**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](http://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]](#footnote-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***](https://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]](#footnote-2) (an invaluable compendium)

Table of Contents

[Introduction to Operations Research 5](#_Toc516003400)

[Linear Programming 6](#_Toc516003401)

[Math Problem 7](#_Toc516003402)

[Graphical Solution 7](#_Toc516003403)

[Simplex & Dual 8](#_Toc516003404)

[Simplex Method 9](#_Toc516003405)

[Tabular & Big M Solution 9](#_Toc516003406)

[Unbounded & Infeasible Solution 9](#_Toc516003407)

[Revised Simplex 9](#_Toc516003408)

[Dual Problem 9](#_Toc516003409)

[Python Projects on Simplex, Revised Simplex Optimization 10](#_Toc516003410)

[Transportation 11](#_Toc516003411)

[NorthWest Corner Method 12](#_Toc516003412)

[Assignment Problem 12](#_Toc516003413)

[Python Projects on Optimization in Transportation 13](#_Toc516003414)

[Optimization in Machine Learning/ Data Science 21](#_Toc516003415)

[Linear Regression 22](#_Toc516003416)

[Robust Regression 22](#_Toc516003417)

[Support Vector Machine 22](#_Toc516003418)

[Python Projects on Machine Learning Optimization 23](#_Toc516003419)

[Network Optimization 24](#_Toc516003420)

[Min Cost 25](#_Toc516003421)

[Max Flow 25](#_Toc516003422)

[Minimum Spanning Tree 25](#_Toc516003423)

[Python Projects on Network Optimization 26](#_Toc516003424)

[Integer Programming 31](#_Toc516003425)

**Python projects in Optimization**

[Python Code of Optimization Project 1: Revised Simplex 9](#_Toc516001529)

[Python Code of Optimization Project 2: Product Distribution Problem for the PuLP Modeller 13](#_Toc516001530)

[Python Code of Optimization Project 3: Travelling Salesman Problem 18](#_Toc516001531)

[Python Code of Optimization Project 4: Min Cost Flow problem 25](#_Toc516001532)

[Python Code of Optimization Project 5: Airlines Network Optimization 26](#_Toc516001533)

# 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:

**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/e2lHyMl1IYY 3. """ 4. import numpy as np 5. import scipy as sp 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) 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) |

# 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 Projects on Optimization in Transportation & Assignment

Python Project on Optimization 1: Transportation Network for distributing products from source to destination[[3]](#footnote-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.

Python Code of Optimization Project 2: Product Distribution Problem for the PuLP Modeller

|  |
| --- |
| # -\*- 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 3: 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. """ 6. **from** ortools.constraint\_solver **import** pywrapcp 7. **from** ortools.constraint\_solver **import** routing\_enums\_pb2 9. # Distance callback 10. **class** CreateDistanceCallback(object): 11. """Create callback to calculate distances between points.""" 12. **def** \_\_init\_\_(self): 13. """Array of distances between points.""" 15. self.matrix = [ 16. [  0, 290, 250,  230,  190,  334, 365,   40], # Dhaka 17. [290,   0, 337,  453,  396,  560, 581,  244], # Syhlet 18. [250, 337,   0,  495,  396,  540, 120,  240], # Chittagonj 19. [230, 453, 495,    0,  360,  150, 595,  242], # Rajshahi 20. [190, 396, 396,  360,    0,  356, 496,  253], # Jossore 21. [334, 560, 540,  150,  356,    0, 674,  275], # Dinajpur 22. [365, 581, 120,  595,  496,  674,   0,  397], # Coxsbazar 23. [40,  244, 240,  242,  253,  275, 397,    0]] # Narsingdi 24. # distance between Dhaka to Syhlet is 290kms and so on 25. **def** Distance(self, from\_node, to\_node): 26. **return** int(self.matrix[from\_node][to\_node]) 27. **def** main(): 28. # The order of the cities in the array is the following: 29. # Cities 30. city\_names = ["Dhaka", "Syhlet", "Chittagonj", "Rajshahi", "Jossore", "Dinajpur", "Coxsbazar", 31. "Narsingdi"] 32. tsp\_size = len(city\_names) 33. num\_routes = 1    # The number of routes, which is 1 in the TSP. 34. # Nodes are indexed from 0 to tsp\_size - 1. The depot is the starting node of the route. 35. depot = 0 37. # Create routing model 38. **if** tsp\_size > 0: 39. routing = pywrapcp.RoutingModel(tsp\_size, num\_routes, depot) 40. search\_parameters = pywrapcp.RoutingModel.DefaultSearchParameters() 42. # Create the distance callback, which takes two arguments (the from and to node indices) 43. # and returns the distance between these nodes. 44. dist\_between\_nodes = CreateDistanceCallback() 45. dist\_callback = dist\_between\_nodes.Distance 46. routing.SetArcCostEvaluatorOfAllVehicles(dist\_callback) 47. # Solve, returns a solution if any. 48. assignment = routing.SolveWithParameters(search\_parameters) 49. **if** assignment: 50. # Solution cost. 51. **print** ("Total distance: " + str(assignment.ObjectiveValue()) + " miles\n") 52. # Inspect solution. 53. # Only one route here; otherwise iterate from 0 to routing.vehicles() - 1 54. route\_number = 0 55. index = routing.Start(route\_number) # Index of the variable for the starting node. 56. route = '' 57. **while** **not** routing.IsEnd(index): 58. # Convert variable indices to node indices in the displayed route. 59. route += str(city\_names[routing.IndexToNode(index)]) + ' -> ' 60. index = assignment.Value(routing.NextVar(index)) 61. route += str(city\_names[routing.IndexToNode(index)]) 62. **print** ("Route:\n\n" + route) 63. **else**: 64. **print** ('No solution found.') 65. **else**: 66. **print** ('Specify an instance greater than 0.') 68. **if** \_\_name\_\_ == '\_\_main\_\_': 69. 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 4: Linear Regression (plot\_ols.py)

|  |
| --- |
| 1. #!/usr/bin/python 2. # -\*- coding: utf-8 -\*- 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. 15. The coefficients, the residual sum of squares and the variance score are also 16. calculated. 18. """ 19. **print**(\_\_doc\_\_) 20. # Code source: Jaques Grobler 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 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:] 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) 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)) 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) 57. plt.xticks(()) 58. plt.yticks(()) 59. plt.show() |

Python Code of Optimization Project 5: 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 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) 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) 22. # Fit line using all data 23. lr = linear\_model.LinearRegression() 24. lr.fit(X, y) 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) 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) 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 Projects on Network Optimization

Python Code of Optimization Project 6: 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 5. """ 6. # """From Bradley, Hax, and Magnanti, 'Applied Mathematical Programming', figure 8.1.""" 8. **from** \_\_future\_\_ **import** print\_function 9. **from** ortools.graph **import** pywrapgraph 11. **def** main(): 12. """MinCostFlow simple interface example.""" 14. # Define four parallel arrays: start\_nodes, end\_nodes, capacities, and unit costs 15. # between each pair. For instance, the arc from node 0 to node 1 has a 16. # capacity of 15 and a unit cost of 4. 18. start\_nodes = [0, 0, 1, 1, 1, 2, 2, 3, 4] 19. end\_nodes = [1, 2, 2, 3, 4, 3, 4, 4, 2] 20. capacities = [15, 8, 20, 4, 10, 15, 5, 20, 4] 21. unit\_costs = [4, 4, 2, 2, 6, 1, 3, 2, 3] 23. # Define an array of supplies at each node. 25. supplies = [20, 0, 0, -5, -15] 27. # Instantiate a SimpleMinCostFlow solver. 28. min\_cost\_flow = pywrapgraph.SimpleMinCostFlow() 30. # Add each arc. 31. **for** i **in** range(0, len(start\_nodes)): 32. min\_cost\_flow.AddArcWithCapacityAndUnitCost(start\_nodes[i], end\_nodes[i], 33. capacities[i], unit\_costs[i]) 35. # Add node supplies. 37. **for** i **in** range(0, len(supplies)): 38. min\_cost\_flow.SetNodeSupply(i, supplies[i])  41. # Find the minimum cost flow between node 0 and node 4. 42. **if** min\_cost\_flow.Solve() == min\_cost\_flow.OPTIMAL: 43. **print**('Minimum cost:', min\_cost\_flow.OptimalCost()) 44. **print**('') 45. **print**('  Arc    Flow / Capacity  Cost') 46. **for** i **in** range(min\_cost\_flow.NumArcs()): 47. cost = min\_cost\_flow.Flow(i) \* min\_cost\_flow.UnitCost(i) 48. **print**('%1s -> %1s   %3s  / %3s       %3s' % ( 49. min\_cost\_flow.Tail(i), 50. min\_cost\_flow.Head(i), 51. min\_cost\_flow.Flow(i), 52. min\_cost\_flow.Capacity(i), 53. cost)) 54. **else**: 55. **print**('There was an issue with the min cost flow input.') 57. **if** \_\_name\_\_ == '\_\_main\_\_': 58. main() |

Python Code of Optimization Project 7: Airlines Network Optimization[[4]](#footnote-4)

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. """ 3. Created on Tue Jun  5 20:09:14 2018 5. @adopted by: Tanmoy Das 6. Earlier version: https://www.analyticsvidhya.com/blog/2018/04/introduction-to-graph-theory-network-analysis-python-codes/ 8. """ 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 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. 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}$)', '') + ':' + data.sched\_dep\_time.astype(str).str.extract('(\d{2}$)', 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}$)', '') + ':' + data.sched\_arr\_time.astype(str).str.extract('(\d{2}$)', 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']) 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 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 Graph 41. nx.average\_degree\_connectivity(FG) # For a node of degree k - What is the average of its neighbours' degree? 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. dijpath = nx.dijkstra\_path(FG, source='JAX', target='DFW') 49. dijpath 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 |

Table 2: Output of Airlines Network Optimization

|  |
| --- |
||

# Integer Programming



**Further Reference in Integer Programming:**

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

1. Latter two chapters is estimated to be covered by another instructor, hence, skipped for this document! [↑](#footnote-ref-1)
2. A technical writeup [↑](#footnote-ref-2)
3. Source: https://www.coin-or.org/PuLP/CaseStudies/a\_transportation\_problem.html [↑](#footnote-ref-3)
4. Formatting & Color help from http://www.planetb.ca/syntax-highlight-word [↑](#footnote-ref-4)