Operations Research

Aim: Solve the following problem by using Vogel's Approximation Method:

- A company has a current shipping schedule, which is being questioned by the management as to whether or not it is optimal. The firm has three factories and five warehouses. The necessary data in terms of transportation costs in ₹ per unit from a factory to a destination and factory capacities and warehouse requirements are as follows:

		Factories			Requirements
		X	Y	Z	
Warehouses	A	5	4	8	400
	B	8	7	4	400
	C	6	7	6	500
	D	6	6	6	400
	E	3	5	4	800
Capacity		800	600	1100	

Solve for a basic feasible shipping schedule in terms of lowest possible shipping cost.[1]

Theory:

Vogel's Approximation Method (VAM) is one of the methods used to calculate the initial basic feasible solution to a transportation problem. However, it is an iterative procedure such that in each step, we should find the penalties for each available row and column by taking the least cost and second least cost. [2]

Steps:

1. Identify the two lowest costs in each row and column of the given cost matrix and then write the absolute row and column difference. These differences are called penalties.
2. Identify the row or column with the maximum penalty and assign the corresponding cell's min(supply, demand). If two or more columns or rows have the same maximum penalty, then we can choose one among them as per our convenience.
3. If the assignment in the previous satisfies the supply at the origin, delete the corresponding row. If it satisfies the demand at that destination, delete the corresponding column.
4. Stop the procedure if supply at each origin is 0, i.e., every supply is exhausted, and demand at each destination is 0, i.e., every demand is satisfying. If not, repeat the above steps. [2]

Solution:

		Factories			Requirements
		X	**Y**	**Z**	
Warehouses	**A**	5	4	8	400
	B	8	7	4	400
	C	6	7	6	500
	D	6	6	6	400
	E	3	5	4	800
Capacity		800	600	1100	2500

As Σ Capacity = Σ Requirements, the above problem is balanced.

We will use Vogel's Approximation Method to solve the above problem.

Iteration 1:

		Factories			Requirements	Penalties
		X	Y	Z		
Warehouses	A	5	4	8	400	1
	B	8	7	$^{400}4$	0	4
	C	6	7	6	500	0
	D	6	6	6	400	0
	E	3	5	4	800	1
Capacity		800	600	700	2100	-
Penalties		2	1	0	-	-

Iteration 2:

		Factories			Requirements	Penalties
		X	Y	Z		
Warehouses	A	5	4	8	400	1
	B	8	7	$^{400}4$	0	
	C	6	7	6	500	0
	D	6	6	6	400	0
	E	$^{800}3$	5	4	0	1
Capacity		0	600	700	1300	-
Penalties		2	1	2	-	-

Iteration 3:

		Factories			Requirements	Penalties
		X	Y	Z		
Warehouses	A	5	$^{400}4$	8	0	4
	B	8	7	$^{400}4$	0	
	C	6	7	6	500	1
	D	6	6	6	400	0
	E	$^{800}3$	5	4	0	
Capacity		0	200	700	900	-
Penalties			2	0	-	-

Iteration 4:

		Factories			Requirements	Penalties
		X	Y	Z		
Warehouses	A	5	$^{400}4$	8	0	
	B	8	7	$^{400}4$	0	
	C	6	7	$^{500}6$	0	1
	D	6	6	6	400	0
	E	$^{800}3$	5	4	0	
Capacity		0	200	200	400	-
Penalties			1	0	-	-

Iteration 5:

		Factories			Requirements	Penalties
		X	Y	Z		
Warehouses	A	5	^{400}4	8	0	
	B	8	7	^{400}4	0	
	C	6	7	^{500}6	0	
	D	6	^{200}6	^{200}6	0	
	E	^{800}3	5	4	0	
Capacity		0	0	0	0	-
Penalties					-	-

Total cost = 400 x 4 + 400 x 4 + 500 x 6 + 200 x 6 + 200 x 6 + 800 x 3

= 1600 + 1600 + 3000 + 1200 + 1200 + 2400

= 11000.

Program:

```
#include
<iostream
>
            #include <numeric>
            #include <vector>


            template <typename T>
            std::ostream &operator<<(std::ostream &outStream, const std::vector<T> v) {
                    auto iterator = v.cbegin();
                    auto end = v.cend();


                    outStream << '[';
                    if (iterator != end) {
                            outStream << *iterator;
                            iterator = std::next(iterator);
                    }


                    while (iterator != end) {
                            outStream << ", " << *iterator;
                            iterator = std::next(iterator);
```

```
        }


        return outStream << ']';
}


std::vector<int> demand = {800, 600, 1100};
std::vector<int> supply = {400, 400, 500, 400, 800};
std::vector<std::vector<int>> costs = {
        {5, 4, 8},
        {8, 7, 4},
        {6, 7, 6},
        {6, 6, 6},
        {3, 5, 4}
};


int rowCount = supply.size();
int columnCount = demand.size();


std::vector<bool> rowsDone(rowCount, false);
std::vector<bool> columnsDone(columnCount, false);
std::vector<std::vector<int>> result(rowCount, std::vector<int>(columnCount, 0));


std::vector<int> difference(int index, int length, bool isRow) {
        int min1 = INT_MAX;
        int min2 = INT_MAX;
        int minP = -1;


        for (int i = 0; i < length; i++) {
                if (isRow ? columnsDone[i] : rowsDone[i]) {
                        continue;
                }


                int cost = isRow ? costs[index][i] : costs[i][index];
                if (cost < min1) {
                        min2 = min1;
                        min1 = cost;
                        minP = i;
                }
                else if (cost < min2) {
                        min2 = cost;
                }
        }
```

```
        return {min2 - min1, min1, minP};
}


std::vector<int> maxPenality(int length1, int length2, bool isRow) {
        int md = INT_MIN;
        int pc = -1;
        int pm = -1;
        int mc = -1;


        for (int i = 0; i < length1; ++i)
        {
                if (isRow ? rowsDone[i] : columnsDone[i]) {
                        continue;
                }


                std::vector<int> res = difference(i, length2, isRow);
                if(res[0] > md) {
                        md = res[0];
                        pm = i;
                        mc = res[1];
                        pc = res[2];
                }
        }


        return isRow ? std::vector<int> {pm, pc, mc, md} : std::vector<int> {pc, pm, mc,
md};
}


std::vector<int> nextCell() {
        auto res1 = maxPenality(rowCount, columnCount, true);
        auto res2 = maxPenality(columnCount, rowCount, false);


        if (res1[3] == res2[3]) {
                return res1[2] < res2[2] ? res1 : res2;
        }


        return res1[3] > res2[3] ? res2 : res1;
}


int main(int argc, char const *argv[]) {
        int supplyLeft = std::accumulate(supply.cbegin(), supply.cend(), 0, [](int a,
int b) { return a + b; });
        int totalCost = 0;
```

```
        while (supplyLeft > 0) {
                auto cell = nextCell();
                int r = cell[0];
                int c = cell[1];


                int quantity = std::min(demand[c], supply[r]);


                demand[c] -= quantity;
                if (demand[c] == 0) {
                        columnsDone[c] = true;
                }


                supply[r] -= quantity;
                if (supply[r] == 0) {
                        rowsDone[r] = true;
                }


                result[r][c] = quantity;
                supplyLeft -= quantity;


                totalCost += quantity * costs[r][c];
        }


        for (auto &a : result)
        {
                std::cout << a << '\n';
        }


        std::cout << "Total cost:" << totalCost;


        return 0;
}
```

Output:

Used Software(s):

- g++ 11.3.0 (MinGW-w64) (link).
- Nu shell 0.65.0 (link).
- Windows Terminal 1.14.1861.0 (link).