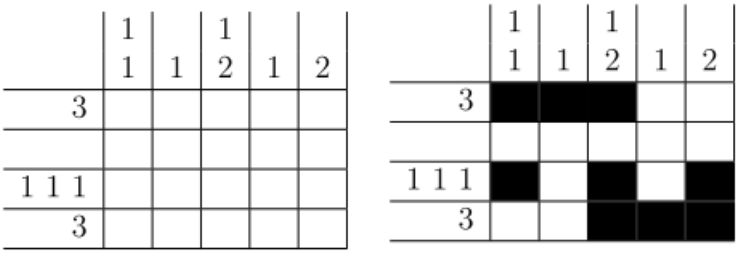
SORBONNE UNIVERSITE FACULTE DES SCIENCES
UFR D'INFORMATIQUE LICENCE 3

PROJET LU3IN003

Un problème de tomographie discrète



Réalisé par :
Louiza AOUAOUCHE 3802084
Ines YAICI 3802715

Encadrants :
Olivier SPANJAARD - Fanny PASCUAL

Parcous Bi-disciplinaire intensif IM 2020 - 2021

Table des matières

Avant toute chose voici un lien vers un notebook contenant notre code testé :
https://colab.research.google.com/drive/1k_413eCoqy1pbySRmSZTY4ppFgqcYWkY?usp=sharing

1 Introduction

Dans le cadre de l'Unité d'Enseignement LU3IN003 intitulée Algorithmique Avancé, nous sommes chargés d'étudier différents problèmes et leur apporter une solution par une approche algorithmique. Le but ultime est de définir l'algorithme et les structures de données les plus optimales en termes de complexité qui permettent de résoudre efficacement la problématique donnée.
Le projet sur lequel nous avons travaillé traite d'un problème de tomographie discrète où le but est de construire, s'il en existe, une coloration d'une grille répondant à différentes contraintes.

2 Présentation et notation

Ici, nous reprenons certaines notation et définitions du sujet :

Nous considérons une grille de N lignes numérotées de 0 à $N-1$ et M colonnes numérotées de 0 à $M-1$. Chacune des $N*M$ cases doit être coloriée en blanc ou en noir. A chaque ligne l_i $(i = 0, \cdots, N-1)$, est associée une séquence d'entiers représentant les longueurs des blocs de cases noires de la ligne. De même,à chaque colonne c_j $(j = 0, \cdots, M-1)$ est associée une séquence d'entiers représentant les longueurs des blocs de cases noires de la colonne.
Le but du projet est de construire, s'il en existe, une solution (un coloriage noir-blanc des cases) répondant aux contraintes.
Plus généralement, chaque ligne/colonne se voit donc assigner une séquence (potentiellement vide) $(s_1, s_2, \cdots, s_k)$ de nombres entiers strictement positifs. Le coloriage doit comporter sur cette ligne/colonne un premier bloc de s_1 cases noires consécutives, puis un deuxième de s_2 cases noires... Il n'y a pas d'autres cases noires dans la ligne/colonne. Les blocs doivent être séparés d'au moins une case blanche. Il peut y avoir des cases blanches avant le premier bloc, et/ou après le dernier.

3 Etude du problème de tomographie discrète

3.1 Méthode incomplète de résolution

Dans un premier temps, on étudie la coloration partielle des grilles, on remarquera que certaines cases sont plus simples à colorier.

3.1.1 Première étape

Considérons une ligne l_i munie de la séquence $(s_1, s_2, \cdots, s_k)$. Nous allons définir un algorithme permettant de déterminer s'il existe un coloriage possible de la ligne avec cette séquence. On définit pour cela $T(j,l)$ qui vaut $True$ s'il est possible de colorier les $j+1$ premières cases de l_i avec la sous séquence $(s_1, s_2, \cdots, s_l)$

(Q1) Après le calcul de tous les $T(j,l)$ on peut savoir s'il est possible de colorier la ligne l_i entière avec la séquence entière. En effet, soit $l \in 0, \cdots, k$:

- s'il existe $j \in 0, \cdots, M$ tel que $T(j,l) = True$ alors les $j+1$ premières cases peuvent être coloriées par la sous séquence $(s_1, s_2, \cdots, s_l)$.
- s'il existe $j \in 0, \cdots, M$ tel que $T(j,l) = True$ alors les $j+1$ premières cases peuvent être coloriées par la séquence entière. On peut ainsi colorier les cases de $j+2$ à $M-1$ en blanc. Finalement, on aura colorié la ligne entièrement.
  Par l'absurde :
  Supposons qu'il n'existe pas de $j \in 0, \cdots, M$ tel que $T(j,l)$ est vrai pour un $l \in 0, \cdots, k$.
  Donc on peut en déduire qu'il n'est pas possible de colorier les $M-1$ premières cases $(i,0), (i,1), \cdots, (i, M-1)$ avec la sous séquence $(s_1, s_2, \cdots, s_l)$ pour $l \in 0, \cdots, k$.
  Si un coloriage est possible de cette ligne avec la séquence complète $(s_1, \cdots, s_l, \cdots, s_k)$ alors on peut trouver un $j_0 \in 0, \cdots, M$ tel que on peut colorier les j_0+1 premières cases avec la séquence complète par construction il existe forcément un $j_0' \in 0, \cdots, j_0$ tel que on peut colorier les $j_0'+1$ première cases avec la séquence $(s_1, s_2, \cdots, s_l)$.
  Contradiction ■

Remarque : Dans cette première étape nous pouvons résoudre le problème de manière plus simple sans recourir à la récurrence et sans l'utilisation du tableau des $T(j,l)$
Donc en étant sur une ligne i ayant M colonnes pour savoir si on peut la colorier avec une séquence $(s_1, s_l, \cdots, s_k)$ il suffit que $\sum_{l=1}^{k} s_l + k - 1 \leq M$ (càd : la somme du nombre de cases noires et du nombre de cases blanches entre chaque bloc noir est inférieur au nombre de colonnes).

Complexité : Dans le pire cas $O(M)$ (car dès qu'on dépasse M on peut arrêter car on sait déjà que la coloration n'est pas possible).

(Q2) Etude des cas 1,2a et 2b :

- Cas 1 : $l = 0$
  Dans ce cas la séquence est vide elle ne contient aucun bloc. On peut alors colorier toute la ligne en blanc. Ainsi pour tout $j \in 0. \cdots, M$ $T(j,0) = True$
- Cas 2 : $l \geq 1$
  Dans ce cas il existe au moins un bloc dans la séquence
  - Cas 2.a : $j < s_l - 1$ (i.e $j+1 < s_l$)
    Le nombre des $j+1$ premières cases est strictement inférieur à la taille du bloc s_l. Il est donc impossible de les colorier avec la sous séquence $(s_1, s_2, \cdots, s_l)$. Donc $T(j,l) = False$
  - Cas 2.b : $j = s_l - 1$ (i.e $j+1 = s_l$)
    Le nombre des $j+1$ cases est exactement égal à la taille du bloc s_l.
    - Si $l = 1$: On peut colorier les $j+1$ cases en noir avec le bloc s_1. Ainsi $T(j,l) = True$

❍ Si $l > 1$:
. Si $\sum_{k=1}^{l} s_k > j+1$, les $j+1$ cases ne suffisent même pas à stocker toute la séquence sans séparateur
. Si $\sum_{k=1}^{l} s_k = j+1$, les $j+1$ premières cases peuvent être coloriées totalement en noir mais il manquera au moins un séparateur blanc entre chaque bloc.
Dans les deux cas on a $T(j,l) = False$

(Q3) Etude du cas 2.c :

- Cas 2 : $l \geq 1$
  - Cas 2.c : $j > s_l - 1$
    - ❍ Si la case précédente $(i, j-1)$ est coloriable alors (i,j) est coloriable en blanc donc :
    $$T(j-1,l) = True \Rightarrow T(j,l) = True$$
    - ❍ Sinon : la dernière case coloriée dans par le bloc précédent est exactement à la position $(i, j-s_l-1)$. A partir de cette case (qui constitue la fin du bloc $l-1$) jusqu'à la case (i,j) il faut qu'on puisse insérer un séparateur blanc et toute la séquence s_l. Donc la case (i,j) serait la dernière case noire du bloc l. D'où :
    $$T(j-s_l-1, l-1) = True \Rightarrow T(j,l) = True$$
    - ❍ Comme on étudie ces deux seuls cas (coloriage en noir ou en blanc) on conclue :
    $$T(j,l) = T(j-s_l-1, l-1) \text{ ou } T(j-1,l)$$

ALGORITHME 1

```
1 :  Fonction T(j,s,l) :
2 :  Si l = 0 : (cas 1)
3 :       Retourner Vrai
4 :  Si l ≥ 1 : (cas 2)
5 :       Si j < s[l] − 1 :   (cas 2a)
6 :            Retourner Faux
7 :       Si j = s[l] − 1 :   (cas 2b)
8 :            Si l = 1 :
9 :                 Retourner Vrai
10 :           Sinon :
11 :                Retourner Faux
12 :      Si j > s[l] − 1 :   (cas 2c)
13 :           Retourner T(j − 1, l) ou T(j − s[l] − 1, l − 1)
```

(Q4) Code de l'algorithme : *voir le fichier ".py" joint avec ce rapport ou directement voir le lien :*
https://colab.research.google.com/drive/1k_413eCoqy1pbySRmSZTY4ppFgqcYWkY?usp=sharing

Signature de la fonction :

```
def T(t :list,s :list,j :int,l :int):
```

Justification des structures utilisées : Ici ainsi que dans tout le code nous retrouverons un aspect de programmation dynamique.
En effet, lors du calcul des $T(j,l)$ on remarquera que dans la fonction récursive on décompose le problème en sous problèmes plus simples (des T(j,l) qui se rapprochent de plus en plus de nos cas de base), et ces sous problèmes peuvent être plusieurs fois les mêmes donc on stocke les résultats intermédiares qui sont déjà calculés par d'autres appels récursifs (optimisation) .

t : tableau 2D de booléens qui stocke les valeurs des$T(j,l)$ pour une ligne donnée. Le but de cette structure est d'éviter les appels récursifs répétitifs et ainsi réduire massivement la complexité de l'algorithme par la mémorisation des données préalablement calculées.

s : tableau 1D qui contient la séquence d'une ligne donnée.

Le choix des tableaux comme structure est lié à la facilité d'accès aux données et leur modification en $O(1)$.

Analyse et commentaires :

— Complexité :
. Si le nombre d'éléments d'une séquence $(s_1, s_2, \cdots, s_k)$ d'une ligne donnée dépasse M ($k \geq M$), la coloration n'est pas possible (la vérification se fait en $O(1)$).
. Sinon, pour tout $j = 0, \cdots, M-1$ et pour tout $l = 0, \cdots, k \leq M$, on peut calculer tous les $T(j,l)$. Ce qui se fait dans le pire des cas avec une complexité en $O(M^2)$.

3.1.2 Généralisation

(Q5) Modification de l'algorithme précédent pour qu'il prenne en compte les cases déjà coloriées :

- Cas 1 : $l = 0$
  Il faut vérfier en plus que toutes les cases ne sont pas noires.
- Cas 2 : $l \geq 1$
  - Cas 2.a : $j < s_l - 1$ (i.e $j + 1 < s_l$)
    Similaire à l'algorithme précédent.
  - Cas 2.b : $j = s_l - 1$ (i.e $j + 1 = s_l$)
    Le nombre des $j + 1$ cases est exactement égal à la taille du bloc s_l.
    - Si $l = 1$:
      $T(j,l) = True$ //A condition qu'il n'y ait pas de cases blanches avant sur la ligne.
    - Si $l > 1$:
      Similaire à l'algorithme précédent.
  - Cas 2.c : $j > s_l - 1$
    - Si la case (i,j) est blanche :
      il faut que la séquence rentre dans les cases d'avant (on fait l'appel récursif sur $T(j-1,l)$).
    - Si la case (i,j) est noire :
      Il faut qu'il y'ait au moins une case de séparation (blanche ou vide) avec le bloc précedent.
      On vérifie qu'aucune case devant être coloriée en noir par le bloc n'est déjà coloriée en blanc.
      On vérifie si le bloc d'avant rentre dans les ($j-$la longueur de la séquence) premières cases (on fait un appel récursif sur $T(j - s_l - 1, l-1)$).
    - Si la case (i,j) n'est pas coloriée :
      Il faut qu'il y'ait au moins une case de séparation (blanche ou vide) avec le bloc précédent.
      On vérifie qu'aucune case devant être coloriée en noir par le bloc n'est déjà coloriée en blanc.
      Cette case peut alors être blanche ou noire. Pour déterminer cela, on fait l'appel récursif sur $T(j-1,l)$ ou $T(j - s_l - 1, l-1)$.
      Si les conditions énoncées ne sont pas satisfaites la case est blanche.

(Q6) Analyse de la complexité :
Pour savoir si la coloration d'une ligne i est possible avec une séquence $(s_1, s_2, \cdots, s_k)$, il suffit de lancer l'appel récursif sur un seul $T(j,l)$ qui est $T(M-1,k)$.
Calcul de la complexité pour un $T(j,l)$ donné : *(pire cas)*

— Cas 1 : $O(j)$

— Cas 2.a : $O(1)$

— Cas 2.b : $O(j)$

— Cas 2.c : si la cases (i,j) est blanche ou noir $O(j)$ sinon $O(j^2)$

Ce qui fait que la résolution du $T(M-1,k)$ (dans le pire des cas) est en $O(M^2)$: une complexité polynomiale quadratique.

(Q7) *voir le fichier ".py" joint avec ce rapport ou directement voir le lien :*
https://colab.research.google.com/drive/1k_413eCoqy1pbySRmSZTY4ppFgqcYWkY?usp=sharing

Signature de la foncion codée et description : $T\_ge(t : list, Li : list, j : int, s : list, l : int)$,Retourne True s'il est possible de colorier les cases de 0 à j de la ligne Li avec la séquence "s" en prenant compte des cases déjà coloriées, les informations sont stockées au fur et à mesure dans 't'.

3.1.3 Propagation

(Q8) Complexité de l'algorithme :*Il est conseillé d'avoir le code sous les yeux*

— Fonction ColoreLig $O(M^2)$:
Création du tableau de mémorisation t : en $O(M^2)$
Appel de la fonction de la séction 1.2 pour calculer le $t[M-1][k]$, avec k le nombre de blocs de la séquence correspondante à une ligne donnée : en $O(M^2)$

— Fonction ColoreCol $O(N^2)$:
même principe que ColoreLig.

— Dans Coloration :
Pour la coloration des Lignes :
La taille de l'ensemble LignesAVoir est bornée par le nombre de ligne (N),donc la boucle *"for i in LignesAVoir"* en $O(N)$.
"for j in range(M) :" en $O(M)$.
Ce qui nous donne du $O(N * M^3)$
Pour la coloration des Colonnes :
Un même raisonnement nous donne du $O(M * N^3)$.

Conclusion : la complexité totale de la partie propagation est en $O(N * M^3 + M * N^3)$ qui est bien ce qui est attendu (complexité polynomiale).

(Q9) *voir le fichier ".py" joint avec ce rapport ou directement voir le lien :*
https://colab.research.google.com/drive/1k_413eCoqy1pbySRmSZTY4ppFgqcYWkY?usp=sharing

Signatures et descriptions des fonctions :

$ColoreLig(grille : list, s : list, len_s : int, i : int, taille : int)$: Renvoie True si la ligne i de la grille (Li) peut être coloriée par la séquence s et tel que taille représente la taille de Li (le nombre de colonnes) et len_s le nombre de blocs dans la séquence.

$ColoreLig(grille : list, s : list, len_s : int, i : int, taille : int)$: Même principe que la fonction précédente mais cette fois-ci sur les colonnes.

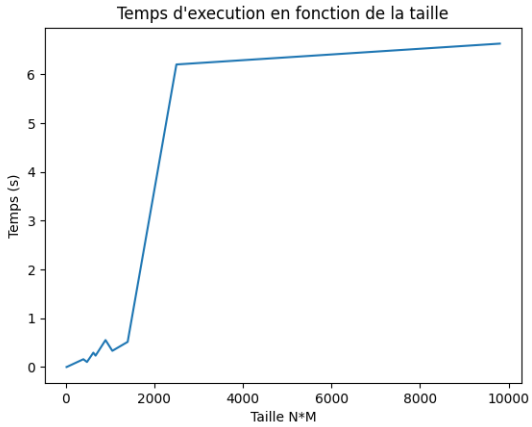
$Coloration(fichier)$: fonction principale qui, à partir d'un fichier contenant les séquences des lignes et des colonnes, revoie l'instance coloriée s'il y'a une possibilité de colorier.

3.1.4 Tests

(Q10)

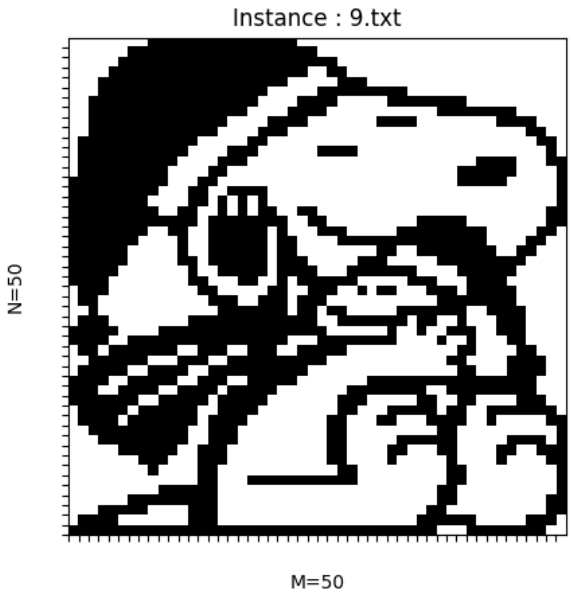
- Temps (en secondes) de résolution des instances de $0.txt$ à $10.txt$:
NB : Le temps de la lecture du fichier est compris

Instance	Nb Lignes	Nb Colonnes	Temps
0.txt	4	5	0.0
1.txt	5	5	0.000995635986328125
2.txt	20	20	0.15957307815551758
3.txt	13	37	0.10273408889770508
4.txt	25	25	0.2942185401916504
5.txt	25	27	0.23337149620056152
6.txt	30	30	0.5525193214416504
7.txt	31	34	0.332108736038208
8.txt	40	35	0.5155787467956543
9.txt	50	50	6.205426216125488
10.txt	99	99	6.63320517539978

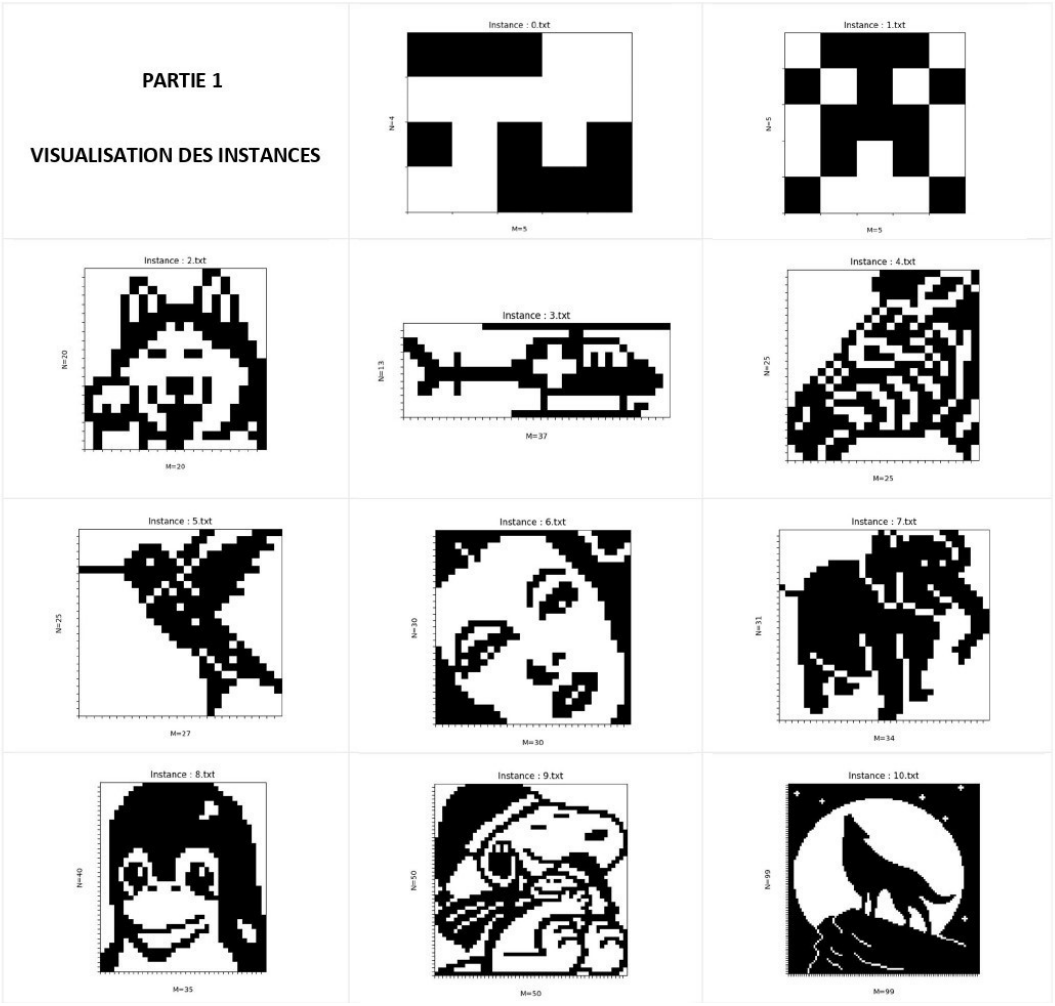


A partir des temps d'execution obtenus nous avons abouti à plusieurs observations : Le temps d'exécution croit en fonction de la taille de l'instance ce qui est normal et relatif au nombre d'essais de coloration. Ceci dit, on remarque un bond au niveau de l'instance $9.txt$ où on passe de $0.5s$ à $6.2s$. L'augmentation de la taille contribue à cette forte hausse mais un facteur important en est le principal responsable : le format des séquences. Dans l'instance $9.txt$, la taille des séquences est considérable, ce qui implique de nombreux tests avant de savoir si une ligne/une colonne peut être coloriée. C'est carrément relatif à doubler la taille puisqu'en passant de l'instance $9.txt$ de taille 50x50 à l'instance $10.txt$ de taille 99x99 le temps d'execution est quasi équivalent.

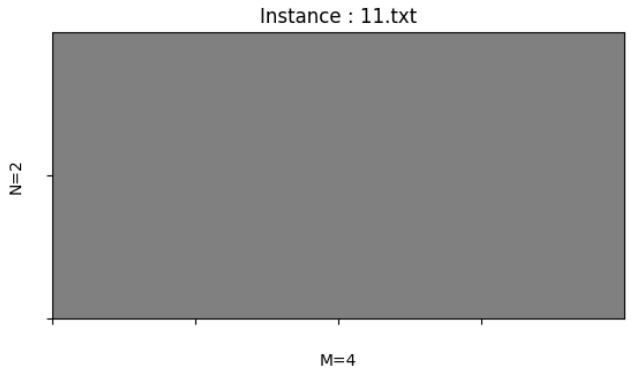
- Voici le résultat obtenu sur l'instance 9 (en 7 secondes environ) :



- Visualisation globale des grilles sur les instances $0.txt$ à $10.txt$:



(Q11) Résultat sur l'instance 11 :



Comme le montre la figure précédente, la Partie 1 ne permet pas de résoudre l'instance 11. La coloration n'est même pas partielle. En effet, lors des tests de coloration aucun test n'échoue. Ne pouvant déduire une coloration, l'algorithme renvoie donc une grille vide sans pour autant détecter une impossibilité de coloriage.

3.2 Méthode complète de résolution

Dans cette partie, nous allons programmer une méthode qui va venir compléter les grilles non/partiellement résolus par la première méthode. Pour cela, on effectue une énumération des possibilités en faisant le choix de colorier en blanc ou noir une case vide. On lance alors des appels récursifs sur ce choix qui vont propager la coloration sur les prochaines cases vides. On numérote les cases de la gauche vers la droite et de haut en bas, de 0 à $NM-1$.

3.2.1 Implantation et tests

Signature et description des fonctions :
$ColorierEtPropager(G : list, ii : int, jj : int, c : int, Lignes : list, Colonnes : list, Li\_Len : list, Col\_Len : list, N, M)$: équivalent de la fonction Coloration mais G est partiellement coloriée.
$EnumRec(G : list, k : int, c : int, Lignes : list, Li_Len : list, N : int, Colonnes : list, Col_Len : list, M : int)$: G : grille partiellement coloriée, k un indice de case, c une couleur
$Enumeration(F)$: Renvoie Vrai si la coloration complète sur l'instance F peut se faire , False sinon

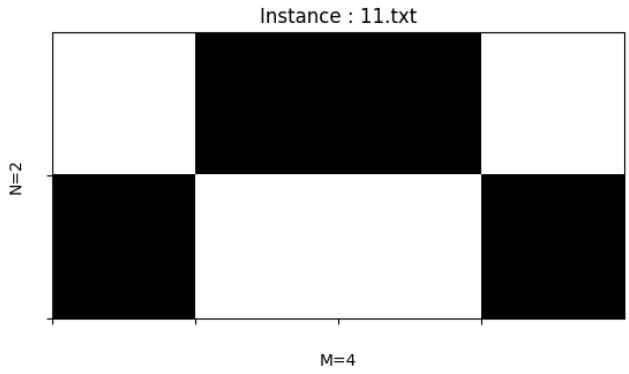
(Q12) Complexité de l'algorithme : *Il est conseillé d'avoir le code sous les yeux*
Pour trouver la complexité de cet algorithme on calcule la complexité de la fonction principale *"Enumeration"* :

- La fonction fait appel à Coloration de la partie 1.3 qui est en $O(N*M^3+M*N^3)$. Si cette dernière résoud complètement l'instance on s'arrête là.
  Sinon notre fonction fait appel à *Enum_rec"*.
- Pour la fonction *"Enum_rec"* :
  Soit k l'indice de numérotation des cases comme énoncé dans la description.
  Soit $T(k)$ la complexité temporelle en fonction de k.
  A chaque appel récursif on fait appel à la fonction *"ColorierEtPropager"* qui est en $O(N*M^3+M*N^3)$.
  Nous recherchons le prochain indice tel que la case soit vide en $O(N*M)$.Puis nous faisons deux appels récursifs sur cet indice en considérant cette case coloriée en blanc puis en noir. Ce qui nous donne la relation de récurrence suivante :

  Pour un indice k donné : $T(N*M)=0$ (cas de base) $T(k)=2*T(k+1)+O(N*M^3+M*N^3)$
  $T(k)=2*T(k+1)+O(N*M^3+M*N^3)$
  $T(k)=2*[2*T(k+2)+O(N*M^3+M*N^3)]+O(N*M^3+M*N^3)$
  $T(k)=2^2*T(k+2)+(1+2)*O(N*M^3+M*N^3)$
  $T(k)=2^3*T(k+3)+(1+2+2^2)*O(N*M^3+M*N^3)$
  $\vdots$
  $T(k)=2^{N*M-k}*T(N*M)+(1+2+2^2+\cdots+2^N*M-k-1)*O(N*M^3+M*N^3)$
  $T(k)=(\sum_{i=0}^{N*M-k-1}2^i)*O(N*M^3+M*N^3)$ car $T(N*M)=0$
  Dans le pire des cas on parcourt toutes les cases et on commence à k=0.
  $T(k)=(2^{N*M}-1)*O(N*M^3+M*N^3)$
  $T(k)\sim O(2^{N*M}*(N*M^3+M*N^3))$

  On obtient donc une complexité exponentielle en $O(2^{N*M}*(N*M^3+M*N^3))$

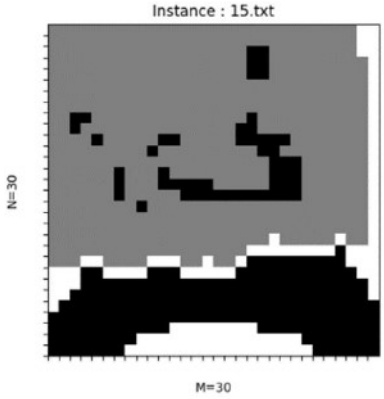
(Q13) Contrairement à la partie 1, la partie 2 résoud bien l'instance 11. Le résultat est illustré sur l'image suivante :



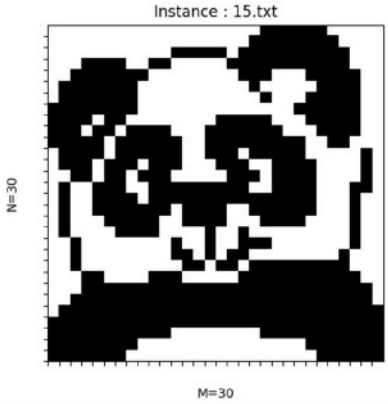
(Q14) • Temps de résolution des instances 11.*txt* à 16.*txt* avec la Partie 2 :

11.txt	2	4	0.0009970664978027344
12.txt	33	28	0.5694751739501953
13.txt	45	45	0.6413280963897705
14.txt	30	38	0.4547390937805176
15.txt	30	30	0.5854322910308838
16.txt	35	50	48.33165144920349

- Grilles obtenues avec chacune des deux méthodes (Partie 1 et Partie 2) sur l'instance 15.*txt* :

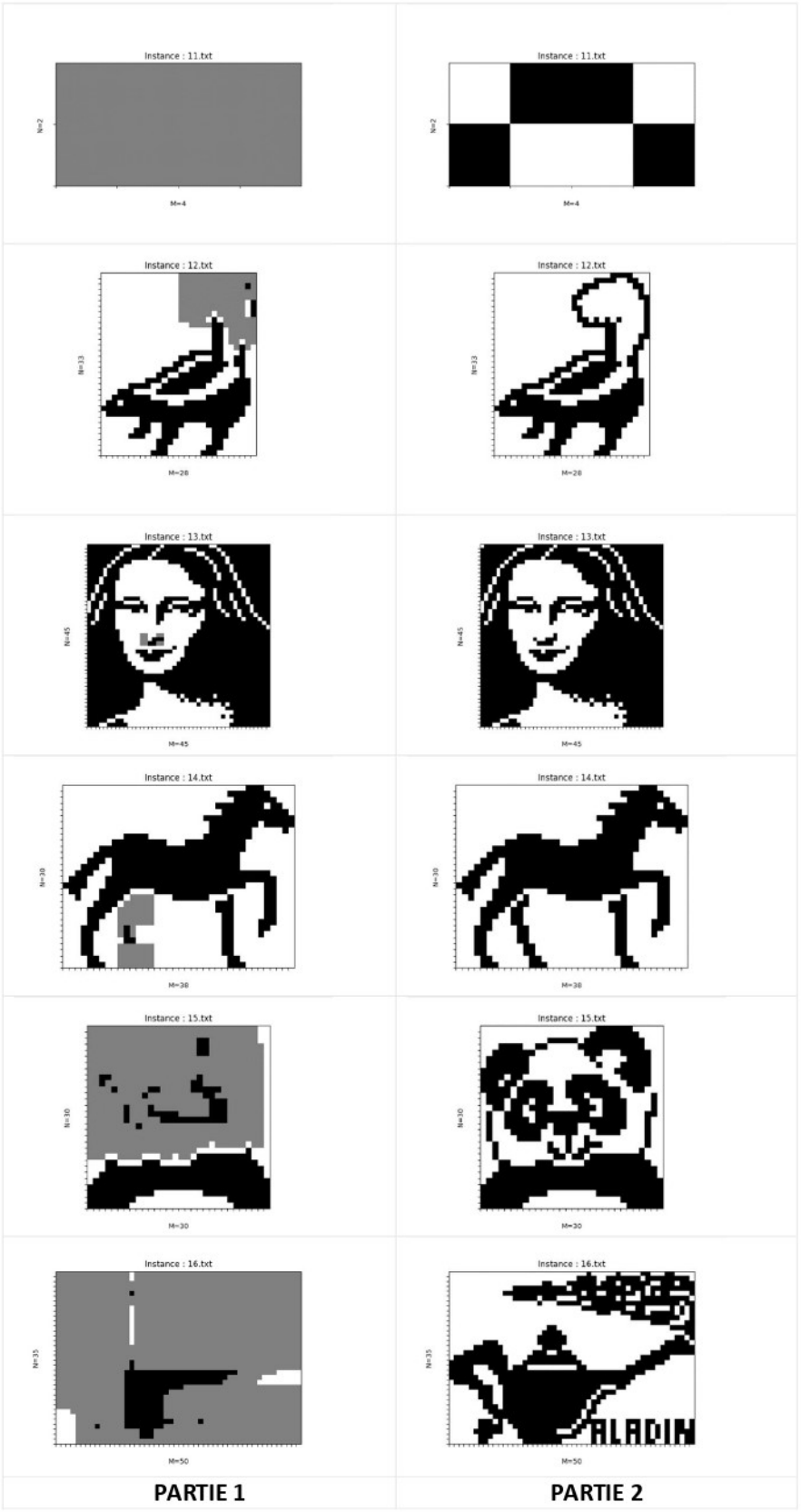


PARTIE 1



PARTIE 2

- Comparaison des deux méthodes sur les instances de 11.*txt* à 16.*txt* :



4 Conclusion

Ce projet a pour principal objectif d'élaborer des algorithmes résolvant un problème de tomographie discrète. Au-delà de la résolution de cette problématique, nous avons pu mettre en pratique les principes de la programmation dynamique qui nous a permis d'éviter des appels récursifs assez coûteux afin d'optimiser les complexités qui témoignent de l'efficacité des algorithmes réalisés. Nous avons également observé l'importance du choix des structures dans l'amélioration de cette efficacité. Enfin, grâce à ce projet nous avons pu étudier un sujet où un algorithme peut être très efficace mais n'est pas forcément valide pour toute instance. On trouve ainsi un moyen de généraliser les solutios à toute instance possible.

5 Annexe

5.1 Outils de réalisation

Le code accompagnant ce présent document a été réalisé avec les outils suivants :

- Langage de programmation : Python 3.8.2
- Ordinateur de calcul des temps d'exécution : DELL Inspiron 13 5000 Processeur Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 MHz, 4 cœur(s), 8 processeur(s) logique(s)

5.2 Comentaires

Nous avons choisi comme langage de programmation "Python" pour sa simplicité, et la bonne lisibilité qu'il offre des algorithmes. En comparaison aux langages "C" ou "C++" qui sont compilés, Python est un langage interprété, il est donc moins rapide.