**Step 1:** To mathematically express the above assignment problem we consider the generalized form of an assignment problem in which *n* resources are to be assigned to *n* activities. The cost of assigning resource *i* to activity *j* is known as $c_{ij}$. Table 13.13 describes cost matrix for the problem.

Table 13.13: Cost matrix for assignment problem

| Resources | Activities | | | | |
|---|---|---|---|---|---|
| | $A_1$ | $A_2$ | … | $A_n$ | Available |
| $R_1$ | $c_{11}$ | $c_{12}$ | … | $c_{1n}$ | 1 |
| $R_2$ | $c_{21}$ | $c_{22}$ | … | $c_{2n}$ | 1 |
| R3 | $c_{31}$ | $c_{32}$ | … | $c_{3n}$ | 1 |
| … | … | … | … | … | … |
| $R_n$ | $c_{n1}$ | $c_{n2}$ | … | $c_{nn}$ | |
| Required | 1 | 1 | 1 | 1 | 1 |

The cost matrix is same as it is with the transportation problem. However, this time the requirement at each of the destinations and the availability at each of the resources is unity (1). This is because of the fact that assignments are to be made on one-to-one basis.

**Step 2:** Let $x_{ij}$ denotes the assignment of the $i^{th}$ resource to $j^{th}$ activity, such that:

$$x_{ij} = \begin{cases} 1, & \text{if resource i is assigned to activity j} \\ 0, & \text{Otherwise} \end{cases} \qquad (13.12)$$

**Step 2:** Following above notions, the generalized assignment problem can mathematically be formulated as shown in Table 13.14. Equation 13.13 expresses the objective function to be minimized while equation 13.14 expresses the constraints.

Table 13.14: Mathematical model for generalized assignment problem

| **Mathematical Model for the generalized assignment problem** |
|---|
| Minimize: |
| $z = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \, . \, x_{ij}$ (13.13) |
| Subject to the constraints: |
| $\sum_{i=1}^{n} x_{ij} = 1 \; and \; \sum_{j=1}^{n} x_{ij} = 1; where \; x_{ij} = 0 \; or \; 1$ (13.14) |
| $for \; all \; i = 1, 2, 3, \ldots, n \; and \; j = 1, 2, 3, \ldots, n$ |

**Step 3, 4, 5:** Based on above general mathematical formulation, the problem considered in the present case study could be formulated as shown in Table 13.15. Equation 13.15 expresses the objective function while equation 13.16 expresses the constraints.

Table 13.15: Mathematical model for case study assignment problem

| **Mathematical Model for the case study assignment problem** |
|---|
| Minimize: |
| $$z = \sum_{i=1}^{4} \sum_{j=1}^{4} c_{ij} \, . \, x_{ij} \qquad (13.15)$$ |
| Subject to the constraints: |
| $$\sum_{i=1}^{4} x_{ij} = 1 \; and \; \sum_{j=1}^{4} x_{ij} = 1; where \; x_{ij} = 0 \; or \; 1 \qquad (13.16)$$ $$for \; all \; i = 1, 2, 3, 4 \; and \; j = 1, 2, 3, 4$$ |

**Step 6, 7:** As obvious from the mathematical model of the problem, solution to the model requires the use of two dimensional subscripted variables and Python dictionaries to be utilized. The author selects Google's OR-Tool CBC solver (a MIP solver) and Python language to run the above model. Python code to solve the model are listed in Table 13.16. The code also illustrate how to solve the assignment problem using a mixed-integer programming (MIP) solver:

Table 13.16: Python code to solve case study assignment problem

**Python code for Assignment Problem**

```
#Install Required Package
#!pip install ortools #Execute only once
#Import required functions
from __future__ import print_function
from ortools.linear_solver import pywraplp
def assignment():
  "Initialize Problem Data"
  pd = {}
  pd['cmatrix']=[
            [18,26,17,11],
            [13,28,14,26],
            [38,19,18,15],
            [19,26,24,10],
  ]
```

```
  pd['ntasks']=4
  pd['nsubs']=4
  return pd
pd=assignment()
#Enable the CBC MIP solver
sol=pywraplp.Solver.CreateSolver('amip', 'CBC')
#Create integer variables to hold 0 0r 1
x={}
for i in range(pd['ntasks']):
  for j in range(pd['nsubs']):
    x[i,j]=sol.IntVar(0, 1, '')
print('Decision variables =', sol.NumVariables())
#Enable Constraint 1: Only one task is to be assigned to each s
ubordinate
for i in range(pd['ntasks']):
  sol.Add(sol.Sum([x[i, j] for j in range(pd['nsubs'])]) == 1)

#Enable Constraint 2: Each subordinate should receives exactly
one task
for j in range(pd['nsubs']):
  sol.Add(sol.Sum([x[i, j] for i in range(pd['ntasks'])]) == 1)

#Formulate the objective function
objf=[]
for i in range(pd['ntasks']):
  for j in range(pd['nsubs']):
    objf.append(pd['cmatrix'][i][j] * x[i, j])
sol.Minimize(sol.Sum(objf))
st=sol.Solve()
if st == pywraplp.Solver.OPTIMAL or st == pywraplp.Solver.FEASI
BLE:
        print('Total cost = ', sol.Objective().Value(), '\n')
        for i in range(pd['nsubs']):
            for j in range(pd['ntasks']):
                # Test if x[i,j] is 1 (or greater than 0.5 in w
hich case it should round off to 1.0).
                if x[i, j].solution_value() > 0.5:
                    print('Subordinate %d id assigned to Task %
d.  Assignment Cost = %d' %
                          (i+1, j+1, pd['cmatrix'][i][j]))
```

**Output**

```
Decision variables = 16
Total cost =  59.0
Subordinate 1 id assigned to Task 3.  Assignment Cost = 17
Subordinate 2 id assigned to Task 1.  Assignment Cost = 13
Subordinate 3 id assigned to Task 2.  Assignment Cost = 19
Subordinate 4 id assigned to Task 4.  Assignment Cost = 10
```

**Step 8:** Model Verification: This could be done in the same way as we had done in the last step of Case Study 1. We will notice that all constraint are satisfied so the model is valid.

The above stated assignment problem can also be formulated and solved using GLOP solver as a linear programming problem. The author leaves it as an exercise for the reader.