

RAPPORT



Un problème de tomographie discrète



Université Pierre et Marie Curie
MOGPL

BOURMAUD Alexia
MARCHAL Louise

Table des matières

Raisonnement par programmation dynamique.....	2
Première étape.....	2
Généralisation	2
Propagation	3
Tests	3
La PLNE à la rescousse.....	4
Modélisation.....	4
Implantation et tests	7
 Tableau 1 : Tableau d'exécution avec le raisonnement par programmation dynamique	4
Tableau 2 : Tableau d'exécution avec le raisonnement par Programmation Linéaire en Nombres Entiers	8
Tableau 3 : Tableau d'exécution avec le raisonnement par Programmation Linéaire en Nombres Entiers comparé au raisonnement par Programmation dynamique	9

Raisonnement par programmation dynamique

Première étape

- j peut varier de 0 à $M-1$
- $T(j,l)$ signifie que l'on peut placer les sous-séquence 0 à l de la ligne i dans les j premières cases de cette ligne.

(Q1) Pour savoir s'il est possible de colorier la ligne l , en entier avec la séquence entière il faut que $j=M-1$ de sorte que les j premières cases à colorier correspondent à l'ensemble de la ligne l , et que $l=k$ pour que l corresponde à l'ensemble des sous-séquences de la ligne l .

Il faut donc que dans la liste des $T(j,l)$ calculé pour la ligne l , il y est $T(M-1,k)$ vrai.

(Q2)

1. Cas $l=0$: $T(j,0)$ signifie qu'il n'y a pas de séquence à placer donc **$T(j,l)$ est vrai**.
2. Cas $l \geq 1$:
 - a. $j < s_i - 1$: Le nombre $j + 1$ de cases disponibles est inférieur au nombre de cases nécessaires pour placer le bloc s_i , donc $T(j,l)$ est **faux**.
 - b. $j = s_i - 1$:
 - Si $l = 1$: On observe le premier bloc de la séquence, il n'est pas nécessaire de laisser une case blanche au début de la ligne donc on peut placer notre bloc de taille s_i dans $j+1$ cases. On en déduit que $T(j,l)$ est **vrai**.
 - Si $l > 1$: Dans la séquence observée si l strictement supérieur à 1 alors il existe un bloc $s_{i-1} \geq 1$ donc les $j+1$ cases disponibles ne suffisent plus pour placer les l premiers blocs. en effet j est strictement inférieur à $s_i + s_{i-1}$. On en déduit que $T(j,l)$ est **faux**.

(Q3)

On considère la variable globale « s » qui contient une séquence de blocs. Puis la variable passée en paramètre « l » qui signifie que l'on veut placer le l -ième bloc de la séquence s , et « j » l'indice de la position courante dans la ligne.

Dans le cas 2c) on fait un appel récursif : $T(j-s[l-1]-1, l-1)$.

(Q4)

La fonction $T(j,l)$ est codée dans le fichier projet.py, elle se nomme $T_old(j,L)$.

Généralisation

(Q5)

Dans un premier temps on change la signature de la méthode, on ajoute la ligne visitée (" li ") et la séquence (" s ") de blocs que l'on veut insérer dans la ligne. La définition de la fonction devient donc : $T(j,l,li,s)$. Voici l'algorithme obtenu :

- Si $l=0$: on parcourt la ligne li de la case 0 à la case j , si l'une des cases est noire on retourne faux sinon on retourne vrai.

- Sinon:
 - Si $j < s[l-1]-1$: on retourne faux.
 - Si $j = s[l-1]-1$:
 - si $l=1$, alors on veut placer le dernier bloc, dans ce cas si les cases précédents j ne sont pas blanches on renvoie vrai sinon faux.
 - sinon on retourne faux.
 - Sinon:
 - Si la case d'indice j est blanche on retourne $T(j-1, l, li, s)$.
 - Si la case d'indice j est noire :
 - Si une case parmi $li[j-s[l-1]+1:j]$ est blanche, cela signifie qu'il n'y a pas assez de place pour insérer le l -ième bloc de s donc on retourne faux.
 - Si $l=1$, alors on veut placer le dernier bloc. S'il y a une case noire parmi d'indice compris entre 0 et $j-s[l-1]+1$, alors on retourne faux, sinon on renvoie vrai.
 - Si la case d'indice $j-s[l-1]$ est noire, cela signifie que la case séparant deux blocs n'est pas blanche donc on retourne faux.
 - Sinon on retourne $T(j-s[l-1]-1, l-1, li, s)$.
 - Si la case n'est pas encore coloriée:
 - Si une des cases parmi $li[j-s[l-1]:j]$ est blanche, on vérifie qu'il n'y a pas de cases noire entre la case visitée ($li[j]$) et la case blanche trouvée. S'il y a une case noire on retourne faux sinon on retourne $T(j-c-1, l, li, s)$, avec c l'indice de la case blanche trouvée.
 - Si la case d'indice $j-s[l-1]$ est noire, cela signifie que la case séparant deux blocs n'est pas blanche, comme notre bloc n'est pas obligé de commencer à la case d'indice j , on essaye de le faire commencer une case avant. On retourne donc $T(j-1, l, li, s)$.
 - Sinon on teste si on peut placer un bloc à partir de la case d'indice j ou s'il est préférable de le placer à partir de la case d'indice $j-1$, en retournant $T(j-s[l-1]-1, l-1, li, s)$ or $T(j-1, l, li, s)$.

(Q6)

Notre algorithme est codé dans le fichier projet.py.

Propagation

(Q7)

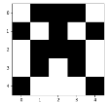
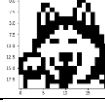
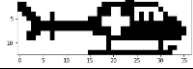
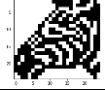
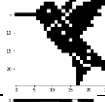


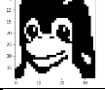
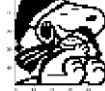

L'algorithme demandé est codé dans le fichier projet.py, nous avons écrit deux fonctions, une première nommée `propagation()` qui prend le nom d'un fichier en paramètre. Cette fonction lit le fichier et le converti en deux séquences de blocs, une pour les lignes et une pour les colonne, puis appelle la fonction `coloration()`. Cette dernière prend en paramètre une grille vierge ainsi que les deux séquences de blocs précédemment créés. Elle retourne la grille remplie en fonction des séquences de blocs.

Tests

(Q8)

Tous les tests ont été réalisés sur un ordinateur personnel, l'exécution est donc légèrement plus longue.

Tableau 1 : Tableau d'exécution avec le raisonnement par programmation dynamique

Instance	Temps d'exécution en seconde	Résultat obtenu
1	0.1888579692196161	
2	0.20082878977813357	
3	0.14214027880563657	
4	0.21496730150579213	
5	0.23633686245487917	
6	1.095474316956397	
7	0.3531392064762655	
8	0.8258488015586534	
9	163.77994139946534	
10	14.72011901345698	

(Q9)

Lorsque nous appliquons notre algorithme à l'instance 11, nous remarquons qu'il la colore totalement en blanc.

La PLNE à la rescousse

Dans les réponses qui vont suivre, sl correspondra à un tableau à deux dimensions représentant les séquences de blocs désirés par ligne, donc $sl[i][t]$ représentera la longueur du t -ième blocs de la i -ième ligne. Et sc correspondra à un tableau à deux dimensions représentant les séquences de blocs désirés par colonne, donc $sl[j][t]$ représentera la longueur du t -ième blocs de la j -ième colonne.

Modélisation

(Q10)

Voici la contrainte qui exprime le fait que si le t-ième bloc de la ligne i commence à la case (i, j), alors les cases (i, j) à (i, j + sl[i][t] - 1) sont noires. Pour chaque ligne i :

$$\sum_{k=j}^{j+sl[i][t]-1} (x_{ik}) \leq sl[i][t] * y_{ij}^t$$

Remarque: si j+sl[i][t]-1 est supérieur au nombre de colonnes de la ligne i on fait varier k de j au nombre de colonnes.

La contrainte qui exprime le fait que si le t-ième bloc de la colonne j commence à la case (i, j), alors les cases (i, j) à (i+sc[j][t]-1, j) sont noires s'écrit comme suit, pour chaque colonne j :

$$\sum_{k=i}^{i+sc[j][t]-1} (x_{kj}) \leq sc[j][t] * z_{ij}^t$$

Remarque: si i+sc[j][t]-1 est supérieur au nombre de lignes de la colonne j on fait varier k de i au nombre de lignes.

(Q11)

La contrainte suivante exprime le fait que si le t-ième bloc de la ligne i commence à la case (i, j), alors le (t + 1)ième ne peut pas commencer avant la case (i, j + sl[i][t] + 1). Pour chaque ligne on a :

$$y_{ij}^t + \sum_{k=j}^{j+sl[i][t]+1} y_{ik}^{t+1} \leq 1$$

La contrainte suivante exprime le fait que si le t-ième bloc de la colonne j commence à la case (i, j), alors le (t + 1)ième ne peut pas commencer avant la case (i+sc[j][t]+1, j). Pour chaque colonne on a :

$$z_{ij}^t + \sum_{k=i}^{i+sc[j][t]+1} z_{kj}^{t+1} \leq 1$$

Remarque: Comme pour les contraintes précédentes, si j+sl[i][t]+1(respectivement i+sc[j][t]+1) est supérieur au nombres de colonnes (respectivement lignes) de la ligne i (respectivement de la colonne j), alors k varie de j (resp. i) au nombre de lignes(resp. colonnes).

(Q12)

Pour formuler notre problème sous la forme d'un PLNE nous avons rajouté plusieurs contraintes.

Le t-ième bloc d'une ligne d'indice i ne peut commencer que sur une seule case:

$$\sum_{k=0}^{M-1} y_{ik}^t = 1 \text{ avec } M \text{ le nombre de colonnes dans la ligne } i$$

Le t-ième bloc d'une colonne d'indice j ne peut commencer que sur une seule case:

$$\sum_{k=0}^{N-1} z_{kj}^t = 1 \text{ avec } N \text{ le nombre de lignes dans la colonne } j$$

Le nombre de cases coloriées en noires dans une ligne d'indice i correspond à la sommes de tous les blocs de cette même ligne:

$$\sum_{k=0}^{M-1} x_{ik} = \sum_{t=0}^l sl[i][t]$$

avec M le nombres de colonnes dans la ligne i et l le nombre de blocs dans la séquence correspondant à la ligne i .

Le nombre de cases coloriées en noires dans une colonne d'indice j correspond à la sommes de tous les blocs de cette même colonne:

$$\sum_{k=0}^{N-1} x_{kj} = \sum_{t=0}^l sl[j][t]$$

avec N le nombres de colonnes dans la ligne i et l le nombre de blocs dans la séquence correspondant à la colonne j .

On en déduit le programme linéaire suivant. Avec N le nombre de lignes et M le nombre de colonnes et l le nombre de blocs dans une séquence.

$$\text{Max } \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} x_{ij}$$

sc: Pour i variant de 0 à $N-1$, j variant de 0 à $M-1$ et t variant de 0 à l :

$$\sum_{k=j}^{j+sl[i][t]-1} (x_{ik}) \geq sl[i][t] * y_{ij}^t$$

$$y_{ij}^t + \sum_{k=j}^{j+sl[i][t]+1} y_{ik}^{t+1} \leq 1$$

Pour j variant de 0 à $M-1$, i variant de 0 à N , et t variant de 0 à l :

$$\sum_{k=i}^{i+sc[j][t]-1} (x_{kj}) \geq sc[j][t] * z_{ij}^t$$

$$z_{ij}^t + \sum_{k=i}^{i+sc[j][t]+1} z_{kj}^{t+1} \leq 1$$

Pour i variant de 0 à $N-1$:

$$\sum_{k=0}^{M-1} x_{ik} = \sum_{t=0}^l sl[i][t]$$

avec l le nombre de blocs dans la séquence correspondant à i .

Pour j variant de 0 à $M-1$:

$$\sum_{k=0}^{N-1} x_{kj} = \sum_{t=0}^l sl[j][t]$$

avec l le nombre de blocs dans la séquence correspondant à j .

Pour i variant de 0 à $N-1$, et t variant de 0 à l :

$$\sum_{k=0}^{M-1} y_{ik}^t = 1$$

Pour j variant de 0 à $M-1$, et t variant de 0 à l :

$$\sum_{k=0}^{N-1} z_{kj}^t = 1$$

Pour i variant de 0 à $N-1$, j variant de 0 à M et t variant de 0 à l :

$$x_{ij}, y_{ij}^t, z_{ij}^t \in \{0,1\}$$

Implantation et tests

(Q13)

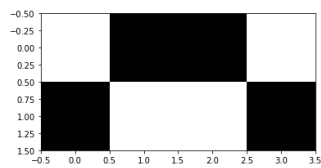
Pour une ligne li l'intervalle hors duquel le i -ième bloc ne peut commencer est:

$li[\text{somme}(sl[0 : i]) + i : M - (\text{somme}(sl[i : k]) - k + i + 1)]$

avec M le nombre de colonne dans la ligne i et k le nombre de blocs de la ligne i .

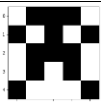
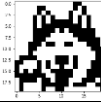

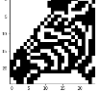
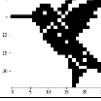





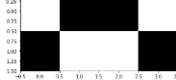
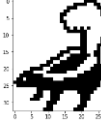


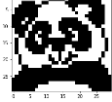

(Q14)

Pour l'instance 11 nous obtenons le coloriage suivant:



(Q15)

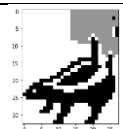
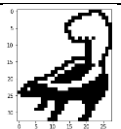

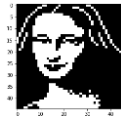
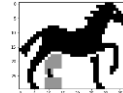

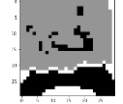
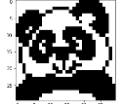
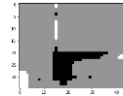
Tableau 2 : Tableau d'exécution avec le raisonnement par Programmation Linéaire en Nombres Entiers

instance	Temps d'exécution en seconde	Résultat
1	0.0682594281916902	
2	12.273912189817402	
3	0.3894537916660372	
4	25.56986527918322	
5	9.309254882119284	
6	89.00597446043709	
7	1.4811296461313077	
8	5.59395606470224	
9	∞ *	
10	18.981692241470455	
11	0.03800803491867555	
12	196.92217116491213	
13	3.5946757336395194	
14	1.877564528552739	
15	443.3773806467203	
16	297.13178323843476	

*Nous avons arrêté le programme en cours de route car il prenait trop de temps.

Lorsque nous appliquons notre algorithme sur les instances 12 à 16.txt, nous observons que notre algorithme ne se termine jamais. Cela est sûrement dû au fait que notre algorithme ne parvient pas à colorier certaines cases des grilles. Il doit probablement hésiter continuellement entre les colorier en blanc ou en noir et donc réexaminer ces cases indéfiniment. Pour résoudre ce problème nous avons décidé d'ajouter un compteur à notre fonction pour ne pas qu'elle essaye de colorier les cases plus de 33 fois. Nous avons choisi 33 car c'est le nombre d'itérations nécessaires à notre algorithme pour réaliser l'instance 9.

Tableau 3 : Tableau d'exécution avec le raisonnement par Programmation Linéaire en Nombres Entiers comparé au raisonnement par Programmation dynamique

instance	Raisonnement PD	Résultat	PLNE	Résultat
12	0.48550542961140764		196.92217116491213	
13	1.1730101493709526		3.5946757336395194	
14	0.36827943997229795		1.877564528552739	
15	0.8725634713265435		443.3773806467203	
16	5057.989557626375		297.13178323843476	