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 Model[1]).

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-fevmrx2V3U13hPDDlUSBbi7

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/UC0yUOupBXybIfQ2x7uM6kzg

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 Stacho[2] (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

Python projects in Optimization

Python Codes in Optimization

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:

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

$$subject\ to$$

$$Ax \leq b$$

$$x \geq 0.$$

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

Math Problem

Graphical Solution

<u>Further Reference in Intro to OR:</u>

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

Tabular

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 Project on Optimization 1: Revised Simplex

Linear Program:

$$maximize\ Z = 3X_1 + 5X_2$$

$$\text{Subject to,} \quad x_1 \leq 4$$

$$2X_2 \leq 12$$

$$3X_1 + 2X_2 \leq 18$$

$$X_1, X_2 \geq 0$$

Output from following Python code:

```
Optimization terminated successfully.
         Current function value: -36.000000
         Iterations: 2
     fun: -36.0
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([2., 0., 0.])
  status: 0
 success: True
       x: array([2., 6.])
```

Python Code of Optimization Project 1: Revised Simplex (Simplex_in_scipy.py)

```python
"""
Related YouTube Video: https://youtu.be/e21HyMl1IYY
"""
import numpy as np
import scipy as sp

c = [-3, -5]
A = [[1, 0], [0, 2], [3, 2]]
b = [4, 12, 18]
x0_bounds = (0, None)
x1_bounds = (0, None)

from scipy.optimize import linprog
# Solve the problem by Simplex method in Optimization
res = linprog(c, A_ub=A, b_ub=b,  bounds=(x0_bounds, x1_bounds), method='simplex', options={"disp": True})
print(res)
```

Python Project on Optimization 2: Shelf Space Optimization in super shop

In a store, a product's position in store can greatly affect its performance. Having the right space allocation for products and categories plays a critical role in its retail success. From retailers' perspective, given the value of shelf space positions, it is very critical to ensure that retail space is working for value maximization for the store.

The shelves near the POS offer maximum visibility to the customers and help the stores reap in those extra few dollars for items which were not even in the shoppers list. Marketing the right merchandise, at the right place, at the right time, in the right quantities is key to retail revenues and profitability. This has led to a war between brands to occupy the best possible space in a store. On the other hand, the stores also have to optimize their overall profitability considering the sales of all merchandise.

Dataset:

Shelf	Unilever 1	Unilever 2	Unilever 3	Godrej 4	Godrej 5	Godrej 6	Dabur 7	Dabur 8
1	7	818	650	848	630	648	842	842
2	691	849	615	700	653	598	563	563
3	427	413	349	347	407	237	465	465
4	345	153	282	301	477	432	313	313
5	464	470	126	392	534	312	326	326
6	281	144	283	200	168	107	148	148
7	238	500	291	434	465	488	544	544
8	138	86	119	149	92	136	119	119
9	127	124	141	54	130	140	75	75
10	70	141	78	106	69	51	58	58

Python Code of Optimization Project 2: Shelf Space Optimization

```
# Source: https://www.analyticsvidhya.com/blog/2016/09/a-beginners-guide-to-shelf-space-optimization-using-linear-programming/
# Run on Jupyter notebook; dataset: sales_lift.csv
#import all relevant libraries

import pandas as pd
import numpy as np
import math
from math import isnan

    from pulp import *
    from collections import Counter

    #from more_itertools import unique_everseen
    sales=pd.read_csv("sales_lift.csv",header=None) #input file
```

```python
    lift=sales.iloc[2:,1:]
    lift=np.array(lift)
    lift = lift.astype(np.int) # read the lifts from csv
    brands=sales.iloc[0:1,:]
    brands=np.array(brands)
    brands=np.delete(brands,0)
    brands=brands.tolist()  # read the brands from csv
    ff=Counter(brands)
    all_brands=ff.items()

    # the racks and the shelfs available
    rack_shelf=[[1,1,2,3],[2,4,5,6],[3,7,8,9,10]]
    #define the optimization function
    prob=LpProblem("SO",LpMaximize)
    #define decision variables
    dec_var=LpVariable.matrix("dec_var",(range(len(lift)),range(len(lift[0]))),0,1,LpBinary)
    #Compute the sum product of decision variables and lifts
    prodt_matrix=[dec_var[i][j]*lift[i][j] for i in range(len(lift))
    for j in range(len(lift[0]))]
    #total lift which has to be maximized sum(prodt_matrix)
    #define the objective function
    prob+=lpSum(prodt_matrix)
    order=list(unique_everseen(brands))
    order_map = {}
    for pos, item in enumerate(order):
        order_map[item] = pos
    #brands in order as in input file
    brands_lift=sorted(all_brands, key=lambda x: order_map[x[0]])
    #DEFINE CONSTRAINTS
    #1) Each shelf can have only one product i.e. sum (each row)<=1

    for i in range(len(lift)):

        prob+=lpSum(dec_var[i])<=1
    # 2) Each product can be displayed only on a limited number of shelves i.e. Column constraints

    #Constraints are given as
    col_con=[1,0,0,2,2,3,1,1]
    dec_var=np.array(dec_var)
    col_data=[]

    for j in range(len(brands)):
       col_data.append(list(zip(*dec_var)[j]))
       prob+=lpSum(col_data[j])<=col_con[j]

    #write the problem
    prob.writeLP("SO.lp")
```

```
    #solve the problem
    prob.solve()
    print("The maximum Total lift obtained is:",value(prob.objective)) # print the output
    #print the decision variable output matrix
    Matrix=[[0 for X in range(len(lift[0]))] for y in range(len(lift))]
    for v in prob.variables():
        Matrix[int(v.name.split("_")[2])][int(v.name.split("_")[3])]=v.varValue
        matrix=np.int_(Matrix)
    print ("The decision variable matrix is:")
    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 3: Transportation Network for distributing products[3]

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

Python Code of Optimization Project 3: Product Distribution Problem for the PuLP Modeller (transportation_problem_PuLP_example_of_product_distribution_from_warehouse_to_customer.py)

```python
# -*- 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)
```

```python
    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))
```

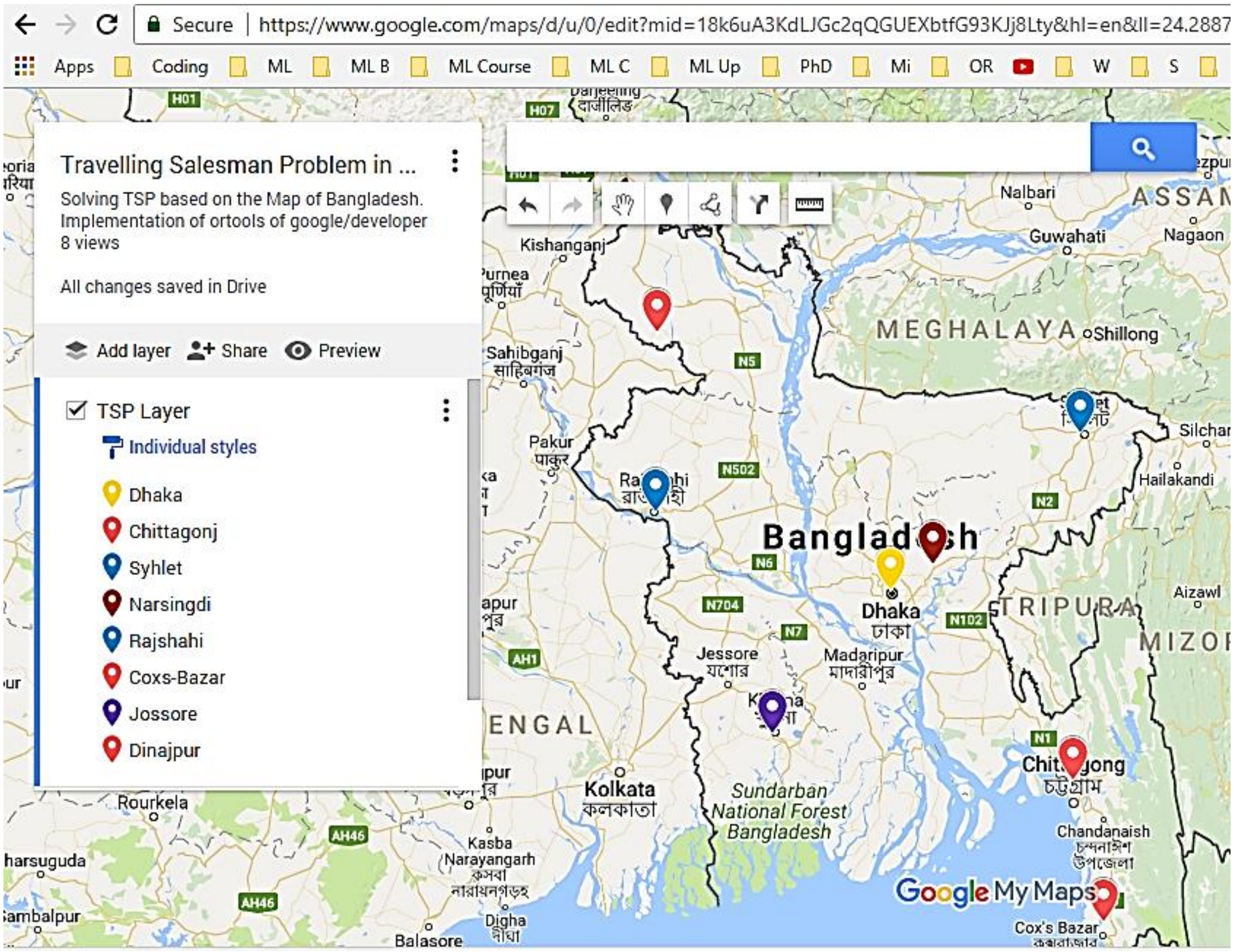
Table 1: Output

```
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
```

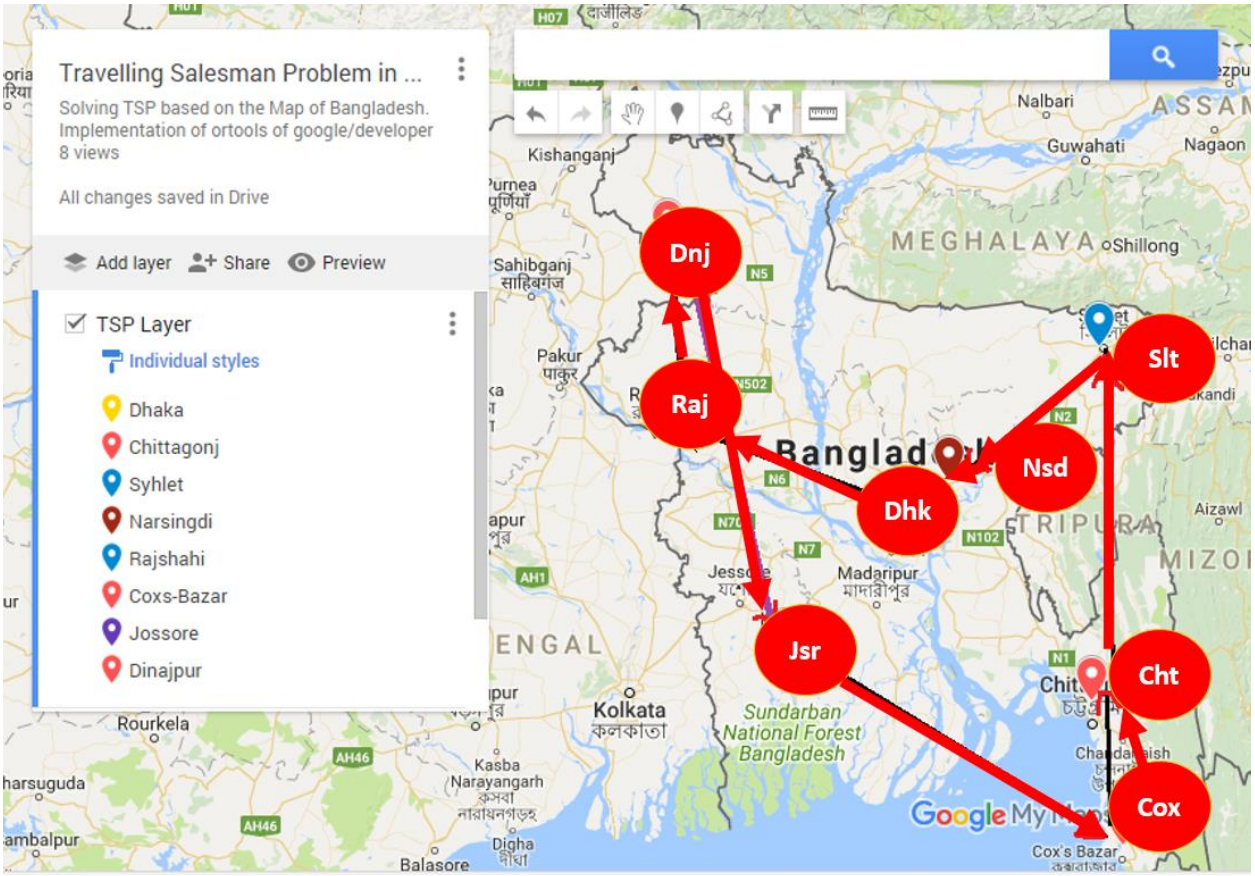
Python Project on Optimization 4: Travelling Salesman Problem



There are eight cities which are considered for this Travelling Salesman Problem. A tourist is planning to visit all these eight cities, starting from Dhaka. We need to find optimal path so that all the cities are covered with minimum possible distance. After applying following program, the output is as follow:

Total distance: 1973 miles

Route: Dhaka -> Rajshahi -> Dinajpur -> Jossore -> Coxsbazar -> Chittagonj -> Syhlet -> Narsingdi -> Dhaka



Python Code of Optimization Project 4: Travelling Salesman Problem (traveling_salesman_problem (finding_travelling_path_covering_all_points_with_min_total_distance))

```
"""
Running Code: https://www.kaggle.com/tanmoyie/traveling-salesman-problem
Source code: https://developers.google.com/optimization/routing/tsp
Google Map: https://drive.google.com/open?id=18k6uA3KdLJGc2qQGUEXbtfG93KJj8Lty&usp=sharing
Detail video: https://youtu.be/e2lHyMl1IYY
"""
from ortools.constraint_solver import pywrapcp
from ortools.constraint_solver import routing_enums_pb2

# Distance callback
class CreateDistanceCallback(object):
  """Create callback to calculate distances between points."""
  def __init__(self):
    """Array of distances between points."""

    self.matrix = [
    [  0, 290, 250,  230,  190,  334, 365,   40], # Dhaka
    [290,   0, 337,  453,  396,  560, 581,  244], # Syhlet
    [250, 337,   0,  495,  396,  540, 120,  240], # Chittagonj
    [230, 453, 495,    0,  360,  150, 595,  242], # Rajshahi
```

```
    [190, 396, 396,  360,   0,  356, 496,  253], # Jossore
    [334, 560, 540,  150,  356,   0, 674,  275], # Dinajpur
    [365, 581, 120,  595,  496,  674,   0,  397], # Coxsbazar
    [40,  244, 240,  242,  253,  275, 397,    0]] # Narsingdi
# distance between Dhaka to Sylhet is 290kms and so on
  def Distance(self, from_node, to_node):
    return int(self.matrix[from_node][to_node])
def main():
  # The order of the cities in the array is the following:   Cities
  city_names = ["Dhaka", "Syhlet", "Chittagonj", "Rajshahi", "Jossore", "Dinajpur", "Coxsbazar",
   "Narsingdi"]
  tsp_size = len(city_names)
  num_routes = 1    # The number of routes, which is 1 in the TSP.
  # Nodes are indexed from 0 to tsp_size - 1. The depot is the starting node of the route.
  depot = 0
  # Create routing model
  if tsp_size > 0:
    routing = pywrapcp.RoutingModel(tsp_size, num_routes, depot)
    search_parameters = pywrapcp.RoutingModel.DefaultSearchParameters()
    # Create the distance callback, which takes two arguments (the from and to node indices)
    # and returns the distance between these nodes.
    dist_between_nodes = CreateDistanceCallback()
    dist_callback = dist_between_nodes.Distance
    routing.SetArcCostEvaluatorOfAllVehicles(dist_callback)
    # Solve, returns a solution if any.
    assignment = routing.SolveWithParameters(search_parameters)
    if assignment:
      # Solution cost.
      print ("Total distance: " + str(assignment.ObjectiveValue()) + " miles\n")
      # Inspect solution.
      # Only one route here; otherwise iterate from 0 to routing.vehicles() - 1
      route_number = 0
      index = routing.Start(route_number) # Index of the variable for the starting node.
      route = ''
      while not routing.IsEnd(index):
        # Convert variable indices to node indices in the displayed route.
        route += str(city_names[routing.IndexToNode(index)]) + ' -> '
        index = assignment.Value(routing.NextVar(index))
      route += str(city_names[routing.IndexToNode(index)])
      print ("Route:\n\n" + route)
    else:
      print ('No solution found.')
  else:
    print ('Specify an instance greater than 0.')

if __name__ == '__main__':
  main()
```

Optimization in Machine Learning/ Data Science

Linear Regression

Robust Regression

Support Vector Machine

Given training vectors $x_i \in \mathbb{R}^p$, i=1,…, n, in two classes, and a vector $y \in \{1, -1\}^n$, SVC solves the following primal problem:

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$
$$\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i,$$
$$\zeta_i \geq 0, i = 1, ..., n$$

Its dual is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$
$$\text{subject to } y^T \alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1, ..., n$$

where e is the vector of all ones, $C > 0$ is the upper bound, Q is an n by n positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ is the kernel. Here training vectors are implicitly mapped into a higher (maybe infinite) dimensional space by the function ϕ.

The decision function is:

$$\operatorname{sgn}(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho)$$

<u>Further Reference in Machine Learning & Optimization:</u>

Linear Regression -	Follow Class Lecture
Robust Regression -	Follow Class Lecture
Support Vector Machine -	Follow Class Lecture

Python Project on Optimization 5: Linear Regression

The project thrives for finding estimated values (Ordinary Least Square method) by Linear Regression. The first few records of the x_test & y_test are given in the following table.

x_test	y_test
0.0778634	233
-0.0396181	91
0.011039	111
-0.0406959	152
-0.0342291	120
0.00564998	67
0.0886415	310
-0.0331513	94
-0.0568631	183
-0.0309956	66

Output from the following Python Programming:

Coefficients:

[938.23786125]

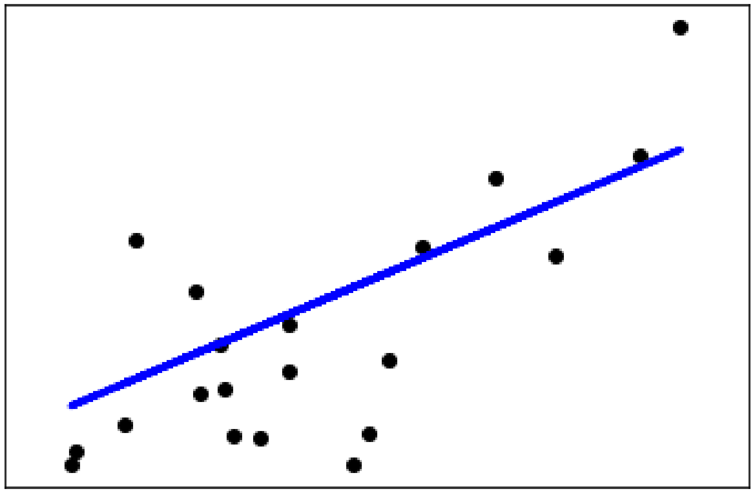
Mean squared error: 2548.07

Variance score: 0.47

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

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
=========================================================
Linear Regression Example
=========================================================
This example uses the only the first feature of the `diabetes` dataset, in
order to illustrate a two-dimensional plot of this regression technique. The
    straight line can be seen in the plot, showing how linear regression attempts
    to draw a straight line that will best minimize the residual sum of squares
    between the observed responses in the dataset, and the responses predicted by
    the linear approximation.

```

```python
The coefficients, the residual sum of squares and the variance score are also
calculated.

"""
print(__doc__)
# Code source: Jaques Grobler

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()
# Use only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())
plt.show()
```

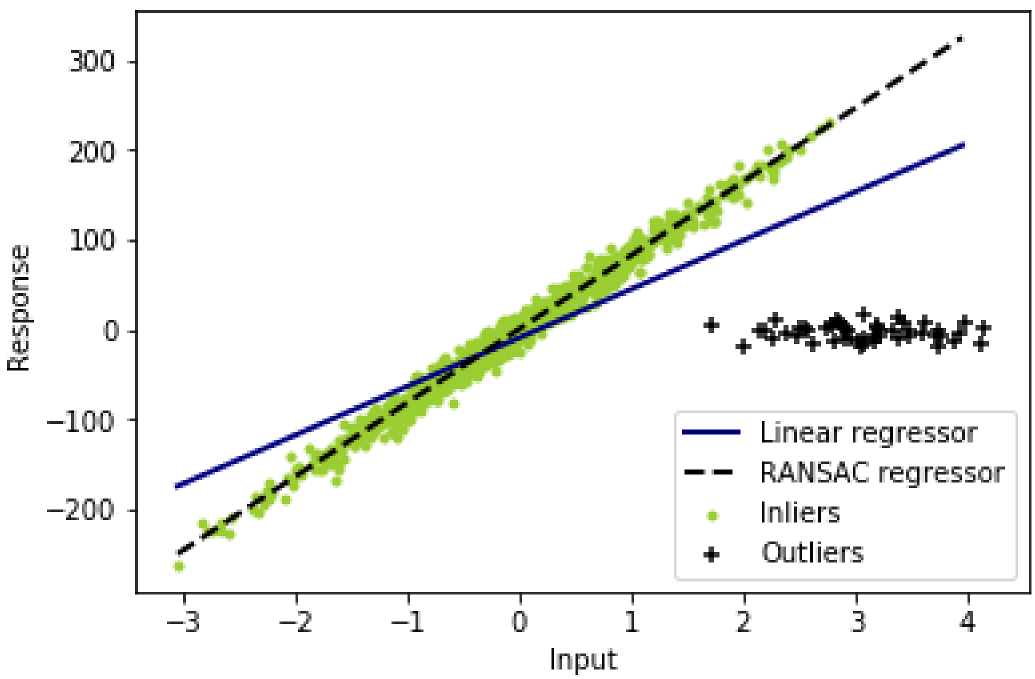


Python Project on Optimization 6: Robust regression

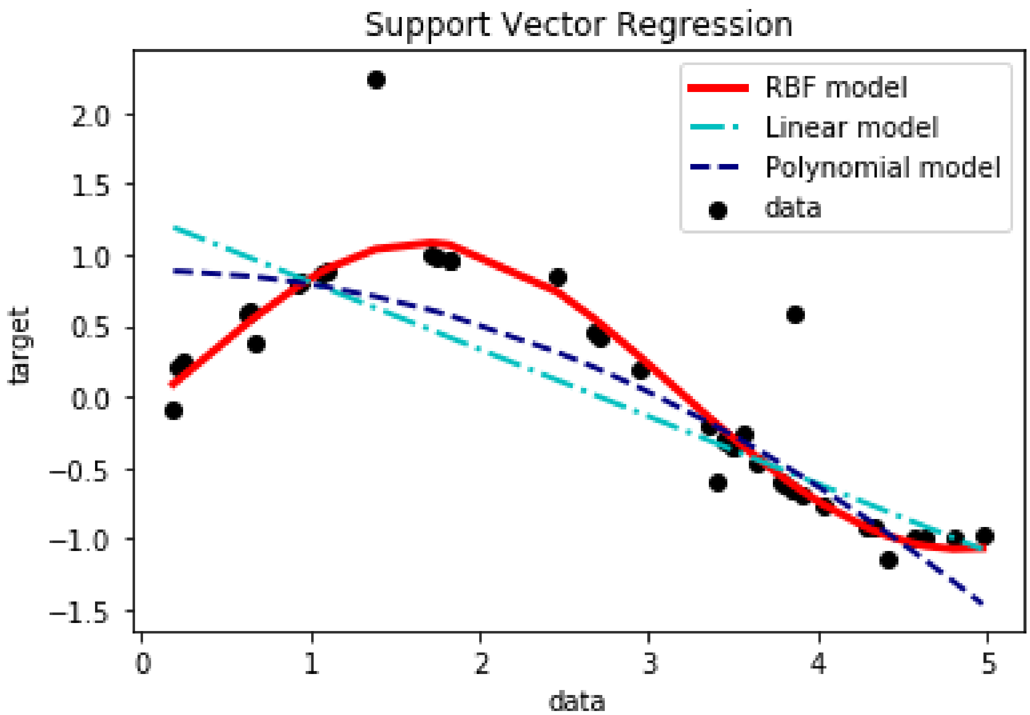
Python Code of Optimization Project 6: Robust Regression compared to Linear Regression (plot_ransac.py)

```
"""
Robust linear model estimation using RANSAC
In this example we see how to robustly fit a linear model to faulty data using
the RANSAC algorithm.
Source Code: http://scikit-learn.org/stable/auto_examples/linear_model/plot_ransac.html
"""
import numpy as np
from matplotlib import pyplot as plt
from sklearn import linear_model, datasets

n_samples = 1000
n_outliers = 50
X, y, coef = datasets.make_regression(n_samples=n_samples, n_features=1,
                                      n_informative=1, noise=10,
                                      coef=True, random_state=0)

# Add outlier data
np.random.seed(0)
X[:n_outliers] = 3 + 0.5 * np.random.normal(size=(n_outliers, 1))
y[:n_outliers] = -3 + 10 * np.random.normal(size=n_outliers)

# Fit line using all data
lr = linear_model.LinearRegression()
lr.fit(X, y)

# Robustly fit linear model with RANSAC algorithm
ransac = linear_model.RANSACRegressor()
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)

# Predict data of estimated models
line_X = np.arange(X.min(), X.max())[:, np.newaxis]
line_y = lr.predict(line_X)
line_y_ransac = ransac.predict(line_X)

# Compare estimated coefficients
print("Estimated coefficients (true, linear regression, RANSAC):")
print(coef, lr.coef_, ransac.estimator_.coef_)
lw = 2
```

```
plt.scatter(X[inlier_mask], y[inlier_mask], color='yellowgreen', marker='.',
            label='Inliers')
plt.scatter(X[outlier_mask], y[outlier_mask], color='black', marker='+',
            label='Outliers')
plt.plot(line_X, line_y, color='navy', linewidth=lw, label='Linear regressor')
plt.plot(line_X, line_y_ransac, color='black', linestyle='--',, linewidth=lw,
         label='RANSAC regressor')
plt.legend(loc='lower right')
plt.xlabel("Input")
plt.ylabel("Response")
plt.show()
```



Python Project on Optimization 7: Support Vector Machine



Python Code of Optimization Project 7: Support Vector Machine

```
"""
Support Vector Regression (SVR) using linear and non-linear kernels
Toy example of 1D regression using linear, polynomial and RBF kernels.
Source: http://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html#sphx-glr-auto-examples-svm-plot-svm-regression-py
Related resources: https://github.com/tanmoyie/Operations-Research/tree/master/Machine%20Learning%20in%20Optimization
"""
print(__doc__)

import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt

# #############################################################################
# Generate sample data
X = np.sort(5 * np.random.rand(40, 1), axis=0)
y = np.sin(X).ravel()

# #############################################################################
```

```python
# Add noise to targets
y[::5] += 3 * (0.5 - np.random.rand(8))

# #############################################################################
# Fit regression model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
svr_lin = SVR(kernel='linear', C=1e3)
svr_poly = SVR(kernel='poly', C=1e3, degree=2)
y_rbf = svr_rbf.fit(X, y).predict(X)
y_lin = svr_lin.fit(X, y).predict(X)
y_poly = svr_poly.fit(X, y).predict(X)

# #############################################################################
# Look at the results
lw = 2
plt.scatter(X, y, color='black', label='data')
plt.plot(X, y_rbf, color='red', lw=3, label='RBF model')
plt.plot(X, y_lin, color='c', lw=lw, linestyle='-.', label='Linear model')
plt.plot(X, y_poly, color='navy', lw=lw, linestyle='--', label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```

Network Optimization

Min Cost

Max Flow

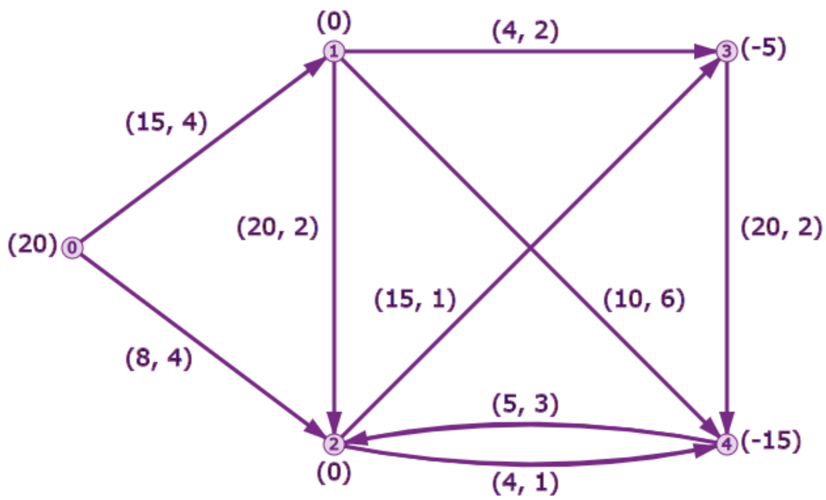
Minimum Spanning Tree

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 Project on Optimization 8: Min Cost Flow

The graph below shows a min cost flow problem. The arcs are labeled with pairs of numbers: the first number is the capacity and the second number is the cost. The numbers in parentheses next to the nodes represent supplies or demands. Node 0 is a supply node with supply 20, while nodes 3 and 4 are demand nodes, with demands -5 and -15, respectively.



Output obtained (https://www.kaggle.com/tanmoyie/min-cost-flow-google-developer)

Minimum cost: 142

Arc	Flow / Capacity	Cost
0 -> 1	12 / 15	48
0 -> 2	8 / 8	32
1 -> 2	8 / 20	16
1 -> 3	4 / 4	8
1 -> 4	0 / 10	0
2 -> 3	12 / 15	12
2 -> 4	4 / 4	4
3 -> 4	11 / 20	22
4 -> 2	0 / 5	0

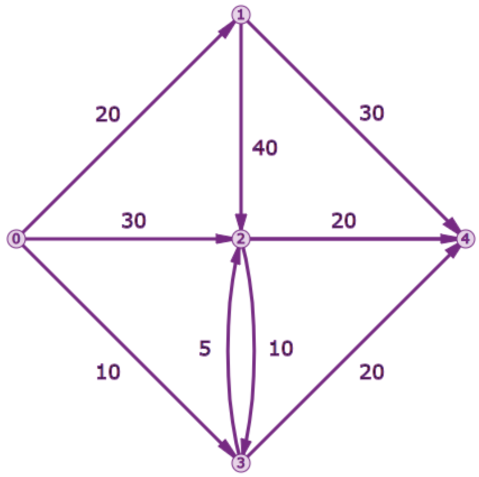
Python Code of Optimization Project 8: Min Cost Flow problem

```
"""
Running kernel: https://www.kaggle.com/tanmoyie/min-cost-flow-google-developer
Source: https://developers.google.com/optimization/flow/mincostflow

"""

# """From Bradley, Hax, and Magnanti, 'Applied Mathematical Programming', figure 8.1."""

```

```python
from __future__ import print_function
from ortools.graph import pywrapgraph

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

  # Define four parallel arrays: start_nodes, end_nodes, capacities, and unit costs
  # between each pair. For instance, the arc from node 0 to node 1 has a
  # capacity of 15 and a unit cost of 4.

  start_nodes = [0, 0, 1, 1, 1, 2, 2, 3, 4]
  end_nodes = [1, 2, 2, 3, 4, 3, 4, 4, 2]
  capacities = [15, 8, 20, 4, 10, 15, 5, 20, 4]
  unit_costs = [4, 4, 2, 2, 6, 1, 3, 2, 3]

  # Define an array of supplies at each node.

  supplies = [20, 0, 0, -5, -15]

# Instantiate a SimpleMinCostFlow solver.
  min_cost_flow = pywrapgraph.SimpleMinCostFlow()

  # Add each arc.
  for i in range(0, len(start_nodes)):
    min_cost_flow.AddArcWithCapacityAndUnitCost(start_nodes[i], end_nodes[i],
                                                capacities[i], unit_costs[i])

  # Add node supplies.
  for i in range(0, len(supplies)):
    min_cost_flow.SetNodeSupply(i, supplies[i])
  # Find the minimum cost flow between node 0 and node 4.
  if min_cost_flow.Solve() == min_cost_flow.OPTIMAL:
    print('Minimum cost:', min_cost_flow.OptimalCost())
    print('')
    print('  Arc    Flow / Capacity  Cost')
    for i in range(min_cost_flow.NumArcs()):
      cost = min_cost_flow.Flow(i) * min_cost_flow.UnitCost(i)
      print('%1s -> %1s   %3s  / %3s       %3s' % (
          min_cost_flow.Tail(i),
          min_cost_flow.Head(i),
          min_cost_flow.Flow(i),
          min_cost_flow.Capacity(i),
          cost))
  else:
    print('There was an issue with the min cost flow input.')

if __name__ == '__main__':
  main()
```

Python Project on Optimization 9: Max Flow

We wish to transport material from node 0 (the source) to node 4 (the sink). The numbers next to the arcs are their capacities — the capacity of an arc is the maximum amount that can be transported across it in a fixed period of time. The capacities are the constraints for the problem.



Output obtained

```
Max flow: 60

 Arc    Flow / Capacity
0 -> 1   20  /  20
0 -> 2   30  /  30
0 -> 3   10  /  10
1 -> 2    0  /  40
1 -> 4   20  /  30
2 -> 3   10  /  10
2 -> 4   20  /  20
3 -> 2    0  /   5
3 -> 4   20  /  20
Source side min-cut: [0]
Sink side min-cut: [4, 1]
```

Python Code of Optimization Project 9: Max Flow

```
"""
Working Code: https://www.kaggle.com/tanmoyie/max-flow-google-developer
From Taha 'Introduction to Operations Research', example 6.4-2.
Source: https://developers.google.com/optimization/flow/maxflow
"""
```

```python
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 Project on Optimization 10: Airlines Network Optimization

Python Code of Optimization Project 10: Airlines Network Optimization[4] *(airlines_network_optimization.py)*

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  5 20:09:14 2018

@adopted by: Tanmoy Das
Earlier version: https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/

"""

# import the libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os

# load the dataset
data = pd.read_csv('airlines_network_optimization.csv') # download the csv file in your local directory and play with it.

# data.shape
# converting sched_dep_time to 'std' - Scheduled time of departure
data['std'] = data.sched_dep_time.astype(str).str.replace('(\d{2}$)', '') + ':' + data.sched_dep_time.astype(str).str.extract('(\d{2}$)', expand=False) + ':00'
# converting sched_arr_time to 'sta' - Scheduled time of arrival
data['sta'] = data.sched_arr_time.astype(str).str.replace('(\d{2}$)', '') + ':' + data.sched_arr_time.astype(str).str.extract('(\d{2}$)', expand=False) + ':00'
# converting dep_time to 'atd' - Actual time of departure
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'
# converting arr_time to 'ata' - Actual time of arrival
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'
data['date'] = pd.to_datetime(data[['year', 'month', 'day']])
# finally we drop the columns we don't need
data = data.drop(columns = ['year', 'month', 'day'])

import networkx as nx
```

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

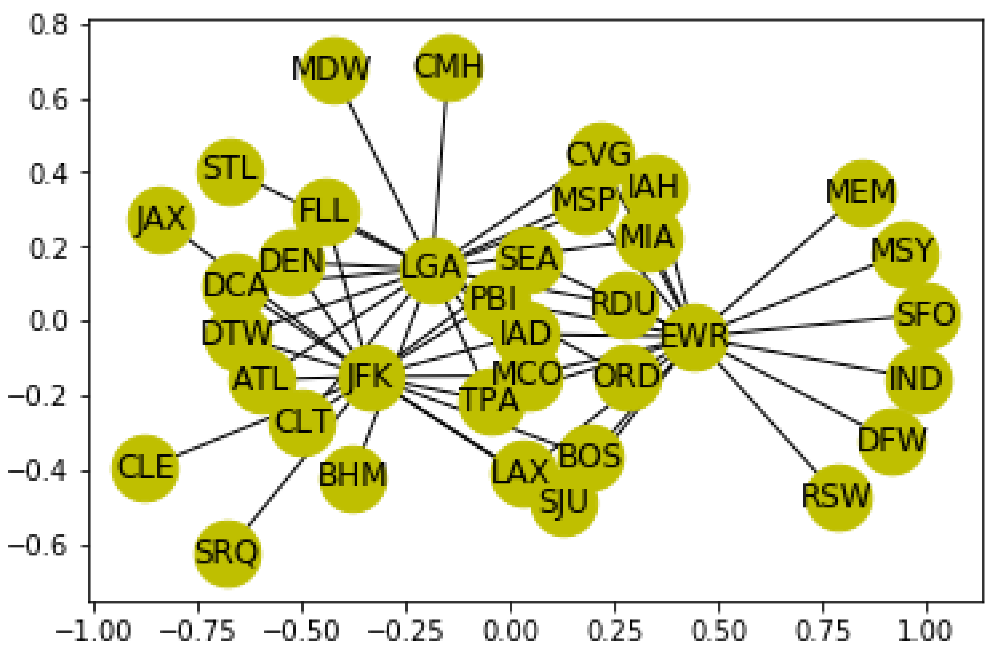
```python
FG = nx.from_pandas_edgelist(data, source='origin', target='dest', edge_attr=True,)
# detail documentation of networkx https://networkx.github.io/documentation/networkx-1.7/reference/generated/networkx.drawing.nx_pylab.draw_networkx.html
FG.nodes()
FG.edges()
nx.draw_networkx(FG, with_labels=True,node_size=600, node_color='y') # Quick view of the Graph. As expected we see 3 very busy airports

nx.algorithms.degree_centrality(FG) # Notice the 3 airports from which all of our 100 rows of data originates
nx.density(FG) # Average edge density of the Graphs
nx.average_shortest_path_length(FG) # Average shortest path length for ALL paths in the Graph
nx.average_degree_connectivity(FG) # For a node of degree k - What is the average of its neighbours' degree?

# Let us find all the paths available
for path in nx.all_simple_paths(FG, source='JAX', target='DFW'):
 print(path)
# Let us find the dijkstra path from JAX to DFW.
# You can read more in-depth on how dijkstra works from this resource - https://courses.csail.mit.edu/6.006/fall11/lectures/lecture16.pdf
dijpath = nx.dijkstra_path(FG, source='JAX', target='DFW')
dijpath
# Let us try to find the dijkstra path weighted by airtime (approximate case)
shortpath = nx.dijkstra_path(FG, source='JAX', target='DFW', weight='air_time')
shortpath
```

Airlines_network_optimization.csv (first 10 records)

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_ti	arr_delay	carrier	flight	tailnum	origin	dest	air_time	distance
2013	2	26	1807	1630	97	1956	1837	79	EV	4411	N13566	EWR	MEM	144	946
2013	8	17	1459	1445	14	1801	1747	14	B6	1171	N661JB	LGA	FLL	147	1076
2013	2	13	1812	1815	-3	2055	2125	-30	AS	7	N403AS	EWR	SEA	315	2402
2013	4	11	2122	2115	7	2339	2353	-14	B6	97	N656JB	JFK	DEN	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	LGA	ORD	116	733
2013	2	14	1442	1445	-3	1833	1747	46	UA	346	N446UA	EWR	MIA	200	1085

2013	7	25	752	755	-3	1037	1057	-20	DL	2395	N909DL	LGA	PBI	140	1035
2013	7	10	557	600	-3	725	715	10	MQ	3267	N542MQ	EWR	ORD	113	719

Table 2: Output of Airlines Network Optimization



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

```
['JAX', 'JFK', 'TPA', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'ORD', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'PBI', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'IAD', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'MIA', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'RDU', 'EWR', 'DFW']
['JAX', 'JFK', 'TPA', 'LGA', 'MSP', 'EWR', 'DFW']
['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