

Operations Research Lecture Notes

Prepared & edited by

Tanmoy Das

Industrial Engineer & Jr. Data Scientist

Reference materials can be found at

<https://github.com/tanmoyie/Operations-Research>

<https://kaggle.com/tanmoyie>

www.linkedin.com/in/tanmoyie

How to read this document

This document is a reference material along with the topics covered in class of Operations Research (taught by Tanmoy Das). It is agreed that there are some other chapters which are also crucial for the theory course of Operations Research. However, in the class environment, only the following chapters are expected to be covered (Intro to LP, Graphical Solution, Simplex, Dual, Transportation, Machine Learning, Network Optimization, Integer Programming, Game Theory & Queuing Model¹).

Given that this document is not comprehensive, readers would find relevant materials (including Python codes, scan copies, PDF format, of theoretical and mathematical solutions of the problem discussed from other books) in the following Github repository github.com/tanmoyie/Operations-Research. Download all the pdf, py & other files from the repository to follow accordingly.

There are about fifteen (15) Python projects related to Operations Research (e.g. Travelling Salesman Problem in real world) are covered in this document. YouTube videos related to the explanations of abstruse contents in Operations Research, which involves Python Programming, can be found in <https://www.youtube.com/playlist?list=PLHyZ7Tamw-fevmrx2V3U13hPDDIUSBbi7>

Some additional Python Projects would be obtained from <https://www.linkedin.com/pulse/python-industrial-engineering-datacamp-level-3-tanmoy-das/>. More contents & YouTube videos will be added shortly. Follow the channel to get more update
www.youtube.com/channel/UC0yU0upBXyblfQ2x7uM6kzg

Reference Book:

1. Operations Research (2nd edition) by R. Panneerselvam (Pupils might find this book convenient)
2. Introduction to Operations Research (7th edition) by Lieberman (commonplace textbook)
3. Introduction to OR - deterministic model by Juraj Stacho² (an invaluable compendium)

¹ Latter two chapters is estimated to be covered by another instructor, hence, skipped for this document!

² A technical writeup

Table of Contents

| | |
|---|----|
| Introduction to Operations Research | 5 |
| Linear Programming | 6 |
| Math Problem | 6 |
| Graphical Solution | 7 |
| Simplex & Dual..... | 8 |
| Simplex Method | 9 |
| Tabular & Big M Solution | 9 |
| Unbounded & Infeasible Solution | 9 |
| Revised Simplex..... | 9 |
| Dual Problem | 9 |
| Python Projects on Simplex, Revised Simplex Optimization | 10 |
| Transportation & Assignment..... | 13 |
| NorthWest Corner Method | 14 |
| Assignment Problem | 14 |
| Python Projects on Optimization in Transportation & Assignment..... | 15 |
| Optimization in Machine Learning/ Data Science | 23 |
| Linear Regression..... | 24 |
| Robust Regression..... | 24 |
| Support Vector Machine | 24 |
| Python Projects on Machine Learning Optimization..... | 25 |
| Network Optimization | 28 |
| Min Cost..... | 29 |
| Max Flow | 29 |
| Minimum Spanning Tree | 29 |
| Python Projects on Network Optimization | 30 |
| Integer Programming..... | 38 |

Python projects in Optimization

| | |
|--|----|
| Python Code of Optimization Project 1: Revised Simplex..... | 10 |
| Python Code of Optimization Project 2: Shelf Space Optimization | 10 |
| Python Code of Optimization Project 3: Product Distribution Problem for the PuLP Modeller..... | 16 |
| Python Code of Optimization Project 4: Travelling Salesman Problem..... | 20 |
| Python Code of Optimization Project 5: Linear Regression (plot_ols.py)..... | 25 |
| Python Code of Optimization Project 6: Robust Regression compared to Linear Regression (plot_ransac.py) | 26 |
| Python Code of Optimization Project 7: Min Cost Flow problem | 30 |
| Python Code of Optimization Project 8: Max Flow..... | 31 |
| Python Code of Optimization Project 9: Airlines Network Optimization | 32 |

Introduction to Operations Research



Operations research is a discipline that deals with the application of advanced analytical methods to help make better decisions. **Operational Research always try to find the best and optimal solution to the problem.** For this purpose, objectives of the organization are defined and analyzed. These objectives are then used as the basis to compare the alternative courses of action.

Optimization Approach:

1. Define the problem of interest and gather relevant data.
2. Formulate a mathematical model to represent the problem.
3. Develop a computer-based procedure for deriving solutions to the problem from the model.
4. Test the model and refine it as needed.
5. Prepare for the ongoing application of the model as prescribed by management.
6. Implement.

Operations Research

- May involve current operations or proposed developments due to expected market shifts
- May become apparent through consumer complaints or through employee suggestions
- May be a conscious effort to improve efficiency or respond to an unexpected crisis

Linear Programming

Linear programming: The general problem of *optimizing* a linear function of several variables subject to a number of *constraints* that are linear in these variables and a subset of which restrict the variables to be non-negative.

NOTE: The general mathematical formulation of the *linear programming* problem is the set of matrix relationships as follows:

$$\min(\text{or}) \max f(x) = c^T x$$

subject to

$$Ax \leq b$$

$$x \geq 0.$$

Optimizing means obtaining the best possible mathematical solution to a given set of equations.

Math Problem

Graphical Solution

Further Reference in Intro to OR:

1. Linear Programming Math - Linear Programming from OPERATIONS RESEARCH by R. PANNEERSELVAM 2nd edition (selected pages).pdf
2. Graphical Solution - Graphical Soln Introduction to OR - deterministic model JURAJ STACHO.pdf

Simplex & Dual



Simplex Method

Tabular & Big M Solution

Big M

Unbounded & Infeasible Solution

Revised Simplex

Dual Problem

Further Reference in Simplex & Dual:

Simplex Method, Big M, Infeasible & Unbounded Solution - Simplex from Introduction to
Operations Research Lieberman.pdf

Duality - Duality from Introduction to OR - deterministic model Juraj Stacho.pdf

Revised Simplex - <https://youtu.be/e2lHyMl1IYY>

Python Projects on Simplex, Revised Simplex Optimization

Python Code of Optimization Project 1: Revised Simplex

```
1. """
2. Related YouTube Video: https://youtu.be/e2lHyM1lIYY
3. """
4. import numpy as np
5. import scipy as sp
6.
7. c = [-3, -5]
8. A = [[1, 0], [0, 2], [3, 2]]
9. b = [4, 12, 18]
10. x0_bounds = (0, None)
11. x1_bounds = (0, None)
12.
13. from scipy.optimize import linprog
14. # Solve the problem by Simplex method in Optimization
15. res = linprog(c, A_ub=A, b_ub=b, bounds=(x0_bounds, x1_bounds), method='simplex', options={"disp": True})
16. print(res)
```

Python Code of Optimization Project 2: Shelf Space Optimization

```
1. #import all relevant libraries
2.
3. import pandas as pd
4.
5. import numpy as np
6.
7. import math
8.
9. from math import isnan
10.
11. from pulp import *
12.
13. from collections import Counter
14.
15. #from more_itertools import unique_everseen
16.
17.
18.
19. sales=pd.read_csv("sales_lift.csv",header=None) #input file
20.
21. lift=sales.iloc[2:,1:]
22.
23. lift=np.array(lift)
24.
25. lift = lift.astype(np.int) # read the lifts from csv
26.
27. brands=sales.iloc[0:1,: ]
28.
29. brands=np.array(brands)
30.
31. brands=np.delete(brands,0)
```

```

32.
33. brands=brands.tolist() # read the brands from csv
34.
35. ff=Counter(brands)
36.
37. all_brands=ff.items()
38.
39. # the racks and the shelves available
40.
41. rack_shelf=[[1,1,2,3],[2,4,5,6],[3,7,8,9,10]]
42.
43.
44.
45. #define the optimization function
46.
47. prob=lpProblem("SO",LpMaximize)
48.
49.
50.
51. #define decision variables
52.
53. dec_var=LpVariable.matrix("dec_var",(range(len(lift)),range(len(lift[0]))),0,1,LpBinary)
54.
55.
56.
57. #Compute the sum product of decision variables and lifts
58.
59. prodt_matrix=[dec_var[i][j]*lift[i][j] for i in range(len(lift))
60. for j in range(len(lift[0]))]
61.
62.
63.
64.
65. #total lift which has to be maximized sum(prodt_matrix)
66.
67.
68.
69. #define the objective function
70.
71. prob+=lpSum(prodt_matrix)
72.
73.
74.
75. order=list(unique_everseen(brands))
76.
77. order_map = {}
78.
79. for pos, item in enumerate(order):
80.
81.     order_map[item] = pos
82.
83. #brands in order as in input file
84.
85. brands_lift=sorted(all_brands, key=lambda x: order_map[x[0]])
86.
87.
88.
89. #DEFINE CONSTRAINTS
90.
91. #1) Each shelf can have only one product i.e. sum (each row)<=1

```

```

92.
93. for i in range(len(lift)):
94.
95.     prob+=lpSum(dec_var[i])<=1
96.
97.
98.
99. # 2) Each product can be displayed only on a limited number of shelves i.e. Column co
nstraints
100.
101.     #Constraints are given as
102.
103.     col_con=[1,0,0,2,2,3,1,1]
104.
105.     dec_var=np.array(dec_var)
106.
107.     col_data=[]
108.
109.     for j in range(len(brands)):
110.
111.         col_data.append(list(zip(*dec_var)[j]))
112.
113.         prob+=lpSum(col_data[j])<=col_con[j]
114.
115.     #write the problem
116.
117.     prob.writeLP("SO.lp")
118.
119.     #solve the problem
120.
121.     prob.solve()
122.
123.     print("The maximum Total lift obtained is:",value(prob.objective)) # print th
e output
124.
125.     #print the decision variable output matrix
126.
127.     Matrix=[[0 for X in range(len(lift[0]))] for y in range(len(lift))]
128.
129.     for v in prob.variables():
130.
131.         Matrix[int(v.name.split("_")[2])][int(v.name.split("_")[3])]=v.varValue
132.
133.         matrix=np.int_(Matrix)
134.
135.     print ("The decision variable matrix is:")
136.
137.     print(matrix)

```

Transportation & Assignment



NorthWest Corner Method

Assignment Problem

Further Reference in Transportation & Assignment:

| | |
|---------------------------|---|
| NorthWest Corner Method - | NorthWest Corner Method from Introduction to Operations Research by Lieberman.pdf |
| Assignment problem - | Assignment problem from OR topcu.pdf |

Python Project on Optimization 1: Transportation Network for distributing products from source to destination³

Problem Description

A company has two warehouses from which it distributes products to five carefully chosen distribution centers. The company would like to have an interactive computer program which they can run week by week to tell them which warehouse should supply which distribution center so as to minimize the costs of the whole operation. For example, suppose that at the start of a given week the company has 2050 cases at warehouse A, and 8010 cases at warehouse B, and that the distribution centers or customer points require 1000, 1800, 4000, 500, and 1350 cases respectively. Which warehouse should supply which customer point?

Formulation

For transportation problems, using a graphical representation of the problem is often helpful during formulation. Here is a graphical representation of The Product Distribution Problem.

³ Source: https://www.coin-or.org/PuLP/CaseStudies/a_transportation_problem.html

```
# -*- coding: utf-8 -*-
"""
The Product Distribution Problem for the PuLP Modeller

Original Authors: Antony Phillips, Dr Stuart Mitchell 2007
Adopted by: Tanmoy Das, 2018
Source code: https://github.com/openstack/deb-python-
pulp/edit/master/examples/BeerDistributionProblem_resolve.py
https://www.coin-or.org/PuLP/CaseStudies/a_transportation_problem.html
https://github.com/tanmoyie/Operations-Research/tree/master/Transportation

"""

# Import PuLP modeler functions
from pulp import *

# Creates a list of all the supply nodes
Warehouses = ["A", "B"]

# Creates a dictionary for the number of units of supply for each supply node
supply = {"A": 2050,
          "B": 8010}

# Creates a list of all demand nodes
CustomerPoint = ["1", "2", "3", "4", "5"]

# Creates a dictionary for the number of units of demand for each demand node
demand = {"1":1000,
          "2":1800,
          "3":4000,
          "4":500,
          "5":1350,}

# Creates a list of costs of each transportation path
costs = [ #CustomerPoint
          #1 2 3 4 5
          [2,4,5,2,1],#A Warehouses
          [3,1,3,2,3] #B
        ]

# The cost data is made into a dictionary
costs = makeDict([Warehouses,CustomerPoint],costs,0)

# Creates the 'prob' variable to contain the problem data
prob = LpProblem("Product Distribution Problem",LpMinimize)

# Creates a list of tuples containing all the possible routes for transport
Routes = [(w,b) for w in Warehouses for b in CustomerPoint]
```



```

# A dictionary called 'Vars' is created to contain the referenced variables(the routes)
vars = LpVariable.dicts("Route", (Warehouses, CustomerPoint), 0, None, LpInteger)

# The objective function is added to 'prob' first
prob += lpSum([vars[w][b]*costs[w][b] for (w,b) in Routes]), "Sum_of_Transporting_Costs"

# The supply maximum constraints are added to prob for each supply node (warehouse)
for w in Warehouses:
    prob += lpSum([vars[w][b] for b in CustomerPoint]) <= supply[w],
    "Sum_of_Products_out_of_Warehouse_%s"%w

# The demand minimum constraints are added to prob for each demand node (customer)
# These constraints are stored for resolve later
customer_demand_constraint = {}
for b in CustomerPoint:
    constraint = lpSum([vars[w][b] for w in Warehouses]) >= demand[b]
    prob += constraint, "Sum_of_Products_into_customer_%s"%b
    customer_demand_constraint[b] = constraint

# The problem data is written to an .lp file
prob.writeLP("ProductDistributionProblem.lp")

for demand in range(500, 601, 10):
    # reoptimise the problem by increasing demand at customer '1'
    # note the constant is stored as the LHS constant not the RHS of the constraint
    customer_demand_constraint['1'].constant = - demand

# The problem is solved using PuLP's choice of Solver
prob.solve()

# The status of the solution is printed to the screen
print("Status:", LpStatus[prob.status])

# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)

# The optimised objective function value is printed to the screen
print("Total Cost of Transportation = ", value(prob.objective))

```

Table 1: Output

```

runfile('G:/Github Tanmoy Das/Operations-
Research/Transportation/transportation_problem_PuLP_example_of_product_distribution_from_
warehouse_to_customer.py', wdir='G:/Github Tanmoy Das/Operations-
Research/Transportation')
Status: Optimal
Route_A_1 = 500.0

```

Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 200.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 300.0
Route_B_5 = 0.0
Total Cost of Transportation = 17150.0

Status: Optimal
Route_A_1 = 510.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 190.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 310.0
Route_B_5 = 0.0
Total Cost of Transportation = 17170.0

Status: Optimal
Route_A_1 = 520.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 180.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 320.0
Route_B_5 = 0.0
Total Cost of Transportation = 17190.0

Status: Optimal
Route_A_1 = 530.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 170.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 330.0
Route_B_5 = 0.0
Total Cost of Transportation = 17210.0

Status: Optimal
Route_A_1 = 540.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 160.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 340.0
Route_B_5 = 0.0
Total Cost of Transportation = 17230.0

Status: Optimal
Route_A_1 = 550.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 150.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 350.0
Route_B_5 = 0.0
Total Cost of Transportation = 17250.0

Status: Optimal
Route_A_1 = 560.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 140.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 360.0
Route_B_5 = 0.0
Total Cost of Transportation = 17270.0

Status: Optimal
Route_A_1 = 570.0
Route_A_2 = 0.0
Route_A_3 = 0.0
Route_A_4 = 130.0
Route_A_5 = 1350.0
Route_B_1 = 0.0
Route_B_2 = 1800.0
Route_B_3 = 4000.0
Route_B_4 = 370.0
Route_B_5 = 0.0

Total Cost of Transportation = 17290.0

Status: Optimal

Route_A_1 = 580.0

Route_A_2 = 0.0

Route_A_3 = 0.0

Route_A_4 = 120.0

Route_A_5 = 1350.0

Route_B_1 = 0.0

Route_B_2 = 1800.0

Route_B_3 = 4000.0

Route_B_4 = 380.0

Route_B_5 = 0.0

Total Cost of Transportation = 17310.0

Status: Optimal

Route_A_1 = 590.0

Route_A_2 = 0.0

Route_A_3 = 0.0

Route_A_4 = 110.0

Route_A_5 = 1350.0

Route_B_1 = 0.0

Route_B_2 = 1800.0

Route_B_3 = 4000.0

Route_B_4 = 390.0

Route_B_5 = 0.0

Total Cost of Transportation = 17330.0

Status: Optimal

Route_A_1 = 600.0

Route_A_2 = 0.0

Route_A_3 = 0.0

Route_A_4 = 100.0

Route_A_5 = 1350.0

Route_B_1 = 0.0

Route_B_2 = 1800.0

Route_B_3 = 4000.0

Route_B_4 = 400.0

Route_B_5 = 0.0

Total Cost of Transportation = 17350.0

Python Code of Optimization Project 4: Travelling Salesman Problem

1. """
2. Running Code: <https://www.kaggle.com/tanmoyie/traveling-salesman-problem>
3. Source code: <https://developers.google.com/optimization/routing/tsp>
4. Google Map: <https://drive.google.com/open?id=18k6uA3KdLJGc2qQGUEXbtfG93KJj8Lty&usp=sharing>
5. Detail video: <https://youtu.be/e2lHyMl1IYY>

```

6. """
7. from ortools.constraint_solver import pywrapcp
8. from ortools.constraint_solver import routing_enums_pb2
9.
10. # Distance callback
11. class CreateDistanceCallback(object):
12.     """Create callback to calculate distances between points."""
13.     def __init__(self):
14.         """Array of distances between points."""
15.
16.         self.matrix = [
17.             [ 0, 290, 250, 230, 190, 334, 365, 40], # Dhaka
18.             [290, 0, 337, 453, 396, 560, 581, 244], # Syhlet
19.             [250, 337, 0, 495, 396, 540, 120, 240], # Chittagonj
20.             [230, 453, 495, 0, 360, 150, 595, 242], # Rajshahi
21.             [190, 396, 396, 360, 0, 356, 496, 253], # Jossore
22.             [334, 560, 540, 150, 356, 0, 674, 275], # Dinajpur
23.             [365, 581, 120, 595, 496, 674, 0, 397], # Coxsbazar
24.             [40, 244, 240, 242, 253, 275, 397, 0]] # Narsingdi
25. # distance between Dhaka to Syhlet is 290kms and so on
26.     def Distance(self, from_node, to_node):
27.         return int(self.matrix[from_node][to_node])
28. def main():
29.     # The order of the cities in the array is the following:
30.     # Cities
31.     city_names = ["Dhaka", "Syhlet", "Chittagonj", "Rajshahi", "Jossore", "Dinajpur", "Coxsbazar",
32.
33.         "Narsingdi"]
34.     tsp_size = len(city_names)
35.     num_routes = 1 # The number of routes, which is 1 in the TSP.
36.     # Nodes are indexed from 0 to tsp_size - 1. The depot is the starting node of the route.
37.     depot = 0
38.     # Create routing model
39.     if tsp_size > 0:
40.         routing = pywrapcp.RoutingModel(tsp_size, num_routes, depot)
41.         search_parameters = pywrapcp.RoutingModel.DefaultSearchParameters()
42.
43.         # Create the distance callback, which takes two arguments (the from and to node indices)
44.         # and returns the distance between these nodes.
45.         dist_between_nodes = CreateDistanceCallback()
46.         dist_callback = dist_between_nodes.Distance
47.         routing.SetArcCostEvaluatorOfAllVehicles(dist_callback)
48.         # Solve, returns a solution if any.
49.         assignment = routing.SolveWithParameters(search_parameters)
50.         if assignment:
51.             # Solution cost.
52.             print ("Total distance: " + str(assignment.ObjectiveValue()) + " miles\n")
53.             # Inspect solution.
54.             # Only one route here; otherwise iterate from 0 to routing.vehicles() - 1

```

```

55. route_number = 0
56. index = routing.Start(route_number) # Index of the variable for the starting node.
57. route = ""
58. while not routing.IsEnd(index):
59.     # Convert variable indices to node indices in the displayed route.
60.     route += str(city_names[routing.IndexToNode(index)]) + ' -> '
61.     index = assignment.Value(routing.NextVar(index))
62.     route += str(city_names[routing.IndexToNode(index)])
63.     print ("Route:\n\n" + route)
64. else:
65.     print ('No solution found.')
66. else:
67.     print ('Specify an instance greater than 0.')
68.
69. if __name__ == '__main__':
70.     main()

```

Optimization in Machine Learning/ Data Science



Linear Regression

Robust Regression

Support Vector Machine

Further Reference in Machine Learning & Optimization:

Linear Regression - Follow Class Lecture

Robust Regression - Follow Class Lecture

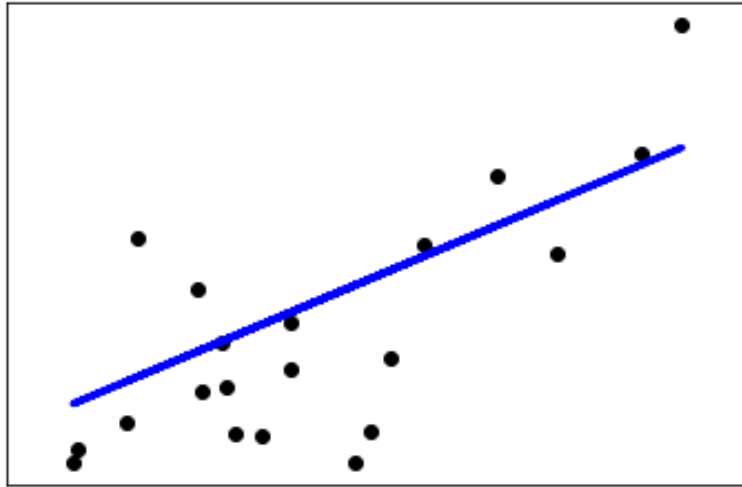
Support Vector Machine - Follow Class Lecture

Python Projects on Machine Learning Optimization

Python Code of Optimization Project 5: Linear Regression (plot_ols.py)

```
1. #!/usr/bin/python
2. # -*- coding: utf-8 -*-
3.
4. """
5. =====
6. Linear Regression Example
7. =====
8. This example uses the only the first feature of the `diabetes` dataset, in
9. order to illustrate a two-dimensional plot of this regression technique. The
10. straight line can be seen in the plot, showing how linear regression attempts
11. to draw a straight line that will best minimize the residual sum of squares
12. between the observed responses in the dataset, and the responses predicted by
13. the linear approximation.
14.
15. The coefficients, the residual sum of squares and the variance score are also
16. calculated.
17.
18. """
19. print(__doc__)
20. # Code source: Jaques Grobler
21.
22. import matplotlib.pyplot as plt
23. import numpy as np
24. from sklearn import datasets, linear_model
25. from sklearn.metrics import mean_squared_error, r2_score
26.
27. # Load the diabetes dataset
28. diabetes = datasets.load_diabetes()
29. # Use only one feature
30. diabetes_X = diabetes.data[:, np.newaxis, 2]
31. # Split the data into training/testing sets
32. diabetes_X_train = diabetes_X[:-20]
33. diabetes_X_test = diabetes_X[-20:]
34. # Split the targets into training/testing sets
35. diabetes_y_train = diabetes.target[:-20]
36. diabetes_y_test = diabetes.target[-20:]
37.
38. # Create linear regression object
39. regr = linear_model.LinearRegression()
40. # Train the model using the training sets
41. regr.fit(diabetes_X_train, diabetes_y_train)
42. # Make predictions using the testing set
43. diabetes_y_pred = regr.predict(diabetes_X_test)
44.
45. # The coefficients
46. print('Coefficients: \n', regr.coef_)
47. # The mean squared error
48. print("Mean squared error: %.2f"
49.       % mean_squared_error(diabetes_y_test, diabetes_y_pred))
50. # Explained variance score: 1 is perfect prediction
51. print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))
52.
53. # Plot outputs
54. plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
55. plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
56.
57. plt.xticks(())
58. plt.yticks()
```

```
59. plt.show()
```



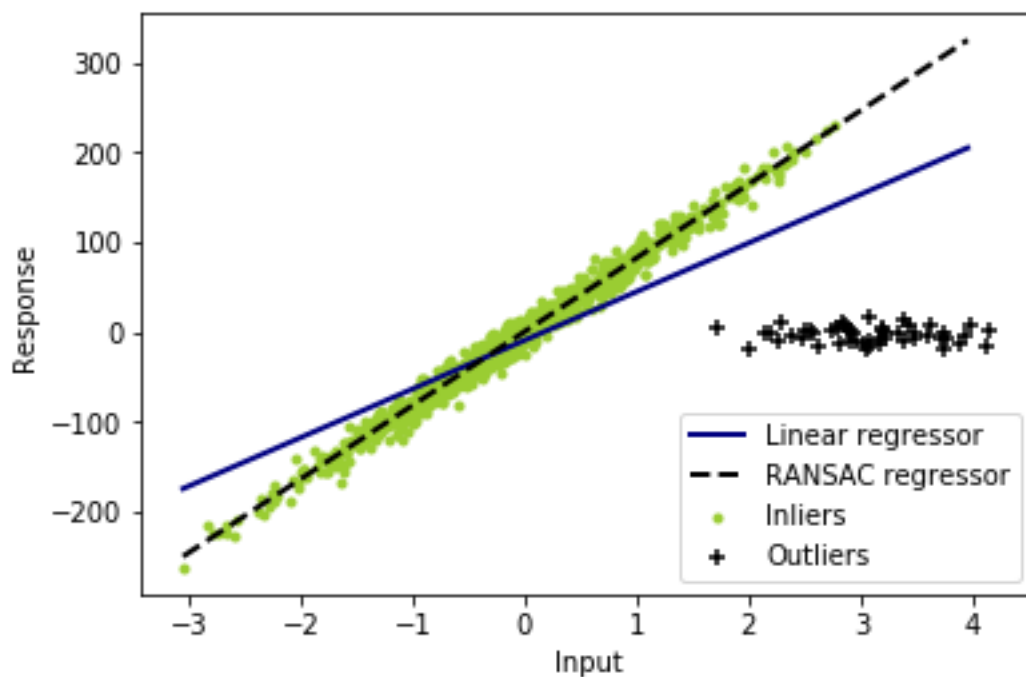
Python Code of Optimization Project 6: Robust Regression compared to Linear Regression (plot_ransac.py)

```
1. """
2. Robust linear model estimation using RANSAC
3. In this example we see how to robustly fit a linear model to faulty data using
4. the RANSAC algorithm.
5. Source Code: http://scikit-learn.org/stable/auto\_examples/linear\_model/plot\_ransac.html
6. """
7. import numpy as np
8. from matplotlib import pyplot as plt
9. from sklearn import linear_model, datasets
10.
11. n_samples = 1000
12. n_outliers = 50
13. X, y, coef = datasets.make_regression(n_samples=n_samples, n_features=1,
14.                                     n_informative=1, noise=10,
15.                                     coef=True, random_state=0)
16.
17. # Add outlier data
18. np.random.seed(0)
19. X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1))
20. y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)
21.
22. # Fit line using all data
23. lr = linear_model.LinearRegression()
24. lr.fit(X, y)
25.
26. # Robustly fit linear model with RANSAC algorithm
27. ransac = linear_model.RANSACRegressor()
28. ransac.fit(X, y)
29. inlier_mask = ransac.inlier_mask_
30. outlier_mask = np.logical_not(inlier_mask)
31.
32. # Predict data of estimated models
33. line_X = np.arange(X.min(), X.max())[:, np.newaxis]
34. line_y = lr.predict(line_X)
35. line_y_ransac = ransac.predict(line_X)
36.
```

```

37. # Compare estimated coefficients
38. print("Estimated coefficients (true, linear regression, RANSAC):")
39. print(coef, lr.coef_, ransac.estimator_.coef_)
40. lw = 2
41. plt.scatter(X[inlier_mask], y[inlier_mask], color='yellowgreen', marker='.',
42.             label='Inliers')
43. plt.scatter(X[outlier_mask], y[outlier_mask], color='black', marker='+',
44.             label='Outliers')
45. plt.plot(line_X, line_y, color='navy', linewidth=lw, label='Linear regressor')
46. plt.plot(line_X, line_y_ransac, color='black', linestyle='--', linewidth=lw,
47.          label='RANSAC regressor')
48. plt.legend(loc='lower right')
49. plt.xlabel("Input")
50. plt.ylabel("Response")
51. plt.show()

```



Network Optimization



Min Cost

Max Flow

Minimum Spanning Tree

Further Reference in Network Optimization:

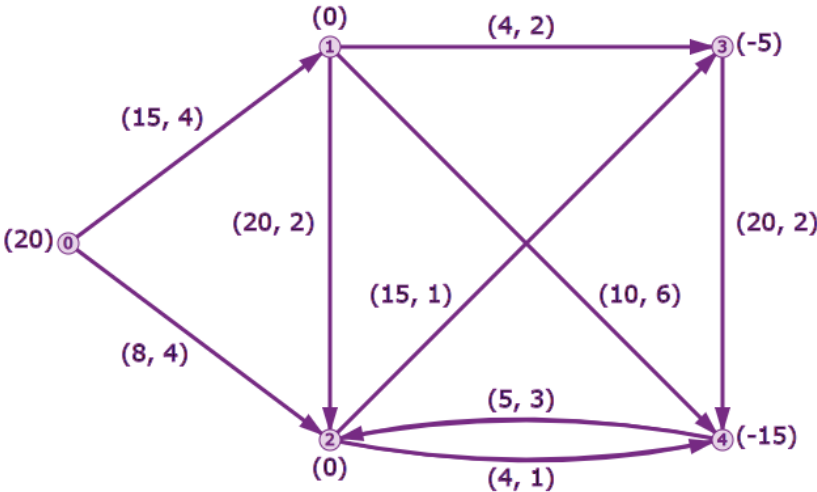
| | |
|----------------------|---|
| Min Cost, Max Flow - | Network Optimization from Introduction to OR - deterministic model Juraj Stacho.pdf |
| Shortest Path, MST - | Network Optimization from Introduction to Operations Research by Lieberman.pdf |

Python Code of Optimization Project 7: Min Cost Flow problem

```

1. """
2. Running kernel: https://www.kaggle.com/tanmoyie/min-cost-flow-google-developer
3. Source: https://developers.google.com/optimization/flow/mincostflow
4.
5. """

```



```

6.
7. # """From Bradley, Hax, and Magnanti, 'Applied Mathematical Programming', figure 8.1."""
8.
9. from __future__ import print_function
10. from ortools.graph import pywrapgraph
11.
12. def main():
13.     """MinCostFlow simple interface example."""
14.
15.     # Define four parallel arrays: start_nodes, end_nodes, capacities, and unit costs
16.     # between each pair. For instance, the arc from node 0 to node 1 has a
17.     # capacity of 15 and a unit cost of 4.
18.
19.     start_nodes = [0, 0, 1, 1, 1, 2, 2, 3, 4]
20.     end_nodes = [1, 2, 2, 3, 4, 3, 4, 4, 2]
21.     capacities = [15, 8, 20, 4, 10, 15, 5, 20, 4]
22.     unit_costs = [4, 4, 2, 2, 6, 1, 3, 2, 3]
23.
24.     # Define an array of supplies at each node.
25.
26.     supplies = [20, 0, 0, -5, -15]
27.
28.     # Instantiate a SimpleMinCostFlow solver.
29.     min_cost_flow = pywrapgraph.SimpleMinCostFlow()
30.

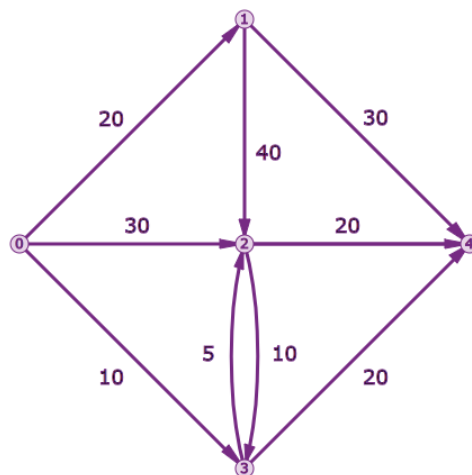
```

```

31. # Add each arc.
32. for i in range(0, len(start_nodes)):
33.     min_cost_flow.AddArcWithCapacityAndUnitCost(start_nodes[i], end_nodes[i],
34.                                                  capacities[i], unit_costs[i])
35.
36. # Add node supplies.
37.
38. for i in range(0, len(supplies)):
39.     min_cost_flow.SetNodeSupply(i, supplies[i])
40.
41.
42. # Find the minimum cost flow between node 0 and node 4.
43. if min_cost_flow.Solve() == min_cost_flow.OPTIMAL:
44.     print('Minimum cost:', min_cost_flow.OptimalCost())
45.     print("")
46.     print(' Arc  Flow / Capacity  Cost')
47.     for i in range(min_cost_flow.NumArcs()):
48.         cost = min_cost_flow.Flow(i) * min_cost_flow.UnitCost(i)
49.         print('%1s -> %1s  %3s / %3s    %3s' % (
50.             min_cost_flow.Tail(i),
51.             min_cost_flow.Head(i),
52.             min_cost_flow.Flow(i),
53.             min_cost_flow.Capacity(i),
54.             cost))
55. else:
56.     print('There was an issue with the min cost flow input.')
57.
58. if __name__ == '__main__':
59.     main()

```

Python Code of Optimization Project 8: Max Flow



""From Taha 'Introduction to Operations Research', example 6.4-2.""

```

# https://developers.google.com/optimization/flow/maxflow
from __future__ import print_function
from ortools.graph import pywrapgraph

def main():
    """MaxFlow simple interface example."""

    # Define three parallel arrays: start_nodes, end_nodes, and the capacities
    # between each pair. For instance, the arc from node 0 to node 1 has a
    # capacity of 20.

    start_nodes = [0, 0, 0, 1, 1, 2, 2, 3, 3]
    end_nodes = [1, 2, 3, 2, 4, 3, 4, 2, 4]
    capacities = [20, 30, 10, 40, 30, 10, 20, 5, 20]

    # Instantiate a SimpleMaxFlow solver.
    max_flow = pywrapgraph.SimpleMaxFlow()
    # Add each arc.
    for i in range(0, len(start_nodes)):
        max_flow.AddArcWithCapacity(start_nodes[i], end_nodes[i], capacities[i])

    # Find the maximum flow between node 0 and node 4.
    if max_flow.Solve(0, 4) == max_flow.OPTIMAL:
        print('Max flow:', max_flow.OptimalFlow())
        print('')
        print(' Arc  Flow / Capacity')
        for i in range(max_flow.NumArcs()):
            print('%1s -> %1s  %3s / %3s' % (
                max_flow.Tail(i),
                max_flow.Head(i),
                max_flow.Flow(i),
                max_flow.Capacity(i)))
        print('Source side min-cut:', max_flow.GetSourceSideMinCut())
        print('Sink side min-cut:', max_flow.GetSinkSideMinCut())
    else:
        print('There was an issue with the max flow input.')

if __name__ == '__main__':
    main()

```

Python Code of Optimization Project 9: Airlines Network Optimization⁴

```

1.  # -*- coding: utf-8 -*-
2.  """
3.  Created on Tue Jun 5 20:09:14 2018

```

⁴ Formatting & Color help from <http://www.planetb.ca/syntax-highlight-word>


```

4.
5. @adopted by: Tanmoy Das
6. Earlier version: https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/
7.
8. """
9.
10. # import the libraries
11. import numpy as np # linear algebra
12. import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
13. import os
14.
15. # load the dataset
16. data = pd.read_csv('airlines_network_optimization.csv') # download the csv file in your local
    directory and play with it.
17.
18. # data.shape
19. # converting sched_dep_time to 'std' - Scheduled time of departure
20. data['std'] = data.sched_dep_time.astype(str).str.replace('\d{2}$', '') + ':' + data.sched_dep_ti
    me.astype(str).str.extract('\d{2}$', expand=False) + ':00'
21. # converting sched_arr_time to 'sta' - Scheduled time of arrival
22. data['sta'] = data.sched_arr_time.astype(str).str.replace('\d{2}$', '') + ':' + data.sched_arr_ti
    me.astype(str).str.extract('\d{2}$', expand=False) + ':00'
23. # converting dep_time to 'atd' - Actual time of departure
24. data['atd'] = data.dep_time.fillna(0).astype(np.int64).astype(str).str.replace('\d{2}$', '') + ':'
    + data.dep_time.fillna(0).astype(np.int64).astype(str).str.extract('\d{2}$', expand=False) + '
    :00'
25. # converting arr_time to 'ata' - Actual time of arrival
26. data['ata'] = data.arr_time.fillna(0).astype(np.int64).astype(str).str.replace('\d{2}$', '') + ':'
    + data.arr_time.fillna(0).astype(np.int64).astype(str).str.extract('\d{2}$', expand=False) + '
    00'
27. data['date'] = pd.to_datetime(data[['year', 'month', 'day']])
28. # finally we drop the columns we don't need
29. data = data.drop(columns = ['year', 'month', 'day'])
30.
31. import networkx as nx
32. FG = nx.from_pandas_edgelist(data, source='origin', target='dest', edge_attr=True,)
33. # detail documentation of networkx https://networkx.github.io/documentation/networkx-
    1.7/reference/generated/networkx.drawing.nx\_pylab.draw\_networkx.html
34. FG.nodes()
35. FG.edges()
36. nx.draw_networkx(FG, with_labels=True, node_size=600, node_color='y') # Quick view of the
    Graph. As expected we see 3 very busy airports
37.
38. nx.algorithms.degree_centrality(FG) # Notice the 3 airports from which all of our 100 rows of
    data originates
39. nx.density(FG) # Average edge density of the Graphs
40. nx.average_shortest_path_length(FG) # Average shortest path length for ALL paths in the Gra
    ph

```

```

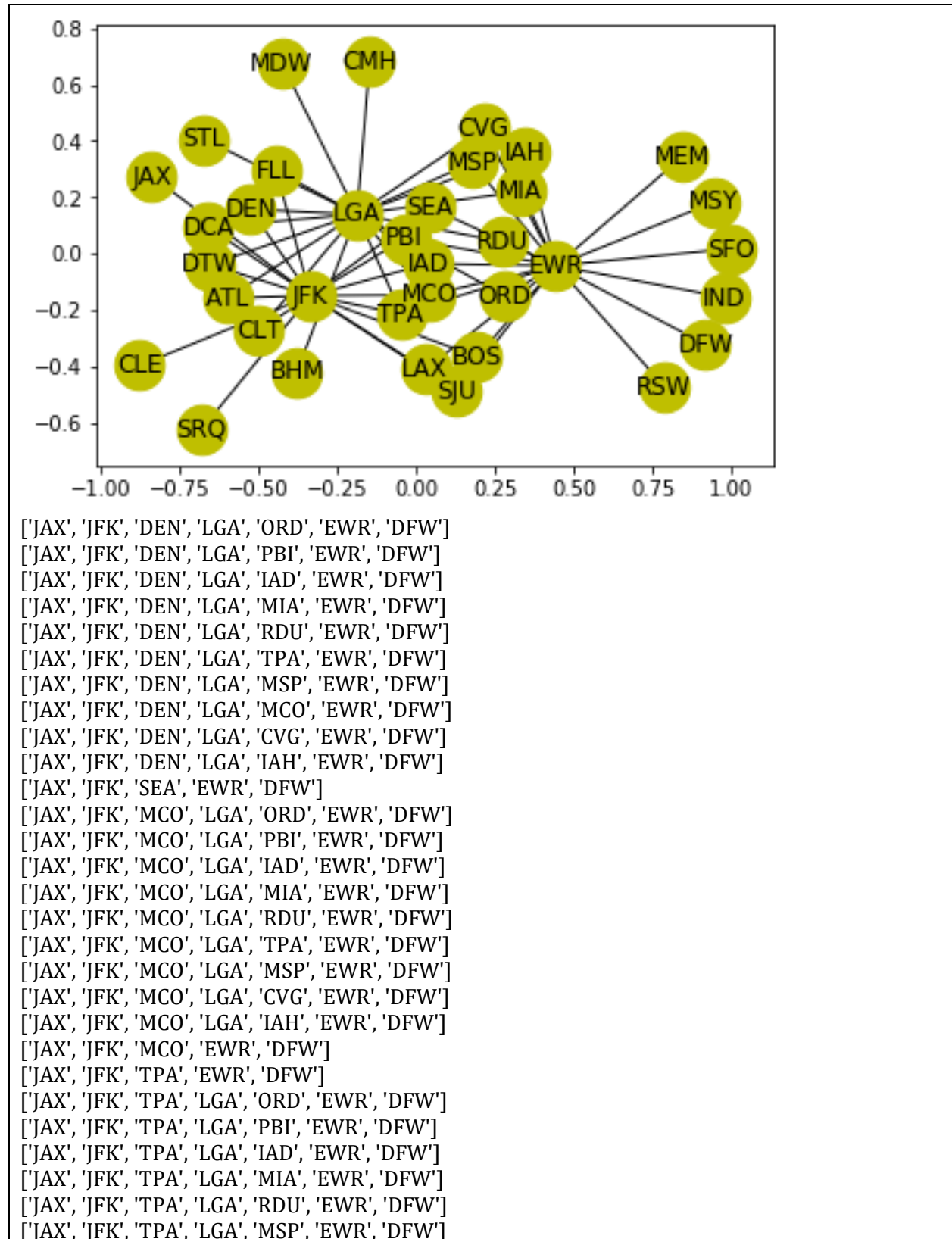
41. nx.average_degree_connectivity(FG) # For a node of degree k - What is the average of its neighbours' degree?
42.
43. # Let us find all the paths available
44. for path in nx.all_simple_paths(FG, source='JAX', target='DFW'):
45.     print(path)
46. # Let us find the dijkstra path from JAX to DFW.
47. # You can read more in-
    depth on how dijkstra works from this resource - https://courses.csail.mit.edu/6.006/fall11/lectures/lecture16.pdf
48. dijkpath = nx.dijkstra_path(FG, source='JAX', target='DFW')
49. dijkpath
50. # Let us try to find the dijkstra path weighted by airtime (approximate case)
51. shortpath = nx.dijkstra_path(FG, source='JAX', target='DFW', weight='air_time')
52. shortpath

```

Airlines_network_optimization.csv (first 10 records)

| year | month | day | dep_time | sched_dep_time | dep_delay | arr_time | sched_arr_time | arr_delay | carrier | flight | tailnum | origin | dest | air_time | distance |
|------|-------|-----|----------|----------------|-----------|----------|----------------|-----------|---------|--------|---------|--------|------|----------|----------|
| 2013 | 2 | 26 | 1807 | 1630 | 97 | 1956 | 1837 | 79 | EV | 4411 | N13566 | EW R | MEM | 144 | 946 |
| 2013 | 8 | 17 | 1459 | 1445 | 14 | 1801 | 1747 | 14 | B6 | 1171 | N661JB | LG A | FL L | 147 | 1076 |
| 2013 | 2 | 13 | 1812 | 1815 | -3 | 2055 | 2125 | -30 | AS | 7 | N403AS | EW R | SEA | 315 | 2402 |
| 2013 | 4 | 11 | 2122 | 2115 | 7 | 2339 | 2353 | -14 | B6 | 97 | N656JB | JFK | DE N | 221 | 1626 |
| 2013 | 8 | 5 | 1832 | 1835 | -3 | 2145 | 2155 | -10 | AA | 269 | N3EYAA | JFK | SEA | 358 | 2422 |
| 2013 | 6 | 30 | 1500 | 1505 | -5 | 1751 | 1650 | 61 | UA | 685 | N424UA | LG A | ORD | 116 | 733 |
| 2013 | 2 | 14 | 1442 | 1445 | -3 | 1833 | 1747 | 46 | UA | 346 | N446UA | EW R | MIA | 200 | 1085 |
| 2013 | 7 | 25 | 752 | 755 | -3 | 1037 | 1057 | -20 | DL | 2395 | N909DL | LG A | PBI | 140 | 1035 |
| 2013 | 7 | 10 | 557 | 600 | -3 | 725 | 715 | 10 | MQ | 3267 | N542MQ | EW R | ORD | 113 | 719 |

Table 2: Output of Airlines Network Optimization



```

['JAX', 'JFK', 'TPA', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'SJU', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'ATL', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'DCA', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'DTW', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'LAX', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'FLL', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'RDU', 'EWR', 'DFW']

```

```

['JAX', 'JFK', 'CLT', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'CLT', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MSP', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'MCO', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'CVG', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'LGA', 'IAH', 'EWR', 'DFW']
['JAX', 'JFK', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'BOS', 'EWR', 'DFW']

```

Integer Programming



Further Reference in Integer Programming:

Integer Programming - Integer Programming from OPERATIONS RESEARCH by R.
PANNEERSELVAM 2nd edition (selected pages).pdf