,7,12.7)
cwsa.add_dist(2,'DC',11.5)
cwsa.add_dist(3,4,8.9)
cwsa.add_dist(3,5,10.1)
```

```
cwsa.add_dist(3,6,11)
cwsa.add_dist(3,7,10.8)
cwsa.add_dist(3,'DC',9.8)
cwsa.add_dist(4,5,7.3)
cwsa.add_dist(4,6,13.4)
cwsa.add_dist(4,7,19.1)
cwsa.add_dist(4,'DC',17.5)
cwsa.add_dist(5,6,12.9)
cwsa.add_dist(5,7,16.4)
cwsa.add_dist(5,'DC',16.1)
cwsa.add_dist(6,7,9.4)
cwsa.add_dist(6,'DC',17.4)
cwsa.add_dist(7,'DC',3.6)

CWSA_df, CWSA_savings_df = CWSA_savings(cwsa)
display(CWSA_df)
display(CWSA_savings_df)
```

```
from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0

status,inventory,prod_schedule,total_cost =
MPS_MILP(demand_forecast,setup_cost,holding_cost,init_inventory)
```

```
from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0

inventory,prod_schedule,total_cost =
MPS_onetime(demand_forecast,setup_cost,holding_cost,init_inventory)
```

```
from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
```

```
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0

inventory,prod_schedule,total_cost =
MPS_chase(demand_forecast,setup_cost,holding_cost,init_inventory)


from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0

inventory,prod_schedule,total_cost =
MPS_silvermeal(demand_forecast,setup_cost,holding_cost,init_inventory)


from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0
Q = 3600

inventory,prod_schedule,total_cost = MPS_FOQ(Q,
demand_forecast,setup_cost,holding_cost,init_inventory)


from scanalytics import *
from IPython.display import display

demand_forecast = [1040,240,480,400,1600,4400,1440,1120,480,400,800,2000]
setup_cost = 1822.5
holding_cost = 0.3375
init_inventory = 0
t = 3
inventory,prod_schedule,total_cost = MPS_POQ(t,
demand_forecast,setup_cost,holding_cost,init_inventory)
```

CHAPTER 6

RESULTS AND DISCUSSIONS

	0	1	2	3	4	5	6	7
0	0.0	16.3	16.5	20.0	19.6	17.9	9.3	12.7
1	7.9	0.0	7.2	14.9	16.6	16.6	12.7	11.5
2	6.0	14.1	0.0	8.9	10.1	11.0	10.8	9.8
3	10.2	14.1	18.4	0.0	7.3	13.4	19.1	17.5
4	9.2	11.0	15.8	26.3	0.0	12.9	16.4	16.1
5	12.2	12.3	16.2	21.5	20.6	0.0	9.4	17.4
6	7.0	2.4	2.6	2.0	3.3	11.6	0.0	3.6

Figure 6.1. Distance Matrix

	0	1
0	(4, 5)	26.3
1	(4, 6)	21.5
2	(5, 6)	20.6
3	(3, 4)	18.4
4	(3, 6)	16.2
5	(3, 5)	15.8
6	(2, 3)	14.1
7	(2, 4)	14.1
8	(2, 6)	12.3

Figure 6.2. Sample sorted savings record

```
'Total Cost:'

11043.0

'Schedule:'

[2160.0, 0.0, 0.0, 0.0, 1600.0, 7840.0, 0.0, 0.0, 0.0, 0.0, 2800.0, 0.0]

'Inventory:'

[1120.0,
 880.0,
 400.0,
 0.0,
 0.0,
 3440.0,
 2000.0,
 880.0,
 400.0,
 0.0,
 2000.0,
 0.0]

'Status:'

'Optimal'
```

Figure.6.3. Total Cost and Production Schedule obtained by MILP Method

```
'Total Cost:'

30415.5

'Schedule:'

[14400, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

'Inventory:'

[13360, 13120, 12640, 12240, 10640, 6240, 4800, 3680, 3200, 2800, 2000, 0]
```

Figure.6.4. Total Cost and Production Schedule obtained by One Time Run Method

```
'Total Cost:'

21870.0

'Schedule:'

[1040, 240, 480, 400, 1600, 4400, 1440, 1120, 480, 400, 800, 2000]

'Inventory:'

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Figure.6.5. Total Cost and Production Schedule obtained by Chase Method

```
'Total Cost:'

11866.5

'Schedule:'

[2160, 0, 0, 0, 9440, 0, 0, 0, 0, 0, 2800, 0]

'Inventory:'

[1120, 880, 400, 0, 7840, 3440, 2000, 880, 400, 0, 2000, 0]
```

Figure.6.6. Total Cost and Production Schedule obtained by SilverMeal Method

```
'Total Cost:'

15228.0

'Schedule:'

[3600, 0, 0, 0, 3600, 3600, 0, 0, 3600, 0, 0, 0]

'Inventory:'

[2560, 2320, 1840, 1440, 3440, 2640, 1200, 80, 3200, 2800, 2000, 0]
```

Figure.6.7. Total Cost and Production Schedule obtained by FOQ Method

```
'Total Cost:'

13527.0

'Schedule:'

[1760, 0, 0, 6400, 0, 0, 3040, 0, 0, 3200, 0, 0]

'Inventory:'

[720, 480, 0, 6000, 4400, 0, 1600, 480, 0, 2800, 2000, 0]
```

Figure.6.8. Total Cost and Production Schedule obtained by POQ Method

CHAPTER 7
CONCLUSION

Hence, we have designed a system which is able to find the shortest path to reach the destination from the source and also it should is able to design class assignments, to minimize tedious manual work on spreadsheet. Using six algorithms for Master Production Scheduling we have found an optimal solution using the Multiple Integer Linear Programming Algoritnm which helps us to to cut down the cost for the transportation of the Inventories to a maximum extend. This application can find its efficient usage in the Production Unit of a Logistics Platform.

REFERENCES

- https://www.sciencedirect.com/book/9780080256627/logistics-and-benefits-of-using-mathematical-models-of-hydrologic-and-water-resource-systems
- https://www.sciencedirect.com/book/9780128140604/global-logistics-network-modelling-and-policy
- https://www.sciencedirect.com/journal/transportation-research-part-e-logistics-and-transportation-review
- https://www.sciencedirect.com/book/9780123852021/logistics-operations-and-management
- https://www.sciencedirect.com/journal/cleaner-logistics-and-supply-chain

****