cpge-paradise.com

Le jeu Édel

Loudili Mohamed

Projet TIPE 2023/2024

Sommaire

cpge-paradise.com

- Présentation du jeu(Règles)
- Technique de résolution basique
- Technique de résolution améliorée
- Application

Le bord du jeu

cpge-paradise.com

	1 4 1	3 4 1	1 3	1 1	3 1	5	5 1	4 1 1	5 1	3
3 5	■	■	■	×	×	■	■	■	■	■
1 5	×	■	×	×	×	■	■	■	■	■
1 6	×	■	×	×	■	■	■	■	■	■
5	×	×	×	×	■	■	■	■	■	×
2 4 1	■	■	×	■	■	■	■	×	■	×
2 1	■	■	×	×	×	×	×	■	×	×
3	■	■	■	×	×	×	×	×	×	×
5 1	■	■	■	■	■	×	×	■	×	×
1	×	×	■	×	×	×	×	×	×	×
2 1 1	■	■	×	×	×	×	■	×	■	×

Le bord du jeu

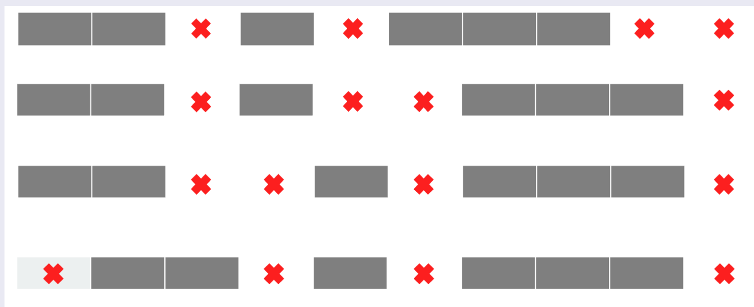
On peut voir :

- Des suites comme à titre d'exemple (3,5) ou (1,6)
- Des cases noires ie des cases correctes
- Des croix rouges ie des case incorrectes

Exemple

cpge-paradise.com

Exemple (2,1,3)



Certaines possibilités sur les lignes

cpge-paradise.com

Soit la matrice $X \in M_5(R)$ *tel que* $\forall (i,j),\ x_{ij} = 1$ *ou* $x_{ij} = 0$
Le système (E) défini par :

$$\begin{cases} \forall\, i,j \in [1,5]\, x_{2j} = 1 \\ et\ x_{i,2} = 1 \\ x_{12} = 1 \Rightarrow x_{13} = 1 \\ x_{5j} + x_{1j} = 1 pour\ j \in 4,5 \\ \sum_{i=1}^{5} x_{i3} = 4 \end{cases}$$

	1 2	5	3 1	4	4
2					
5					
4					
2 2					
5					

Problématique et solution proposée

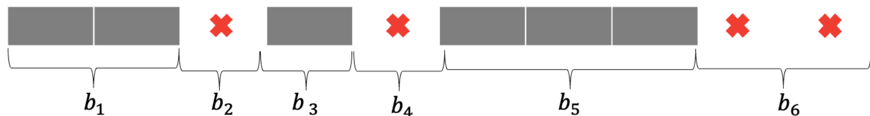
cpge-paradise.com

- Les lemmes et théorèmes utilisées
- La Problématique
- Solution proposée
- Application à travers python ou matlab

cpge-paradise.com

Définition

une séquence $(b_1, ..., b_p)$ signifie qu'on remplit p bloc de $b_1, ..., b_p$ cellules tel que : (i) . b_i est remplit $\Rightarrow$ b_{i+1} ne doit pas être remplit
(ii). il n'existe aucune case vide entre avant b_1 et après b_p



Lemme 2.1(admis)

Sps qu'un nonogram est crée aléatoirement. Ainsi, pour chaque ligne, le nombre de cellule attendu est $\frac{n}{2}$
le nombre de bloc de cellule est $\frac{n+1}{4}$

Les lemmes et théorèmes utilisées

cpge-paradise.com

lemme 2.2

il y'a $\binom{n+p-1}{p-1}$ solution dy système :

$$\begin{cases} a_1 + ... + a_p = n \\ a_1, ..., a_p \geq 0 \end{cases}$$

Theorem 2.3

Notons $(b_1, ..., b_p)$ une séquence donnée pour une ligne et notons $r = b_1 + ... + b_p$ ainsi il y'a $\binom{n-r+1}{p}$ configuration qui satisfait la séquence.

cpge-paradise.com

Pour un nonogram de taille n, en tenant compte de toutes les lignes on a donc $\prod_{i=1}^{m} \binom{n-r_i+1}{p_i}$ configurations, soit environ $s = \binom{\lfloor \frac{n}{2} \rfloor + 1}{\lfloor \frac{n}{4} \rfloor}^m$.

n	2^{mn}	$\binom{\frac{n}{2}+1}{\frac{n}{4}}^m$
5	3.355×10^7	243
10	1.267×10^{30}	5.766×10^{11}
15	5.391×10^{67}	1.670×10^{26}
20	2.582×10^{120}	1.962×10^{53}

cpge-paradise.com

Problématique

- $\vec{x} = [110, 001, 111, 101]$

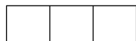
Plusieurs remarques se présentent , l'une d'elles est :

- Appartion des équations non linéaires dont la gravité augmente quand la taille du puzzle augmente à titre d'exemple : $(x_1.x_2 = 1 ou\, x_2.x_3 = 1)$, Pour (1,1) ou (1,2)

	1 2	1 1	3
2	■	■	
1			■
3	■	■	■
1 1	■		■

cpge-paradise.com

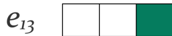
Pour n = 3



$\frac{n(n+1)}{2} = 6$ manière

e_{11}

e_{12}

e_{13} 

e_{21}

e_{22}

e_{31}

$(e_{11}, ..., e_{13})$ forment une base

$= 1.e_{21}$
$= 1.e_{11} + 1.e_{12}$
(FAUX)

$= 1.e_{11} + 1.e_{13}$

Si on regroupe les e_{ikj} dans notre $\vec{x} \in (0, 1)^N$

avec :
$N = \frac{mn(n+1)}{2}$

$$\begin{cases} i = 1, ..., m \\ k = 1, ..., n \\ j = 1, ..., n - k + 1 \end{cases}$$

cpge-paradise.com

on aurai donc :
$\vec{x} = [e_{111}, e_{112}, ..., e_{431}] = [000100, 001000, 000001, 101000]$
Notre vecteur est plus dense que celui auparavant
d'après lemme 2.1 chaque ligne contient $\frac{n+1}{4}$ ainsi $E(s) = \frac{m(n+1)}{4}$ au lien de $\frac{m(n+1)}{2}$

cpge-paradise.com

Trouver la solution sous forme de résolution d'une équation linéaire : $A\vec{x} = \vec{b}$ **et une inégalité :** $B\vec{x} \leq \vec{c}$

Remarque à faire : En ce qui concerne l'inégalité nous allons comparer les termes(afin de ne pas alourdir l'écriture nous nous contenterons de l'écriture précédente).

$Pour\ A \in M_{M1,N}(R)\ et\ B \in M_{M2,N}(R)\ \vec{c} \in R^{M2} et\ \vec{b} \in R^{M1}$

cpge-paradise.com

Contrainte toujours admise

Le niveau de dispersion :

$$\boxed{\sum_{i=1}^{N} x_i = \sum_{i=1}^{m} r_i \text{ donc } A_1.\vec{x} = \vec{b}_1}$$

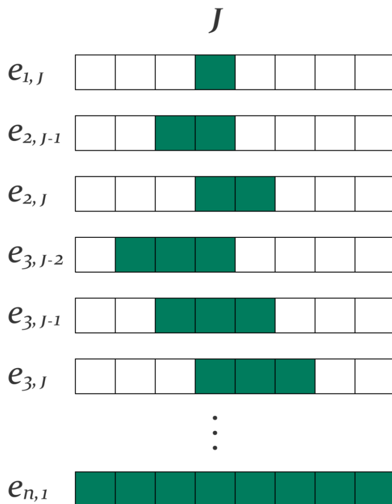
avec r_i la somme des cellules remplises dans la i-ème ligne et $A_1 \in \mathrm{M}_{1,N}(R)$ dont les entrées sont que des 1 et $\vec{b}_1$ avec une seule entrée

Première contrainte

cpge-paradise.com

Mise en situation :

- on prend une séquence $(b_1, ..., b_p)$ suivant les colonnes
  Et r $>$ 0 tel que :
  $r = b_1 + ... + b_p$



Première contrainte

cpge-paradise.com

La J-ème colonne satisfait

$$\boxed{\sum_{i=1}^{m}\sum_{k=1}^{n}\sum_{j=j_{l_0}}^{j_{h_i}} e_{ikj} = r \text{ ou } k = 1, ..., n; j = j_{l_0}, ..., j_{h_i}}$$

$$\text{tel que : } \begin{cases} j_{l_0} = max(J-k+1, 1) \\ j_{h_i} = min(n-k+1, J) \end{cases}$$

Sous sa forme d'équation matricielle $A_2.\vec{x} = \vec{b_2}$ ou $A_2 \in M_{n,N}(R)$

$$A_2 = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

$$\vec{b}_2 = \begin{bmatrix} 3 & 2 & 3 \end{bmatrix}^T.$$

Deuxième contrainte (Nombre de blocs)

cpge-paradise.com

Soit la l-ième ligne ou $l \in [\![1, m]\!]$ et t_K le nombre de bloc de taille K dans une seule ligne , $\boxed{\sum_{j=1}^{j_{h_i}} e_{lKj} = t_K}$ pour $K = 1, .., n$ et $j_{h_i} = n - K + 1$, on a alors :

- $A_3.\vec{x} = \vec{b}_3$ avec A_3 une matrice de taille $mn\,x\,N$

$$A_3 = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc}
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
\hline
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right]$$

$$\vec{b}_3 = \left[\begin{array}{ccc|ccc|ccc|ccc} 0&1&0&1&0&0&0&0&1&2&0&0 \end{array}\right]^T.$$

Les inégalités (Problème)

cpge-paradise.com

-Pour le même exemple, $e_{11} = 1$ et $e_{12} = 1$ (Règle) ; $e_{11} + e_{12} \leq 1$

$$\begin{array}{llll} e_{11}+e_{12} \leq 1 & e_{11}+e_{21} \leq 1 & e_{11}+e_{22} \leq 1 & e_{11}+e_{31} \leq 1 \\ e_{12}+e_{13} \leq 1 & e_{12}+e_{21} \leq 1 & e_{12}+e_{22} \leq 1 & e_{12}+e_{31} \leq 1 \\ e_{13}+e_{21} \leq 1 & e_{13}+e_{22} \leq 1 & e_{13}+e_{31} \leq 1 & \\ e_{21}+e_{22} \leq 1 & e_{21}+e_{31} \leq 1 & & \\ e_{22}+e_{31} \leq 1. & & & \end{array}$$

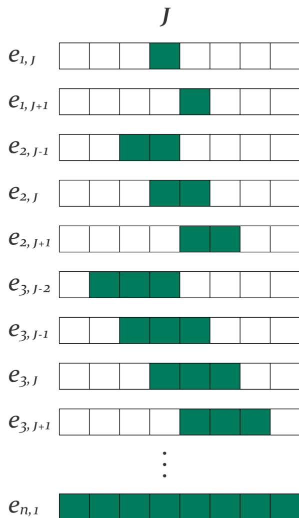
56 inégalités pour 24 variables.

Pus généralement pour un hanjie de taille n x m on a :

$$m \times \frac{1}{12}.(n^4 + 4n^3 - n^2 - 4n) \approx \frac{mn^4}{12}$$

Première solution, si la suite ne se répète pas

cpge-paradise.com



les e_{lkj} vérifient

$$\boxed{\sum_{k=1}^{n} \sum_{j=j_{l_0}}^{j_{h_i}} e_{lkj} \leq 1}$$ avec :

$$\begin{cases} j_{l_0} = max(J-k+1,1) \\ j_{h_i} = min(n-k+1, J+1) \end{cases}$$

$\rightarrow B_1.\vec{x} \leq \vec{c_1}$

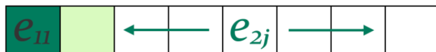
Ou $B_1 \in M_{mn,N}(R)$

Deuxième solution(si un vecteur de la solution se repète)

cpge-paradise.com

$j_{min} = 1 \leq j \leq j_{max} = 5$

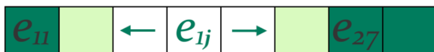
on a $j_{max} = n - 2 + 1 = 7$ et $j_{min} = 3$

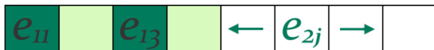


À l'aide de ces deux propositions on a l'inégalité suivante :
$0 < 1e_{11} + 2e_{12} + 3e_{13} + 4e_{14} + 5e_{15} < 3e_{23} + 4e_{24} + 5e_{25} + 6e_{26} + 7e_{27}$
Vérifée par toute séquence dont la taille de bloc se repète une fois.

cpge-paradise.com

Considérons la séquence (1,1,2) :





À l'aide de l'image on a :

$$\begin{cases} 1 \leq e_{11} + e_{12} + e_{13} \leq 2 \\ 1 \leq e_{13} + e_{14} + e_{15} \leq 2 \\ 1 \leq e_{25} + e_{26} + e_{26} \leq 1 \end{cases}$$

on trouve notre B_2 qui dépend des séquences de lignes tel que: $\boxed{\mathrm{B}_2.\vec{x} \leq \vec{c}_2}$

Conclusion

cpge-paradise.com

En combinant nos A_i ie :

$$A = \begin{bmatrix} A_1 \\ \hline A_2 \\ \hline A_3 \end{bmatrix} \tag{1}$$

de même pour $\vec{b_i}$:

$$\vec{b} = \begin{bmatrix} \vec{b}_1 \\ \hline \vec{b}_2 \\ \hline \vec{b}_3 \end{bmatrix} \tag{2}$$

avec $A \in \mathrm{R}^{M_1 x N}$ et $\vec{b} \in \mathrm{R}^{M_1}$ on trouve que $M_1 = 1 + n + mn$

cpge-paradise.com

Avec le même processus fait pour A on :

$$B = \left[\begin{array}{c} B_1 \\ \hline B_2 \end{array}\right] \tag{3}$$

De même pour :

$$\vec{c} = \left[\begin{array}{c} \vec{c_1} \\ \hline \vec{c_2} \end{array}\right] \tag{4}$$

tel que $B \in \mathrm{R}^{M_2 x N}$ et $\vec{c} \in \mathrm{R}^{M_2}$
Alors on a $mn \leq M_2 \leq mn + 2s$ ie M_2 inéquations.

Résultat

cpge-paradise.com

cpge-paradise.com

MERCI POUR VOTRE ATTENTION !!!

cpge-paradise.com

```python
import numpy as np
from scipy.optimize import linprog
from itertools import product
import time

t1 = time.time()
def validate_row(row, clues):
    # Valide si une ligne correspond aux indices donnés
    runs = []
    count = 0
    for cell in row:
        if cell:
            count += 1
        else:
            if count > 0:
                runs.append(count)
                count = 0
    if count > 0:
        runs.append(count)
    return runs == clues

def generate_rows(n, clues):
    # Génère toutes les lignes possibles pour une taille et des indices donnés
    for combination in product([0, 1], repeat=n):
        if validate_row(combination, clues):
            yield combination

def transpose(grid):
    # Transpose une grille
    return list(map(list, zip(*grid)))

def solve_nonogram(row_clues, col_clues):
    n_rows, n_cols = len(row_clues), len(col_clues)
    row_possibilities = [list(generate_rows(n_cols, clue)) for clue in row_clues]
    col_possibilities = [list(generate_rows(n_rows, clue)) for clue in col_clues]

    def is_valid_grid(grid):
        for j in range(n_cols):
            col = [grid[i][j] for i in range(n_rows)]
            if not validate_row(col, col_clues[j]):
                return False
        return True

    def solve(grid, row=0):
        if row == n_rows:
            if is_valid_grid(grid):
                solutions.append([row[:] for row in grid])
            return

        for possibility in row_possibilities[row]:
            grid[row] = possibility
            solve(grid, row + 1)

    solutions = []
    initial_grid = [[0] * n_cols for _ in range(n_rows)]
    solve(initial_grid)
    return solutions

def print_solutions(solutions):
    if not solutions:
        print("Aucune solution trouvée.")
        return

    print("Solutions trouvées:")
    for idx, solution in enumerate(solutions, 1):
        print(f"Solution {idx}:")
        for row in solution:
```

cpge-paradise.com

```python
            print("".join('#' if cell else '.' for cell in row))
        print()

# Exemple d'utilisation
row_clues = [[6], [1], [1,1], [1],[1,1],[1,1],[1,1,1],[2,1]]
col_clues = [[1], [1], [8],[1] ,[1],[1,1,4],[1],[1]]
solutions = solve_nonogram(row_clues, col_clues)
t2 = time.time()
# Affichage des solutions

print(t2-t1)
print_solutions(solutions)


def create_constraints(row_sequences, column_sequences, m, n, s):
    # Les paramètres du problème
    table_globalIndices = local_to_global(m, n)
    N = m * n * (n + 1) / 2

    # Determiner combien d'entrée non nul pour les matrices A,B,b et c
    numEntries_in_A = 0
    numEntries_in_B = 0
    numEntries_in_b = 0
    numEntries_in_c = 0

    # L'équation linéaire de la première contrainte le nombre  d'entrée
    numEntries_in_A += N
    numEntries_in_b += 1

    # 2ème contrainte le nombre d'entrée
    countEntries = 0
    for J in range(n):
        for k in range(n):
            j_lo = max(J - k + 1, 1)
            j_hi = min(n - k + 1, J)
            countEntries += (j_hi - j_lo + 1)
    numEntries_in_A += m * countEntries
    numEntries_in_b += n


    # 3ème contrainte le nombre d'entrée
    numEntries_in_A += N
    numEntries_in_b += m * n

    # Linear inequality constraints in Section 4d
    countEntries = 0
    for J in range(n):
        for k in range(n):
            j_lo = max(J - k + 1, 1)
            j_hi = min(n - k + 1, J + 1)
            countEntries += (j_hi - j_lo + 1)
    numEntries_in_B += m * countEntries
    numEntries_in_c += m * n

    # Le nombre d'entrée pour les inéquations
    countEntries = 0
    countInequalities = 0
    for I in range(m):
        sequence = row_sequences[I]
        p = len(sequence)
        if p == 1:
            continue
        elif len(set(sequence)) == p:
            for L in range(p - 1):
                # For the L-th block
                sequence_left = sequence[:L]
                k = sequence[L]
```

cpge-paradise.com

```
                sequence_right = sequence[L + 1:]
                numCellsTaken_left = sum(sequence_left) + len(sequence_left)
                numCellsTaken_right = sum(sequence_right) + len(sequence_right)
                j_lo = numCellsTaken_left + 1
                j_hi = n - numCellsTaken_right - k + 1
                countEntries += 2 * (j_hi - j_lo + 1)
                # For the (L + 1)-th block
                sequence_left = sequence[:L]
                k = sequence[L + 1]
                sequence_right = sequence[L + 2:]
                numCellsTaken_left = sum(sequence_left) + len(sequence_left)
                numCellsTaken_right = sum(sequence_right) + len(sequence_right)
                j_lo = numCellsTaken_left + 1
                j_hi = n - numCellsTaken_right - k + 1
                countEntries += j_hi - j_lo + 1
            countInequalities += 2 * (p - 1)
        else:
            for L in range(p):
                sequence_left = sequence[:L]
                k = sequence[L]
                sequence_right = sequence[L + 1:]
                numCellsTaken_left = sum(sequence_left) + len(sequence_left)
                numCellsTaken_right = sum(sequence_right) + len(sequence_right)
                j_lo = numCellsTaken_left + 1
                j_hi = n - numCellsTaken_right - k + 1
                countEntries += 2 * (j_hi - j_lo + 1)
            countInequalities += 2 * p
    numEntries_in_B += countEntries
    numEntries_in_c += countInequalities

    # Initialiser les lignes, colonnes et arrays values
    A_row = np.zeros(numEntries_in_A, dtype=int)
    A_col = A_row.copy()
    A_val = A_row.copy()
    B_row = np.zeros(numEntries_in_B, dtype=int)
    B_col = B_row.copy()
    B_val = B_row.copy()
    b = np.zeros(numEntries_in_b)
    c = np.zeros(numEntries_in_c)

    # Construire les matrices A , B , b et c
    index_entry_A = 0
    index_row_A = 0
    for globalIndex in range(N):
        A_row[index_entry_A] = index_row_A
        A_col[index_entry_A] = globalIndex
        A_val[index_entry_A] = 1
        index_entry_A += 1
    b[index_row_A] = s
    index_row_A += 1

    for J in range(n):
        for i in range(m):
            for k in range(n):
                j_lo = max(J - k + 1, 1)
                j_hi = min(n - k + 1, J)
                for j in range(j_lo, j_hi + 1):
                    A_row[index_entry_A] = index_row_A
                    A_col[index_entry_A] = table_globalIndices[k, j, i]
                    A_val[index_entry_A] = 1
                    index_entry_A += 1
        b[index_row_A] = sum(column_sequences[J])
        index_row_A += 1

    for I in range(m):
        sequence = row_sequences[I]
        for K in range(n):
```

cpge-paradise.com

```
            j_hi = n - K + 1
            for j in range(1, j_hi + 1):
                A_row[index_entry_A] = index_row_A
                A_col[index_entry_A] = table_globalIndices[K, j, I]
                A_val[index_entry_A] = 1
                index_entry_A += 1
            b[index_row_A] = sum(row_sequences[I] == K)
            index_row_A += 1

    index_entry_B = 0
    index_row_B = 0
    for I in range(m):
        sequence = row_sequences[I]
        p = len(sequence)
        if p == 1:
            continue
        elif len(set(sequence)) == p:
            for L in range(p - 1):
                sequence_left = sequence[:L]
                k = sequence[L]
                sequence_right = sequence[L + 1:]
                numCellsTaken_left = sum(sequence_left) + len(sequence_left)
                numCellsTaken_right = sum(sequence_right) + len(sequence_right)
                j_lo = numCellsTaken_left + 1
                j_hi = n - numCellsTaken_right - k + 1
                for j in range(j_lo, j_hi + 1):
                    B_row[index_entry_B] = index_row_B
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = -j
                    index_entry_B += 1
                    B_row[index_entry_B] = index_row_B + 1
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = j
                    index_entry_B += 1
                c[index_row_B] = -1
                c[index_row_B + 1] = -1
                index_row_B += 2
        else:
            for L in range(p):
                k = sequence[L]
                numCellsTaken_left = sum(sequence[:L]) + L
                numCellsTaken_right = sum(sequence[L + 1:]) + p - L - 1
                j_lo = numCellsTaken_left + 1
                j_hi = n - numCellsTaken_right - k + 1
                for j in range(j_lo, j_hi + 1):
                    B_row[index_entry_B] = index_row_B
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = -1
                    index_entry_B += 1
                    B_row[index_entry_B] = index_row_B + 1
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = 1
                    index_entry_B += 1
                c[index_row_B] = -1
                index_row_B += 1
            for L in range(p):
                k = sequence[L]
                j_lo = sum(sequence[:L]) + L + 1
                j_hi = n - sum(sequence[L + 1:]) - p + L + 1
                for j in range(j_lo, j_hi + 1):
                    B_row[index_entry_B] = index_row_B
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = -1
                    index_entry_B += 1
                    B_row[index_entry_B] = index_row_B + 1
                    B_col[index_entry_B] = table_globalIndices[k, j, I]
                    B_val[index_entry_B] = 1
```

cpge-paradise.com

```python
                    index_entry_B += 1
                c[index_row_B] = 1
                index_row_B += 1

    numEquations = numEntries_in_b
    numInequalities = numEntries_in_c

    return A_row, A_col, A_val, b, c, numEquations, numInequalities

def local_to_global(m, n):
    table_globalIndices = np.zeros((n, n, m), dtype=int)
    for I in range(m):
        for J in range(n):
            for K in range(n):
                table_globalIndices[K, J, I] = I * n * n + J * n + K
    return table_globalIndices

def solve_optimization_problem(A_row, A_col, A_val, b, B_row, B_col, B_val, c):
    # Convert the sparse matrices to CSR format
    A = sparse.csr_matrix((A_val, A_col, A_row), shape=(len(b), len(c)))
    B = sparse.csr_matrix((B_val, B_col, B_row), shape=(len(c), len(c)))

    # Définier la fonction objet
    c = np.concatenate((c, np.zeros(len(b))))

    # Definir les contraintes
    A_eq = A
    b_eq = b
    A_ineq = B
    b_ineq = c

    # Résoudre le problème optimisé utilisant linprog
    res = linprog(c, A_eq=A_eq, b_eq=b_eq, A_ub=A_ineq, b_ub=b_ineq, method='highs')

    # Extraire les solutions
    x = res.x[:len(c)]

    return x
```