Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

COMPTE RENDU

Mathematics for Engineering - TP4
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant : HATHOUTI Mohammed Taha

Filière : Cybersecurité

Année : 2024/2025

Enseignants : Mme.OUFASKA

Date : 15 décembre 2025

Table des matières

Introduction

L'objectif de ce TP est de résoudre des problèmes de transport en appliquant les méthodes du Coin Nord-Ouest et de Balas-Hammer. Dans un premier temps, nous allons calculer manuellement une solution initiale pour chaque méthode. Ensuite, nous décomposerons chaque méthode en plusieurs fonctions intermédiaires afin de simplifier l'implémentation des algorithmes.

1 Implémentation de la Méthode du Coin Nord-Ouest

1.1 Fonction 1 : Calcul du minimum

Description : Cette fonction calcule le minimum entre l'offre et la demande à l'étape (i, j).

```
# Calcule le minimum entre l'offre et la demande a l'etape (i, j)
def calculer_minimum(offre, demande, i, j):
    return min(offre[i], demande[j])
```

Listing 1 – Fonction calculer_minimum

1.2 Fonction 2 : Mise à jour

Description : Cette fonction met à jour les vecteurs offre et demande après chaque allocation.

```
# Met a jour les vecteurs offre et demande apres chaque
   allocation
def mettre_a_jour(offre, demande, i, j, allocation):
    offre[i] -= allocation
    demande[j] -= allocation
```

Listing 2 – Fonction mettre_a_jour

1.3 Fonction 3 : Algorithme du Coin Nord-Ouest

Description : Cette fonction implémente l'algorithme complet du Coin Nord-Ouest en utilisant les fonctions intermédiaires définies précédemment.

```
# Implemente l'algorithme du Coin Nord-Ouest
def coin_nord_ouest(offre, demande):
    # Travailler sur des copies pour ne pas modifier les
       originaux
    offre_copy = copy.deepcopy(offre)
    demande_copy = copy.deepcopy(demande)

    m = len(offre)
    n = len(demande)
    allocation = np.zeros((m, n))

    i = 0
```

```
    j = 0

    print("\n=== Methode du Coin Nord-Ouest ===\n")

    while i < m and j < n:
        # Calculer l'allocation
        alloc = calculer_minimum(offre_copy, demande_copy, i, j)
        allocation[i][j] = alloc

        print(f"Cellule ({i+1}, {j+1}): allocation = {alloc}")

        # Mettre a jour offre et demande
        mettre_a_jour(offre_copy, demande_copy, i, j, alloc)

        # Avancer selon l'algorithme
        if offre_copy[i] == 0:
            i += 1
        elif demande_copy[j] == 0:
            j += 1

    return allocation
```

Listing 3 – Fonction coin_nord_ouest

2 Implémentation de la Méthode de Balas-Hammer

2.1 Fonction 1 : Calcul des différences

Description : Cette fonction calcule les différences pour chaque ligne et colonne. La différence pour une ligne est la différence entre les deux plus petits coûts disponibles dans cette ligne. De même pour les colonnes.

```
# Calcule les differences pour chaque ligne et colonne
def calculer_differences(couts, offre, demande):
    m, n = couts.shape
    diff_lignes = []
    diff_colonnes = []

    # Differences pour les lignes
    for i in range(m):
        if offre[i] > 0:
            # Trouver les deux plus petits couts disponibles dans
                la ligne
            couts_disponibles = []
            for j in range(n):
                if demande[j] > 0:
                    couts_disponibles.append(couts[i][j])

            if len(couts_disponibles) >= 2:
                couts_disponibles.sort()
```

```
                diff = couts_disponibles[1] - couts_disponibles
                    [0]
            elif len(couts_disponibles) == 1:
                diff = 0
            else:
                diff = -1
            diff_lignes.append(diff)
        else:
            diff_lignes.append(-1)

    # Differences pour les colonnes
    for j in range(n):
        if demande[j] > 0:
            # Trouver les deux plus petits couts disponibles dans
                la colonne
            couts_disponibles = []
            for i in range(m):
                if offre[i] > 0:
                    couts_disponibles.append(couts[i][j])

            if len(couts_disponibles) >= 2:
                couts_disponibles.sort()
                diff = couts_disponibles[1] - couts_disponibles
                    [0]
            elif len(couts_disponibles) == 1:
                diff = 0
            else:
                diff = -1
            diff_colonnes.append(diff)
        else:
            diff_colonnes.append(-1)

    return diff_lignes, diff_colonnes
```

Listing 4 – Fonction calculer_differences

2.2 Fonction 2 : Trouver la position optimale

Description : Cette fonction détermine la position optimale pour l'allocation actuelle en trouvant la ligne ou la colonne avec la différence maximale, puis en choisissant la cellule avec le coût minimum.

```
# Determine la position optimale pour l'allocation actuelle
def trouver_position_optimale(couts, differences, offre, demande)
    :
    diff_lignes, diff_colonnes = differences
    m, n = couts.shape

    # Trouver la difference maximale
    max_diff_ligne = max(diff_lignes) if diff_lignes else -1
    max_diff_colonne = max(diff_colonnes) if diff_colonnes else
        -1
```

```
    max_diff = max(max_diff_ligne, max_diff_colonne)

    # Determiner si c'est une ligne ou une colonne
    if max_diff == max_diff_ligne and max_diff >= 0:
        # C'est une ligne
        i = diff_lignes.index(max_diff)
        # Trouver la cellule avec le cout minimum dans cette
           ligne
        min_cout = float('inf')
        j_min = -1
        for j in range(n):
            if demande[j] > 0 and couts[i][j] < min_cout:
                min_cout = couts[i][j]
                j_min = j
        return i, j_min
    else:
        # C'est une colonne
        j = diff_colonnes.index(max_diff)
        # Trouver la cellule avec le cout minimum dans cette
           colonne
        min_cout = float('inf')
        i_min = -1
        for i in range(m):
            if offre[i] > 0 and couts[i][j] < min_cout:
                min_cout = couts[i][j]
                i_min = i
        return i_min, j
```

Listing 5 – Fonction trouver_position_optimale

2.3 Fonction 3 : Algorithme de Balas-Hammer

Description : Cette fonction implémente l'algorithme complet de Balas-Hammer en utilisant les fonctions intermédiaires définies précédemment.

```
# Implemente l'algorithme de Balas-Hammer
def balas_hammer(offre, demande, couts):
    # Copier pour ne pas modifier les originaux
    offre_copy = copy.deepcopy(offre)
    demande_copy = copy.deepcopy(demande)

    m = len(offre)
    n = len(demande)
    allocation = np.zeros((m, n))

    iteration = 1
    print("\n=== Methode de Balas-Hammer ===\n")

    while sum(offre_copy) > 0 and sum(demande_copy) > 0:
        print(f"Iteration {iteration}:")

        # Calculer les differences
```

```
        differences = calculer_differences(couts, offre_copy,
            demande_copy)
        diff_lignes, diff_colonnes = differences

        print(f"  Differences lignes: {diff_lignes}")
        print(f"  Differences colonnes: {diff_colonnes}")

        # Trouver la position optimale
        i, j = trouver_position_optimale(couts, differences,
            offre_copy, demande_copy)

        # Calculer l'allocation
        alloc = calculer_minimum(offre_copy, demande_copy, i, j)
        allocation[i][j] = alloc

        print(f"  Cellule choisie: ({i+1}, {j+1}) avec cout {
            couts[i][j]}")
        print(f"  Allocation: {alloc}\n")

        # Mettre a jour
        mettre_a_jour(offre_copy, demande_copy, i, j, alloc)

        iteration += 1

    return allocation
```

Listing 6 – Fonction balas_hammer

3 Exercice 1

3.1 Énoncé du problème

Soit le problème de transport suivant :

	D1	D2	D3	D4	D5	Offre
O1	7	12	1	5	9	12
O2	15	3	12	6	14	11
O3	8	16	10	12	7	14
O4	18	8	17	11	16	8
Demande	10	11	15	5	4	

TABLE 1 – Tableau des coûts et contraintes pour l'Exercice 1

3.2 Implémentation en Python

```
print("\n" + "="*70)
print("EXERCICE 1: METHODE DU COIN NORD-OUEST")
print("="*70)

# Donnees du probleme
```

```python
couts = np.array([
    [7,  12, 1,  5,  9],
    [15, 3,  12, 6,  14],
    [8,  16, 10, 12, 7],
    [18, 8,  17, 11, 16]
])

offre = [12, 11, 14, 8]
demande = [10, 11, 15, 5, 4]

print(f"\nOffres:   {offre}")
print(f"Demandes: {demande}")

afficher_matrice_couts(couts)

# Appliquer la methode du Coin Nord-Ouest
allocation = coin_nord_ouest(offre, demande)

# Calculer le cout total
cout_total = calculer_cout_total(allocation, couts)

# Afficher les resultats
afficher_tableau_allocation(allocation, couts, offre, demande,
                            "SOLUTION PAR COIN NORD-OUEST")

print(f"{'='*70}")
print(f"COUT TOTAL: {cout_total}")
print(f"{'='*70}\n")
```

Listing 7 – Code principal pour l'Exercice 1

3.3 Calcul manuel de la solution

Appliquons la méthode du Coin Nord-Ouest manuellement :

Itération 1 : Cellule $(1,1)$: $\min(12,10) = 10$

— Offre O1 : $12 - 10 = 2$

— Demande D1 : $10 - 10 = 0 \Rightarrow$ on avance à droite

Itération 2 : Cellule $(1,2)$: $\min(2,11) = 2$

— Offre O1 : $2 - 2 = 0 \Rightarrow$ on descend

— Demande D2 : $11 - 2 = 9$

Itération 3 : Cellule $(2,2)$: $\min(11,9) = 9$

— Offre O2 : $11 - 9 = 2$

— Demande D2 : $9 - 9 = 0 \Rightarrow$ on avance à droite

Itération 4 : Cellule $(2,3)$: $\min(2,15) = 2$

— Offre O2 : $2 - 2 = 0 \Rightarrow$ on descend

— Demande D3 : $15 - 2 = 13$

Itération 5 : Cellule $(3,3)$: $\min(14,13) = 13$

— Offre O3 : $14 - 13 = 1$

— Demande D3 : $13 - 13 = 0 \Rightarrow$ on avance à droite

Itération 6 : Cellule $(3, 4)$: $\min(1, 5) = 1$

— Offre O3 : $1 - 1 = 0 \Rightarrow$ on descend

— Demande D4 : $5 - 1 = 4$

Itération 7 : Cellule $(4, 4)$: $\min(8, 4) = 4$

— Offre O4 : $8 - 4 = 4$

— Demande D4 : $4 - 4 = 0 \Rightarrow$ on avance à droite

Itération 8 : Cellule $(4, 5)$: $\min(4, 4) = 4$

— Offre O4 : $4 - 4 = 0$

— Demande D5 : $4 - 4 = 0 \Rightarrow$ terminé

3.4 Calcul du coût total

Le coût total est calculé comme suit :

$$z = 10 \times 7 + 2 \times 12 + 9 \times 3 + 2 \times 12 + 13 \times 10 + 1 \times 12 + 4 \times 11 + 4 \times 16 \quad (1)$$
$$= 70 + 24 + 27 + 24 + 130 + 12 + 44 + 64 \quad (2)$$
$$= 395 \quad (3)$$

3.5 Résultats du programme

```
========================================================================
EXERCICE 1: M THODE DU COIN NORD-OUEST
========================================================================

Offres:   [12, 11, 14, 8]
Demandes: [10, 11, 15, 5, 4]

Matrice des co ts:
----------------------------------------
      D1   D2   D3   D4   D5
O1     7   12    1    5    9
O2    15    3   12    6   14
O3     8   16   10   12    7
O4    18    8   17   11   16


=== M thode du Coin Nord-Ouest ===

Cellule (1, 1): allocation = 10
Cellule (1, 2): allocation = 2
Cellule (2, 2): allocation = 9
Cellule (2, 3): allocation = 2
Cellule (3, 3): allocation = 13
Cellule (3, 4): allocation = 1
Cellule (4, 4): allocation = 4
Cellule (4, 5): allocation = 4

========================================================================
```

```
SOLUTION PAR COIN NORD-OUEST
============================================================================

        D1       D2       D3       D4       D5       Offre
----------------------------------------------------------
O1     10( 7)    2(12)     -( 1)    -( 5)    -( 9)      12
O2      -(15)    9( 3)     2(12)    -( 6)    -(14)      11
O3      -( 8)    -(16)    13(10)    1(12)    -( 7)      14
O4      -(18)    -( 8)     -(17)    4(11)    4(16)      8
----------------------------------------------------------
Dem     10        11        15        5        4

============================================================================
CO T TOTAL: 395.0
============================================================================
```

Interprétation : La méthode du Coin Nord-Ouest a permis de trouver une solution initiale avec un coût total de 395.

4 Exercice 2

4.1 Énoncé du problème

Soit le problème de transport suivant :

	D1	D2	D3	D4	Offre
O1	3	6	4	8	20
O2	3	4	7	9	17
O3	9	4	5	6	13
Demande	12	10	15	13	

TABLE 2 – Tableau des coûts et contraintes pour l'Exercice 2

4.2 Implémentation en Python

```python
print("\n" + "="*70)
print("EXERCICE 2: METHODE DE BALAS-HAMMER")
print("="*70)

# Donnees du probleme
couts = np.array([
    [3, 6, 4, 8],
    [3, 4, 7, 9],
    [9, 4, 5, 6]
])

offre = [20, 17, 13]
demande = [12, 10, 15, 13]

print(f"\nOffres:   {offre}")
print(f"Demandes: {demande}")

```

```
afficher_matrice_couts(couts)

# Appliquer la methode de Balas-Hammer
allocation = balas_hammer(offre, demande, couts)

# Calculer le cout total
cout_total = calculer_cout_total(allocation, couts)

# Afficher les resultats
afficher_tableau_allocation(allocation, couts, offre, demande,
                            "SOLUTION PAR BALAS-HAMMER")

print(f"{'='*70}")
print(f"COUT TOTAL: {cout_total}")
print(f"{'='*70}\n")
```

Listing 8 – Code principal pour l'Exercice 2

4.3 Calcul manuel de la solution

Itération 1 :

- Différences lignes : $[1, 1, 1]$ (pour O1 : $|4-3| = 1$, pour O2 : $|4-3| = 1$, pour O3 : $|5-4| = 1$)
- Différences colonnes : $[0, 2, 1, 2]$ (pour D1 : $|3-3| = 0$, pour D2 : $|6-4| = 2$, etc.)
- Maximum $= 2$ (colonnes D2 et D4)
- Choisir D2 avec coût minimum : cellule $(3, 2)$ avec coût 4
- Allocation : $\min(13, 10) = 10$

Itération 2 :

- Différences lignes : $[1, 1, 1]$
- Différences colonnes : $[0, -, 1, 2]$
- Maximum $= 2$ (colonne D4)
- Choisir D4 avec coût minimum : cellule $(3, 4)$ avec coût 6
- Allocation : $\min(3, 13) = 3$

Itération 3 :

- Différences lignes : $[1, 2, -]$
- Différences colonnes : $[0, -, 3, 1]$
- Maximum $= 3$ (colonne D3)
- Choisir D3 avec coût minimum : cellule $(1, 3)$ avec coût 4
- Allocation : $\min(20, 15) = 15$

Itération 4 :

- Choisir cellule $(1, 1)$ avec coût 3
- Allocation : $\min(5, 12) = 5$

Itération 5 :

— Choisir cellule $(2, 1)$ avec coût 3

— Allocation : $\min(17, 7) = 7$

Itération 6 :

— Choisir cellule $(2, 4)$ avec coût 9

— Allocation : $\min(10, 10) = 10$

4.4 Calcul du coût total

Le coût total est calculé comme suit :

$$z = 5 \times 3 + 15 \times 4 + 7 \times 3 + 10 \times 9 + 10 \times 4 + 3 \times 6 \tag{4}$$

$$= 15 + 60 + 21 + 90 + 40 + 18 \tag{5}$$

$$= 244 \tag{6}$$

4.5 Résultats du programme

```
======================================================================
EXERCICE 2: M THODE DE BALAS-HAMMER
======================================================================

Offres:   [20, 17, 13]
Demandes: [12, 10, 15, 13]

Matrice des co ts:
----------------------------------------
      D1   D2   D3   D4
O1     3    6    4    8
O2     3    4    7    9
O3     9    4    5    6


=== M thode de Balas-Hammer ===

It ration 1:
  Diff rences lignes: [1, 1, 1]
  Diff rences colonnes: [0, 0, 1, 2]
  Cellule choisie: (3, 4) avec co t 6
  Allocation: 13

It ration 2:
  Diff rences lignes: [1, 1, -1]
  Diff rences colonnes: [0, 2, 3, -1]
  Cellule choisie: (1, 3) avec co t 4
  Allocation: 15

It ration 3:
  Diff rences lignes: [3, 1, -1]
  Diff rences colonnes: [0, 2, -1, -1]
  Cellule choisie: (1, 1) avec co t 3
  Allocation: 5

It ration 4:
  Diff rences lignes: [-1, 1, -1]
```

```
  Diff rences colonnes: [0, 0, -1, -1]
  Cellule choisie: (2, 1) avec co t 3
  Allocation: 7

It ration 5:
  Diff rences lignes: [-1, 0, -1]
  Diff rences colonnes: [-1, 0, -1, -1]
  Cellule choisie: (2, 2) avec co t 4
  Allocation: 10


======================================================================
SOLUTION PAR BALAS-HAMMER
======================================================================

        D1       D2       D3       D4       Offre
-------------------------------------------------
O1      5( 3)     -( 6)    15( 4)     -( 8)      20
O2      7( 3)    10( 4)     -( 7)     -( 9)      17
O3      -( 9)     -( 4)     -( 5)    13( 6)      13
-------------------------------------------------
Dem     12        10        15        13

======================================================================
CO T TOTAL: 214.0
======================================================================


======================================================================
R SUM   DES R SULTATS
======================================================================

Exercice 1 (Coin Nord-Ouest): Co t total = 395.0
Exercice 2 (Balas-Hammer):    Co t total = 214.0

======================================================================
```

Interprétation : La méthode de Balas-Hammer a permis de trouver une solution avec un coût total de 244, ce qui est nettement meilleur que la solution obtenue avec la méthode du Coin Nord-Ouest pour l'Exercice 1.