APPENDIX A

Question on Milk Characteristics

1. Would you prefer fresh milk?
   i) Yes
   ii) No
   iii) Not sure
2. Which type of milk do you prefer?
   i) Liquid normal milk (Packaged)
   ii) Concentrated liquid milk (e.g. Milk Maid)
   iii) Milk powder
3. What kind of taste do you like most?
   i) Normal milky
   ii) Little sweetened
   iii) Thick creamy
4. How much fat do you prefer in milk?
   i) Low fat
   ii) Medium fat
   iii) High fat

Question on Delivery aspects

5. Would you be willing for spending extra money for delivery of milk at your residence?
   i) Yes
   ii) No
   iii) Not sure
6. If yes, how much of the cost of milk?
   i) 1%
   ii) 5%
   iii) 10%
   iv) 20%
   v) 30%
7. To protect the environment are you willing to purchase milk, if the milk vending machine is available within 1 kilometer of your residence but you have to use your own utensil? (like ATM)
   i) Yes
   ii) No
   iii) Not sure
8. Would you like to purchase milk online in bulk for particular function, with some extra cost?
   i) Yes
   ii) No
   iii) Not sure
9. If yes, then how much?
   i) 1%

ii) 5%
iii) 10%
iv) 20%
v) 30%

Question on Packaging Characteristics

10. What packaging type do you prefer?
    i) Loose milk (your own utensil)
    ii) Plastic (HDPE)
    iii) Tetra (example Frooti Pack)
    iv) Tin
    v) Acrylic (Bottle)
11. Would you prefer the packing to be transparent so that the milk is directly visible?
    i) Yes
    ii) No
    iii) Not sure
12. Would you prefer that the information relating to contents (nutrients and preservatives if any) is printed on the milk packaging?
    i) Yes
    ii) No
    iii) Not sure
13. What packaging size do you prefer?
    i) 250ml
    ii) 500 ml
    iii) 1000ml or more
14. Are you willing to pay more if the package is made of environment friendly material and can be recycled?
    i) Yes
    ii) No
15. If yes, then how much.
    i) 1%
    ii) 5%
    iii) 10%
    iv) 20%
    v) 30%
16. Are you willing to pay more if the package can be reused as a container or else?
    i) Yes
    ii) No
    iii) Not Sure
17. If yes, then how much.
    i) 1%
    ii) 5%
    iii) 10%
    iv) 20%
    v) 30%

18. Would you prefer a package which is easy to handle / carry (innovative packaging design)?
    i) Yes
    ii) No
    iii) Not sure

Question on Branding preference

19. Would you agree to pay extra for your favourite brand?
    i) Yes
    ii) No
    iii) Not sure
20. If yes then how much.
    i) 1%
    ii) 5%
    iii) 10%
    iv) 20%
    v) 30%

Question on Storage Preferences

21. Would you insist on milk which can be stored at room temperature with preservatives?
    i) Yes
    ii) No
    iii) Not sure
22. Would you accept milk which can be stored at cool temperature without preservatives?
    i) Yes
    ii) No
    iii) Not sure

Question on Awareness of Parag Products

23. Are you aware of the following products of Co-operative Dairy?
    - Mattha (मट्ठा)
    i) Yes
    ii) No
    - Pede (पेड़े)
    i) Yes
    ii) No
    - Kheer (खीर)
    i) Yes
    ii) No
    - Laddu (लड्डू)
    i) Yes
    ii) No
    - Ghee (घी)

i) Yes
ii) No

- Butter (मक्खन)

i) Yes
ii) No

- Khoya or Mava (खोया या मावा)

i) Yes
ii) No

- Rasgulla (रसगुल्ला)

i) Yes
ii) No

- Gulab Jamun (गुलाब जामुन)

i) Yes
ii) No

- Kalakand (कलाकन्द)

i) Yes
ii) No

- Rajbhog (राजभोग)

i) Yes
ii) No

24. Are you aware that Co-operative Dairy collects milk form the farmer in the morning and evening and does processing like pasteurisation? It does not add any preservatives.

i) Yes
ii) No
iii) Partially aware

APPENDIX B

Statistical Analysis result

Distribution of Sample according to their Age and Expenditure/ month											
	Adult Male (150)		Adult Female (150)		Men (150)		Women (150)		F	df	p
	Mean	SD	Mean	SD	Mean	SD	Mean	SD			
Age	19.53	1.455	18.62	0.834	36.21	9.1643	33.50	11.415	71.60	3	0.00

Distribution of Sample according to their Preferences											
	Adult Male, (N=150)		Adult Female (N=150)		Men, (N=150)		Women (N=150)		X^2	df	p
	N	%	N	%	N	%	N	%			
Domicile											
Rural	35	23.33%	17	11.3%	64	42.5%	28	18.8%	12.52	3	0.006
Urban	115	76.67%	133	88.7%	86	57.6%	122	81.2%			
Fresh Milk											
Yes	125	83.3%	111	74.2%	136	90.9%	94	62.5%	50.75	6	0.000
No	15	10.0%	32	21.0%	5	3.0%	47	31.2%			
Not sure	10	6.7%	7	4.8%	9	6.1%	9	6.3%			
Type of Milk											
Liquid Normal	130	86.67%	128	85.5%	132	87.9%	117	78.1%	9.06	6	0.170
Concentrated	5	3.33%	15	9.7%	9	6.1%	23	15.6%			
Milk powder	15	10%	7	4.8%	9	6.1%	10	6.3%			
Taste of Milk											
Normal	39	26.0%	46	30.6%	60	40.0%	67	44.6%	22.46	6	0.001
Little sweetened	52	34.6%	62	41.6%	48	32.0%	55	36.6%			
Thick creamy	59	39.3%	42	28.0%	42	28.0%	28	18.6%			
Fat in Milk											
Low Fat	20	13.3%	82	54.6%	57	38.0%	84	56.0%	89.39	6	0.000
Medium fat	86	57.3%	58	38.6%	78	52.0%	47	31.3%			
High Fat	40	29.33%	10	6.6%	15	10.0%	19	12.6%			

Delivery Service											
Yes	66	44.0%	45	30.0%	61	40.6%	44	29.3%	16.79	6	0.010
No	42	28.0%	69	46.0%	49	32.6%	67	44.6%			
Not sure	42	28.0%	36	24.0%	40	26.6%	39	26.0%			
Delivery Service Cost											
1%	50	33.3%	46	30.7%	58	38.7%	53	35.3%	30.60	12	0.001
5%	43	28.7%	37	24.7%	34	22.7%	40	26.7%			
10%	30	20.0%	30	20.0%	31	20.6%	37	24.7%			
20%	17	11.3%	21	14.0%	26	17.3%	20	13.4%			
30%	10	6.7%	16	10.6%	1	0.7%	0	0.0%			
Vending Machine											
Yes	64	42.7%	80	53.3%	72	48.0%	59	39.3%	11.76	6	0.067
No	58	38.7%	43	28.7%	54	36.0%	50	33.3%			
Not sure	28	18.6%	27	18.0%	24	16.0%	41	27.4%			
Online Purchase											
Yes	58	38.6%	55	36.7%	65	43.3%	57	38.0%	4.71	6	0.581
No	87	58.0%	92	61.3%	78	52.0%	85	56.7%			
Not sure	5	3.4%	3	2.0%	7	4.7%	8	5.3%			
Online cost											
1%	17	11.3%	38	25.3%	32	21.3%	21	14.0%	32.45	15	0.001
5%	16	10.7%	9	6.0%	21	14.0%	18	12.0%			
10%	12	8.0%	4	2.6%	9	6.0%	13	8.7%			
20%	11	7.3%	2	1.3%	3	2.0%	4	2.7%			
30%	2	1.3%	2	1.3%	0	0.0%	1	0.7%			

Packaging Type											
Loose milk	48	32.0%	60	40.0%	59	39.3%	48	32.0%	15.01	12	0.024
Plastic	34	22.7%	34	22.7%	48	32.0%	41	27.3%			
Tetra	31	20.7%	32	21.3%	20	13.3%	28	18.7%			
Tin	26	17.3%	17	11.3%	16	10.7%	20	13.3%			
Acrylic	11	7.3%	7	4.7%	7	4.7%	13	8.7%			
Transparent Package											
Yes	87	58.0%	90	60.0%	79	52.7%	87	58.0%	8.03	6	0.235
No	24	16.0%	17	11.4%	32	21.3%	31	20.7%			
Not Sure	39	26.0%	43	28.6%	39	26.0%	32	21.3%			
Information Content											
Yes	80	58.7%	109	72.7%	95	63.4%	79	52.6%	15.30	6	0.018
No	42	28.0%	26	17.3%	38	25.3%	53	35.4%			
Not Sure	20	13.3%	15	10.0%	17	11.3%	18	12.0%			
Package size											
250ml	50	33.3%	32	21.3%	29	19.3%	38	25.3%	15.71	6	0.015
500ml	86	57.3%	103	68.7%	97	64.7%	85	56.7%			
1000ml or more	14	9.4%	15	10.0%	42	16.0%	27	18.0%			
Recycled Package											
Yes	58	38.6%	55	36.7%	65	43.3%	57	38.0%	5.86	6	0.439
No	87	58.0%	92	61.3%	78	52.0%	85	56.7%			
Not Sure	5	3.4%	3	2.0%	7	4.7%	8	5.3%			
Recycle cost											
1%	25	16.7%	34	22.6%	36	24.0%	23	15.3%	15.75	12	0.202
5%	21`	14.0%	22	14.7%	28	18.7%	22	14.7%			
10%	18	12.0%	18	12%	23	15.3%	20	13.3%			
20%	16	10.7%	13	8.7%	6	4.0%	11	7.3%			

30%	11	7.3%	3	2.0%	4	2.7%	7	4.7%			
Reuse											
Yes	95	62.7%	69	45.5%	81	53.5%	70	46.2%	11.86	6	0.065
No	28	18.5%	43	28.5%	36	23.7%	41	27.0%			
Not Sure	27	17.8%	38	25.0%	33	21.8%	39	25.8%			
Reuse cost											
1%	19	12.5%	11	7.3%	16	10.6%	9	5.9%	15.20	12	0.230
5%	30	20.0%	15	10.0%	19	12.6%	10	6.6%			
10%	17	11.2%	17	11.2%	18	12.0%	16	10.7%			
20%	19	12.5%	20	13.3%	24	15.8%	28	18.5%			
30%	10	6.8%	6	4.0%	4	2.7%	7	4.6%			
Easy to Carry											
Yes	89	59.3%	89	59.3%	69	46.0%	69	46.0%	15.73	6	0.015
No	48	32.0%	41	27.3%	65	43.3%	67	44.7%			
Not Sure	13	8.7%	20	13.4%	16	10.7%	14	9.3%			
Mattha											
Yes	115	76.7%	102	68.0%	118	78.7%	116	77.3%	5.66	3	0.128
No	35	23.3%	48	32.0%	32	21.3%	34	22.7%			
Peda											
Yes	93	62.3%	78	52.0%	78	52.0%	95	63.7%	7.03	3	0.070
No	57	37.7%	72	48.0%	72	48.0%	55	36.3%			
Kheer											
Yes	118	78.0%	113	75.0%	123	82.0%	127	84.0%	4.64	3	0.199
No	32	22.0%	37	25.0%	27	18.0%	23	16.0%			
Laddu											
Yes	87	58.0%	108	72.0%	97	64.7%	92	61.3%	7.00	3	0.071
No	63	42.0%	42	28.0%	53	35.3%	58	38.7%			
Ghee											
Yes	114	76.0%	123	82.0%	108	72.0%	104	69.3%	7.24	3	0.064
No	36	24.0%	27	18.0%	42	28.0%	46	30.7%			

Butter											
Yes	108	72%	100	66.7%	113	75.3%	100	66.7%	3.90	3	0.271
No	42	28%	50	33.3%	37	24.7%	50	33.3%			
Khoya											
Yes	101	67.3%	80	53.3%	87	58.0%	80	53.3%	8.04	3	0.045
No	49	32.7%	70	46.3%	63	42.0%	70	46.7%			
Rasgulla											
Yes	106	70.7%	103	68.7%	95	63.6%	94	62.7%	3.13	3	0.371
No	44	29.3%	47	31.3%	55	36.4%	56	37.3%			
Gulab Jamun											
Yes	39	26.0%	50	33.3%	36	24.2%	39	26.0%	3.82	3	0.280
No	111	74.0%	100	66.7%	114	75.8%	111	74.0%			
Kalakand											
Yes	41	27.3%	46	30.7%	30	20.0%	44	29.3%	5.186	3	0.158
No	109	72.7%	104	69.3%	120	80.0%	106	70.7%			
Rajbhog											
Yes	103	68.7%	116	77.3%	121	80.7%	123	82.0%	9.18	3	0.026
No	47	31.3%	34	22.7%	29	19.3%	27	18.0%			
Brand Pay											
Yes	85	56.7%	79	52.7%	87	58.0%	76	50.6%	4.86	6	0.561
No	51	34.0%	59	39.3%	53	35.3%	55	36.7%			
Not Sure	14	9.3%	8	8.0%	10	6.7%	19	12.7%			
Brand cost											
1%	27	18.0%	44	22.6%	30	20.0%	26	17.3%	25.23	9	0.013
5%	23	15.3%	26	38.7%	25	16.7%	22	14.7%			
10%	17	11.3%	6	14.5%	18	12.0%	14	9.3%			
20%	16	10.7%	3	0.0%	14	9.3%	13	8.7%			
30%	2	1.3%	0	0.0%	0	0.0%	1	0.7%			
Storage at Room temperature											
Yes	84	56.0%	71	47.3%	84	56.0%	96	64.0%	10.50	6	0.104
No	57	38.0%	67	44.7%	59	39.3%	50	33.3%			
Not sure	9	6.0%	12	8.0%	7	4.7%	4	2.7%			
Storage at Cool Temperature											
Yes	75	50.0%	67	44.7%	89	59.3%	73	48.7%	13.07	6	0.041

No	56	37.3%	58	38.7%	33	22.0%	52	34.7%			
Not sure	19	12.7%	25	16.6%	28	18.7%	25	16.6%			
Co-operative Dairy											
Yes	96	64.0%	65	43.4%	90	60.0%	83	55.3%	15.98	6	0.013
No	37	24.7%	56	37.3%	41	27.3%	40	26.7%			
Partially aware	17	11.3%	29	19.3%	19	12.7%	27	18.0%			

APPENDIX C

<u>Questionnaire for Multi-Dimensional Scaling</u>

This study uses indirect input method for calculation of the dissimilarity matrix. The respondents were asked to give a score on the difference of quality of two brands being compared on quality, cost and availability.

Question: How different is the quality of brand A and brand B on a scale of 1 to 5.

S.N.	Brand A	Brand B	Quality	Cost	Availability
1	AMU	SUD			
2	AMU	SHU			
3	AMU	PRA			
4	AMU	SHA			
5	SUD	SHU			
6	SUD	PRA			
7	SUD	SHA			
8	SHU	PRA			
9	SHU	SHA			
10	SHA	PRA			

APPENDIX D

Questionnaire for Delphi Method

1. What can be at least eight possible low-cost marketing channels for the co-operative dairy?
2. What attributes should be selected for marketing channels?
3. Any suggestions for the marketing aspects?

APPENDIX E

Lingo programme for Vehicle Routing Problem (VRP)

```
MODEL:
SETS:
        NODE/1 2 3 4 5 6 7 8/:DEM, EARLIEST, LATEST;
        ETC(NODE, NODE):DIST;
        SERVICETIME(NODE): ST;
        CCM(NODE, NODE): X;
        SERVSTTIME(NODE):TI;
        LOAD(NODE):YI;
ENDSETS
DATA:

VCAP=2700;
VEH_COST=300;
DIST_COST=25;
DEM=0 2629 931 1250 563 1438 1896 1584 ;
EARLIEST=0 0 0 0 0 0 0 0;
LATEST=120 120 120 120 120 120 120 120;
DIST= 0  18.6 14.1 13.2 14.8 15.1 9.9 15
        18.6 0 1.7 3.1 4.6 7.5 6 3.8
        14.1 1.7 0 1.7 2.7 5.8 4.6 5.5
        13.2 3.1 1.7 0 2 6.1 3.7 6
        14.8 4.6 2.7 2 0 5.6 1.8 7.4
        15.1 7.5 5.8 6.1 5.6 0 5.1 11.1
        9.9 6 4.6 3.7 1.8 5.1 0 8.7
        15 3.8 5.5 6 7.4 11.1 8.7 0;
ST=0 10 10 10 10 10 10 10;

ENDDATA

MIN = VEH_COST*VC+DIST_COST*TR;!MINIMIZE THE FIXED AND VARIABLE
COST WITH MINIMIZED FLEET;

VC = @SUM(CCM(I,J)|I#LE#1 #AND# I#NE#J:X(I,J));!THIS GIVES COST OF TOTAL
VEHICLE USED;

TR = @SUM(CCM(I,J)|I#NE#J:DIST(I,J)*X(I,J));!THIS GIVE TOTAL TRAVELLING
COST;

@FOR(NODE(I):@FOR(NODE(J):@BIN(X(I,J)))); !GIVES BINARY VALUE IF X AS 0
OR 1;

@FOR(NODE(I):X(I,I)=0);
```

```
@FOR(NODE(J)|J#GE#2:@SUM(CCM(I,J)|I#NE#J :X(I,J))=1); !ENSURES THAT ONLY
ONE VEHICLE IN ONE MODE ENTERS THE NODE;

@FOR(NODE(J)|J#GE#2:(@SUM(NODE(I)|I#NE#J:X(I,J)))-
(@SUM(NODE(K)|K#NE#J:X(J,K)))=0); !ENSURES FLOW CONSERVATION;

@FOR(NODE(I)|I#GE#2:@FOR(NODE(J)|J#NE#I:TI(J)>=(TI(I)+ST(I)+3*DIST(I,J))*X(I,
J))); !ENSURES INCREASING SERVICE TIME AT EACH NODE;

@FOR(NODE(J)|J#GE#2:@FOR(NODE(I)|I#LE#1#AND#I#NE#J:TI(J)>=DIST(I,J)*X(I,J)
)); !ENSURES VEHICLE ARRIVING TIME OF A NODE AFTER LEAVING THE
DEPOT;

@FOR(NODE(I):EARLIEST(I)<=TI(I); !SERVICE START TIME SHOULD BE
GREATER THAN EGUAL TO EARLIEST TIME;

@FOR(NODE(I):TI(I)<=LATEST(I)); !SERVICE START TIME SHOULD BE LESS
THAN LATEST ARRIVAL TIME;

@SUM( NODE( J)| J #GT# 1: X( 1, J)) >=@FLOOR((@SUM( NODE( I)| I #GT# 1: DEM(
I))/ VCAP) + .999););!ENSURE SUFFICIENT NUMBER OF VEHICLES ARE LEAVING
DEPOT;

@FOR( NODE( I)| I #GE# 2 : @FOR(NODE(K)|K#NE#1:YI( K) >=YI( I) + DEM( K) -
VCAP + VCAP*( X( K, I) + X( I, K))- ( DEM( K) + DEM( I)) * X( K,
I)));!CONSTRAINED FOR AMMOUNT DELIVERED UUPTO CITY K;

END
```

Same programme is used for routes and sub-routes.

APPENDIX F

Python programmemeing code for k-means clustering and Cheapest Link Algorithm

```
# -*- coding: utf-8 -*-
"""
Uses distance matrix provided by GoogleDistMatrix.py script
"""

import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from ortools.constraint_solver import pywrapcp, routing_enums_pb2

# This determines how many clusters to create
depot = 6
folder = "C:\\Users\\anubha\\Milk Production\\delivery Points\\"
dist_matrix_filename = "Dist_Matrix in meters.csv"
time_matrix_filename = "Time_Matrix in seconds.csv"
demand_filename = "demand_array.csv"
xcord = "Latitude.csv"
ycord = "Longitude.csv"

factory_index = 0
number_of_vehicles_for_main = {"BIG": depot - 1}
number_of_vehicles_for_clusters = {"SMALL": 7}
label_for_main_cluster = depot
vehicle_max_distance = 100000
existing_cost = 13378.3
EMI = 170
SERVICE_TIME_PER_VEHICLE = 600 #seconds

class Vehicle():
    def __init__(self, capacity, rate, labour, max_distance):
        self.capacity = capacity
        self.rate = rate
        self.labour = labour
        self.max_distance = max_distance
    def __repr__(self):
        return "(capacity = %s, rate = %s, labour = %s, max_distance = %s)" % (self.capacity,
self.rate, self.labour, self.max_distance)
    def cost_provider(self, distance_matrix):
        return lambda x, y: int(distance_matrix[x][y]*self.rate)
```

```
# Modify these to control labour charge, capacity, rate and max travel distance for each vehicle type
vehicle_params = {
    "BIG" : Vehicle(2700, 0.025, 300, 60000),
  "BIG_LOCAL": Vehicle(2700, 0.025, 300, 40000),
    "SMALL" : Vehicle(500, 0.020, 100, 100000)
}

class Solution():
    def __init__(self):
        self.xs = []
        self.ys = []
        self.sps = []
        self.dist_matrix = []
        self.demand = []
        self.depots = []
        self.sps_for_depot = {}
        self.pair_dist = None
        self.cluster = None
        self.routes_for_cluster = {}
        self.read_data_from_files()
        self.init_vehicles()
        return

    def read_data_from_files(self):
        """ Reads demands, distance, time, geo coordinates """
        self.xs = list(map(float, open(folder + xcord).read().strip().split('\n')))
        self.ys = list(map(float, open(folder + ycord).read().strip().split('\n')))
        # X(latitude) and Y(longitude) reversed intentionally
        self.sps = np.array(list(map(list, zip(self.ys, self.xs))))
        self.dist_matrix = np.loadtxt(open(folder + dist_matrix_filename), delimiter = ',')
        self.time_matrix = np.loadtxt(open(folder + time_matrix_filename), delimiter = ',')
        self.demand = np.loadtxt(open(folder + demand_filename))

    def init_vehicles(self):
        """ Sets up vehicle types to be used for each cluster.
            Main cluster (Factory to Stockists) usually uses BIG trucks,
            while all other clusters use SMALL vehicles """
        self.vehicles_for_main = []
        for vtype, count in number_of_vehicles_for_main.items():
            for i in range(count):
                self.vehicles_for_main.append(vehicle_params[vtype])
        self.vehicles_for_clusters = []
        for vtype, count in number_of_vehicles_for_clusters.items():
            for i in range(count):
                self.vehicles_for_clusters.append(vehicle_params[vtype])
```

```
    def print_data(self):
        """ This isn't called by default, intented for debugging """
        print(self.dist_matrix)
        print(self.demand)

    def find_cluster(self):
        """ Uses K-Means clustering to produce cluster.
            K-Means will assign a label to each service point.
            Points with same label belong to the same cluster"""
        self.cluster = KMeans(n_clusters = depot).fit(self.sps)
        self.prepare_for_vrp()
        # If any of the clusters has too large combined demand, we will re-create clusters
        while max(self.get_demand_for_main_cluster()) > 2700:
            self.cluster = KMeans(n_clusters = depot).fit(self.sps)
            self.prepare_for_vrp()

    def prepare_for_vrp(self):
        """ Creates some data structure for easier processing and
            assigns a Stockist/Depot to each cluster.
            Service point with the highest demand in a cluster
            is picked as the depot for that cluster."""
        self.sps_for_cluster = {}
        for index, label in enumerate(self.cluster.labels_):
            if label not in self.sps_for_cluster:
                self.sps_for_cluster[label] = []
            self.sps_for_cluster[label].append(index)

        self.depot_for_cluster = {}
        for label in set(self.cluster.labels_):
            if factory_index in self.sps_for_cluster[label]:
                self.depot_for_cluster[label] = factory_index
            else:
                # Uncomment the line below to use the point closets to Factory as depot
                # self.depot_for_cluster[label] = min(self.sps_for_cluster[label], key = lambda x:
self.dist_matrix[0][x])
                self.depot_for_cluster[label] = max(self.sps_for_cluster[label], key = lambda x:
self.demand[x])
        print("Depots are : %s" % self.depot_for_cluster)
        self.main_cluster = list(self.depot_for_cluster.values())
        # Dairy factory should be first
        self.main_cluster.sort()
        print("Main cluster = %s" % self.main_cluster)


    def get_distance_matrix_for_cluster(self, cluster_label):
```

```
        """Create a distance matrix for all points which belong to the cluster with label = cluster_label"""
        cluster = self.sps_for_cluster[cluster_label]
        cluster_size =  len(cluster)
        distance_matrix_for_cluster = np.arange(cluster_size*cluster_size, dtype=np.float64).reshape((cluster_size, cluster_size))
        for i in range(cluster_size):
            for j in range(cluster_size):
                distance_matrix_for_cluster[i][j] = self.dist_matrix[cluster[i]][cluster[j]]

        return distance_matrix_for_cluster

    def get_time_matrix_for_cluster(self, cluster):
        """Create a travel time matrix for all points which belong to the cluster with label = cluster_label"""
        cluster_size =  len(cluster)
        time_matrix_for_cluster = np.arange(cluster_size*cluster_size, dtype=np.float64).reshape((cluster_size, cluster_size))
        for i in range(cluster_size):
            for j in range(cluster_size):
                time_matrix_for_cluster[i][j] = self.time_matrix[cluster[i]][cluster[j]]

        return time_matrix_for_cluster

    def get_demands_for_cluster(self, cluster_label):
        """Create a demand array for all points which belong to the cluster with label = cluster_label"""
        cluster = self.sps_for_cluster[cluster_label]
        cluster_size =  len(cluster)
        demands_for_cluster = np.zeros(cluster_size, dtype=np.float64)
        depots = self.depot_for_cluster.values()
        for i in range(cluster_size):
            if cluster[i] in depots:
                demands_for_cluster[i] = 0
            else:
                demands_for_cluster[i] = self.demand[cluster[i]]

        print("Demand for cluster %s is %s" % (cluster_label, demands_for_cluster))
        return demands_for_cluster

    def get_demand_for_main_cluster(self):
        """ Create a deamnd array for points in the main cluster (Factory to Depots)"""
        main_cluster = self.main_cluster
        demands = []
        # depot(stockist) -> demand of the cluster depot is in.
        demand_for_depot = {}
```

```
        for cluster_label in set(self.cluster.labels_):
            demand_for_depot[self.depot_for_cluster[cluster_label]] =
sum(self.get_demands_for_cluster(cluster_label))

        print("Demand for depots = %s" % demand_for_depot)
        for depot in main_cluster:
            if depot == 0:
                # No need to deliver to factory
                demands.append(0)
            else:
                demands.append(demand_for_depot[depot])
        print("Main cluster demand = %s" % demands)
        return demands

    def run_vrp_all(self):
        """ This is the function that creates VRP models and solves them"""
        #Distance for Dairy to cluster center and then from each center to service points in that
cluster.
        total_distance = 0.0
        main_cluster = self.main_cluster
        main_cluster_size =  len(main_cluster)
        # create a distance matrix for just the Dairy Factory + selected depots for each cluster
        self.distance_matrix_depots = np.arange(main_cluster_size*main_cluster_size,
dtype=np.float64).reshape((main_cluster_size, main_cluster_size))
        for i in range(main_cluster_size):
            for j in range(main_cluster_size):
                self.distance_matrix_depots[i][j] =
self.dist_matrix[main_cluster[i]][main_cluster[j]]

        main_cluster_demand = self.get_demand_for_main_cluster()

        #Find route from Dairy Factory to depots in each cluster.
        self.run_vrp(self.distance_matrix_depots, main_cluster_demand, self.vehicles_for_main,
main_cluster_size, factory_index)
        total_distance, main_time, total_cost =
self.calculate_total_and_print(self.distance_matrix_depots, main_cluster,
self.vehicles_for_main, label_for_main_cluster)
        #total_distance, total_time = self.calulate_and_print_all(self.distance_matrix_depots,
main_cluster, number_of_vehicles_for_main)
        max_cluster_time = 0
        #Now find route for each cluster
        for cluster_label in set(self.cluster.labels_):
            cluster = self.sps_for_cluster[cluster_label]
            start_point = cluster.index(self.depot_for_cluster[cluster_label])
            print("Calculating route for cluster %s : %s" % (cluster_label, cluster))
            print("Starting point is %s" % self.depot_for_cluster[cluster_label])
```

```
            distance_matrix_for_cluster = self.get_distance_matrix_for_cluster(cluster_label)
            demands_for_cluster = self.get_demands_for_cluster(cluster_label)
            self.run_vrp(distance_matrix_for_cluster, demands_for_cluster, self.vehicles_for_clusters, len(cluster), start_point)
            distance, time, cost = self.calculate_total_and_print(distance_matrix_for_cluster, cluster, self.vehicles_for_clusters, cluster_label)
            #distance, time = self.calulate_and_print_all(distance_matrix_for_cluster, cluster, number_of_vehicles_for_clusters)
            total_distance += distance
            total_cost += cost
            max_cluster_time = max(max_cluster_time, time)

        self.cost = total_cost + EMI
        # meters to kilometers
        total_distance = total_distance/1000
        self.total_solution_distance = total_distance

        self.total_solution_time = (main_time + max_cluster_time + 2*600)/3600
        print("Total time = (%s + %s + 2*600)/3600 = %s" % (main_time, max_cluster_time, self.total_solution_time))
        print("Total Distance of All clusters is %s KMs" % total_distance)

        #self.cost = total_distance*vehicle.rate + (len(self.vehicles_for_main) - 1)*vehicle.labour + depot*len(self.vehicles_for_clusters)*vehicle.labour + EMI + cost_for_cant
        print("Total cost = %s Rupees" % self.cost)
        self.savings = (existing_cost - self.cost)*100/existing_cost
        print("Savings = (%s - %s)*100/%s = %s" % (existing_cost, self.cost, existing_cost, self.savings) + ' %')
        print("Routes = %s" % self.routes_for_cluster)
        #print("Objective value = %s" % self.assignment.ComputeObjectiveValue())

    def replace_big_with_small(self, demands_for_cluster, vehicles):
        big_demands = [d for d in demands_for_cluster if d > vehicle_params['SMALL'].capacity]
        if big_demands:
            return [vehicle_params['BIG']]*len(vehicles)
        else:
            return vehicles

    def run_vrp(self, distance_matrix_for_cluster, demands_for_cluster, vehicles, cluster_size, start_point):
        #vehicles = self.replace_big_with_small(demands_for_cluster, vehicles)
        print("Vehicles are %s" % vehicles)
        number_of_vehicles = len(vehicles)
        demand_provider = lambda x, y: demands_for_cluster[y]
        routing = pywrapcp.RoutingModel(cluster_size, number_of_vehicles, start_point)
```

```
        self.routing = routing

        # Add vehicle constraits (This is to make sure that a vehicle isn't trying to deliver more than it's capacity)
        routing.AddDimensionWithVehicleCapacity(demand_provider, 0, [v.capacity for v in vehicles], True, "Capacity")
        # Add vehicle cost calculation (Not sure if this is used in route optimization)
        # We don't use this cost. We ill calculate cost based on routes later.
        cost_providers = [v.cost_provider(distance_matrix_for_cluster) for v in vehicles]
        for i, vehicle in enumerate(vehicles):
            routing.SetFixedCostOfVehicle(vehicle.labour, i)
            routing.SetArcCostEvaluatorOfVehicle(cost_providers[i], i)



        search_parameters = pywrapcp.RoutingModel.DefaultSearchParameters()
        search_parameters.first_solution_strategy = (routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)

        self.assignment = routing.SolveWithParameters(search_parameters)

    def calculate_total_and_print(self, distance_matrix_for_cluster, cluster, vehicles, cluster_label):
        """Once a VRP model is solved this function will go through each vehicles and print out the route for it.
          We also calculate the total distance, time and cost and return them"""
        routes = []
        number_of_vehicles = len(vehicles)
        time_matrix_for_cluster = self.get_time_matrix_for_cluster(cluster)
        total_dist = 0
        total_cost = 0
        max_time = 0
        for vehicle_id in range(number_of_vehicles):
            index = self.routing.Start(vehicle_id)
            route = []
            plan_output = 'Route for vehicle {0}:\n'.format(vehicle_id)
            route_dist = 0
            route_time = 0
            while not self.routing.IsEnd(index):
                node_index = self.routing.IndexToNode(index)
                route.append(cluster[node_index])
                next_node_index = self.routing.IndexToNode(
                    self.assignment.Value(self.routing.NextVar(index)))
                route_dist += distance_matrix_for_cluster[node_index][next_node_index]
                route_time += time_matrix_for_cluster[node_index][next_node_index]
                plan_output += ' {node_index} -> '.format(
```

```
                node_index=cluster[node_index])
            index = self.assignment.Value(self.routing.NextVar(index))

        node_index = self.routing.IndexToNode(index)
        route.append(cluster[node_index])
        routes.append(route)
        total_dist += route_dist
        cost = 0 if route_dist == 0 else (route_dist*vehicles[vehicle_id].rate + vehicles[vehicle_id].labour)
        total_cost += cost
        max_time = max(max_time, route_time)
        plan_output += ' {node_index}\n'.format(
            node_index=cluster[node_index])
        plan_output += 'Distance of the route {0}: {dist}\n'.format(
            vehicle_id,
            dist=route_dist)
        plan_output += 'Time of the route {0}: {time}\n'.format(
            vehicle_id,
            time=route_time)
        plan_output += 'Cost of the route {0}: {cost}\n'.format(
            vehicle_id,
            cost=cost)
        print(plan_output)

    self.routes_for_cluster[cluster_label] = routes
    print('Total Distance of all routes in cluster: {dist}'.format(dist=total_dist))
    print('Max Time of all routes in cluster: {time}'.format(time=max_time))
    print('Total Cost of all routes in cluster: {cost}'.format(cost=total_cost))
    return total_dist, max_time, total_cost

def plot_clusters(self):
    """ Plots dots for each service point with a color based on which cluster it belongs to.
        All points in same cluster have same color."""
    palette = sns.color_palette()
    cluster_colors = [palette[col] for col in self.cluster.labels_]
    plot_kwds = {'alpha' : 0.8, 's' : 80, 'linewidths':0}
    plt.scatter(self.ys, self.xs, c=cluster_colors, **plot_kwds)

def plot_route(self, route, color):
    """ Plots lines which conenct service points with a color based on which cluster it belongs to.
        All points in same cluster have same color."""
    for i in range(len(route)-1):
        sp1, sp2 = self.sps[route[i]], self.sps[route[i+1]]
        plt.plot([sp1[0], sp2[0]], [sp1[1], sp2[1]], c = color)
```

```
    def plot_routes(self):
        """ Uses plot_route() """
        palette = sns.color_palette()
        labels = set(self.cluster.labels_)
        # -1 is for the main cluster - Factory to depot
        labels.add(label_for_main_cluster)
        for cluster_label in labels:
            for route in self.routes_for_cluster[cluster_label]:
                self.plot_route(route, palette[cluster_label])


    def plot_final(self):
        """ Plots everything and shows it """
        self.plot_clusters()
        self.plot_routes()
        plt.show()

# Keep trying until you get a solution with a desired savings %
savings = 0
solution = None
while True:
    solution = Solution()
    solution.find_cluster()
    solution.prepare_for_vrp()
    solution.run_vrp_all()
    if solution.savings > savings:
        savings = solution.savings
        if solution.savings > 31:
            break
    print ("Max Savings = %s" % savings)
solution.plot_final()
```