Khemis Miliana University
Faculty of Science and Technology
Mathematics and Computer Science Department

جامعة جيلالي بونعامة– خميس مليانة
كلية العلوم والتكنولوجيا
قسم الرياضيات و الاعلام الالي

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

Tutorial sheet 3

Problem: Network flow problem (Transportation problem)

Task: Solve the problem using c++/python programming language.

Solution: cpp 1

```
#include <stdio.h>
#include <imsl.h>

#define NS 5
#define ND 6

int main() {
	float cmin, *x;
	float sup[NS] = { 300, 300, 600, 600, 600 };
	float dem[ND] = { 200, 100, 300, 600, 600, 600 };
	float cost[NS][ND] = {
		{ 1000, 1000, 1000, 16, 10, 12 },
		{ 1000, 1000, 1000, 15, 14, 17 },
		{ 6, 8, 10, 0, 1000, 1000 },
		{ 7, 11, 11, 1000, 0, 1000 },
		{ 4, 5, 12, 1000, 1000, 0 }
	};

	x = imsl_f_transport(NS, ND, sup, dem, &cost[0][0],
		IMSL_TOTAL_COST, &cmin, 0);

	printf("Minimum cost is $%.2f", cmin);

	imsl_f_write_matrix("Solution Matrix", NS, ND, x,
		IMSL_NO_ROW_LABELS, IMSL_NO_COL_LABELS, 0);

	imsl_free(x);
	return 1;
}
```

Solution: Python

```
# Import PuLP modeler functions
from pulp import *

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

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

# Creates a dictionary for the number of units of supply for each supply
node
supply = {"A": 100, "B": 200, "C":200}

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

# Creates a dictionary for the number of units of demand for each demand
```

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

```
node
demand = {
    "1": 50,
    "2": 150,
    "3": 300,
}

# Intermediate nodes
warehouses=["P","Q"]

# Creates a list of costs of each transportation path
costs_1 = [   # warehouses
    [3,2],   # A factories
    [4,3],   # B
    [2.5,3.5] # C
]

costs_2 = [   # projects
    [2,1,4],   # P warehouses
    [3,2,5],   # Q
]

# The cost data is made into a dictionary
costs_1 = makeDict([factories, warehouses], costs_1, 0)

# The cost data is made into a dictionary
costs_2 = makeDict([warehouses, projects], costs_2, 0)
# Creates a list of tuples containing all the possible routes for transport
Routes = [(w, b) for w in warehouses for b in projects]

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

# Creates a list of tuples containing all the possible routes for transport
Routes_2 = [(w, b) for w in warehouses for b in projects]

# A dictionary called 'Vars_2' is created to contain the referenced
variables(the routes)
vars_2 = LpVariable.dicts("Route", (warehouses, projects), 0, None,
LpInteger)

# The objective function is added to 'prob' first
prob += (
    lpSum([vars[w][b] * costs_1[w][b] for (w, b) in Routes]) +
lpSum([vars_2[w][b] * costs_2[w][b] for (w, b) in Routes_2]),
    "Sum_of_Transporting_Costs",
)

# The supply maximum constraints are added to prob for each supply node
(factories)
for w in factories:
    prob += (
        lpSum([vars[w][b] for b in warehouses]) <= supply[w],
        "Sum_of_Products_out_of_factories_%s" % w,
```

Module : Operations Research 1

Responsible: Dr. I. Ait Abderrahim

```
    )

# The demand minimum constraints are added to prob for each demand node
(project)
for b in projects:
    prob += (
        lpSum([vars_2[w][b] for w in warehouses]) >= demand[b],
        "Sum_of_Products_into_projects%s" % b,
    )

# Transshipment constraints: What is shipped into a transshipment node must
ne shipped out.
for w in warehouses:
    prob += (
        lpSum([vars[f][w] for f in factories]) - lpSum([vars_2[w][p] for p
in projects])== 0,
        "Sum_of_Products_out_of_warehouse_%s" % w,
    )

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

# Print the variables optimized value
for v in prob.variables():
    print(v.name, "=", v.varValue)

# The optimised objective function value is printed to the screen
print("Value of Objective Function = ", value(prob.objective))
```

Correct answer: