

TP : Réduction intelligente

2026-2027

Thibaud Lemanissier

PC Lycée Carnot



L'objectif de ce TP est de réduire la largeur d'une image. Par exemple la photo suivante du film « Hundreds of beavers » :





Figure 1 – À gauche l'image dans son format d'origine et à droite réduite de manière naïve de 50%

Cette méthode naïve de réduction est qu'elle ne fait pas la différence entre les parties importantes de l'image et celle qui le sont moins. Dans le cas qui nous intéresse, la neige est moins importante que les sujets au premier plan.

Dans ce TP, l'image considérer sera en niveau de gris c'est-à-dire que chaque pixel ne contient qu'une information le niveau de gris stocké sous la forme d'un entier compris en 0 (noir) et 255 (blanc).

Pour rappel pour utiliser l'image en python, on devra dans un premier temps sauvegarder le fichier python et l'image dans un même dossier commun puis utiliser la bibliothèque `matplotlib.pyplot` que l'on importera avec l'alias `plt` :

```
import matplotlib.pyplot as plt

M = plt.imread("Beavers.jpg").tolist() # permet de convertir le tableau en liste
```

Si vous voulez sauvegarder une image que vous avez créé, vous pouvez utiliser la fonction `imsave` de `matplotlib.pyplot` qui s'utilise de la manière suivante :

```
plt.imsave("Beavers modif.jpg", M, cmap = "gray") # en niveau de gris
```

1. Calcul de la matrice d'énergie d'une image

Pour détecter les zones importantes de l'image en niveau de gris, nous allons nous centrer sur les zones où il y a une forte variation de la luminosité des pixels. Pour détécter ces zones dans une matrice $(M_{i,j})_{0\leqslant i\leqslant l-1,0\leqslant j\leqslant c-1}$, nous allons commencer par calculer la matrice d'**énergie** $(E_{i,j})_{0\leqslant i\leqslant l-1,0\leqslant j\leqslant c-1}$ de la photo. Cette matrice est calculer à l'aide de la norme $(\|.\|_1)$ du gradient pour un pixel de coordonnée (i,j) qui est à l'intérieur de l'image, il vaut :

$$E_{i,j} = \frac{1}{2}|M_{i-1,j} - M_{i+1,j}| + \frac{1}{2}|M_{i,j-1} - M_{i,j+1}|.$$

Si $i = 0$, on remplacera $\frac{1}{2}|M_{i-1,j} - M_{i+1,j}|$ par $|M_{i,j} - M_{i+1,j}|$ et si $i = l-1$, on remplacera $\frac{1}{2}|M_{i-1,j} - M_{i+1,j}|$ par $|M_{i-1,j} - M_{i,j}|$ et de même pour j. On peut considérer la matrice M_{ex} suivante :

2	2	1	7	4	5	3
4	1	3	6	6	6	2
7	7	6	8	7	4	6
6	7	1	7	7	8	5
8	3	4	1	4	1	7

2	1.5	4.5	2.5	3	1.5	3
5.5	3	5	2	1.5	2.5	5.5
1	3.5	1.5	1	2.5	1.5	3.5
1.5	4.5	1	6.5	2	2.5	3.5
7	6	4	6	3	8.5	8

FIGURE 2 – À gauche la matrice à droite la matrice d'énergie E_{ex}.

1. Écrire une fonction `gradient(M, i, j)` qui prend en argument la matrice `M` et les coordonnées `i` et `j` et qui calcule l'énergie `E` du pixel de coordonnée (i,j). On pourra calculer la première composante puis la seconde et renvoyer la somme des deux pour limiter le nombre de tests.
2. Écrire une fonction `Energie(M)` qui prend en argument la matrice `M` et renvoie la matrice d'énergie correspondante.

2. Chemin d'énergie minimale

Pour limiter les effets de création de discontinuité, lorsque l'on réduira d'un pixel la largeur de la photo, nous enlèverons les pixels de sorte à ce que chaque pixel supprimé soit voisin du pixel supprimé dans la ligne du dessus, c'est-à-dire si le pixel (i,j) est enlevé alors les seuls pixels que nous pourrons enlever dans la ligne en dessous seront $(i+1,j-1)$, $(i+1,j)$, $(i+1,j+1)$. Nous dirons qu'ils forment un **chemin**. On peut considérer la matrice M_{ex} suivante :

2	2	1	7	4	5	3
4	1	3	6	6	6	2
7	7	6	8	7	4	6
6	7	1	7	7	8	5
8	3	4	1	4	1	7

2	2	1	7	4	5	3
4	1	3	6	6	6	2
7	7	6	8	7	4	6
6	7	1	7	7	8	5
8	3	4	1	4	1	7

FIGURE 3 – À gauche c'est un chemin, à droite ce n'en est pas un.

Nous allons prioritairement enlever les chemins de faible énergie. Pour déterminer ces chemin nous allons procéder de deux manières distinctes.

2.1. Stratégie gloutonne

Une première stratégie pour déterminer un chemin de faible énergie est de :

- déterminer le pixel de plus faible énergie dans la première ligne ;
- pour chaque pixel de cooronnée `(i,j)` (pas sur la dernière ligne) déjà dans le chemin on ajouter `(i+1,j')` où `j'` est choisi dans `j-1, j, j+1` de sorte à ce que l'énergie de la photo en `(i+1,j')` soit minimale.

Par exemple sur la matrice E_{ex} précédente, on obtient le chemin :

2	1.5	4.5	2.5	3	1.5	3
5.5	3	5	2	1.5	2.5	5.5
1	3.5	1.5	1	2.5	1.5	3.5
1.5	4.5	1	6.5	2	2.5	3.5
7	6	4	6	3	8.5	8

FIGURE 4 – Chemin obtenu par la statégie.

On ne stockera que l'ordonnée j.

3. Expliquer pourquoi cet algorithme est glouton.
4. Écrire une fonction `indice_min(L, d, f)` qui prend en argument une liste `L`, un indice `d` de début et un indice `f` de fin et qui renvoie l'indice de la première occurrence du minimum de la liste restreinte entre l'indice `d` (inclus) et l'indice `f` (exclus).

   Par exemple :

   - `indice_min([2, 1.5, 4.5, 2.5, 3, 1.5, 3], 0, 7)` renvoie `1` ;
   - `indice_min([5.5, 3, 5, 2, 1.5, 2.5, 5.5], 0, 3)` renvoie `1`.
5. Écrire une fonction `chemin_glouton(E)` qui renvoie `J` la liste des ordonnées du chemin obtenue par l'algorithme glouton précédent.

   Par exemple, pour la matrice E_{ex}, `chemin_glouton(E_ex)` renvoie `[1, 1, 0, 0, 1]`.

2.2. Programmation dynamique

Comme vous pouvez le remarquer la stratégie utilisé précédement ne permet pas d'obtenir un chemin d'énergie minimale. Nous allons utiliser la programmation dynamique pour déterminer le chemin d'énergie minimal. Pour cela nous allons construire une matrice $(E^{chemin}_{i,j})_{0\leqslant i\leqslant l-1,\ 0\leqslant j\leqslant c-1}$ tel que $E^{chemin}_{i,j}$ contient l'énergie minimal d'un chemin finissant en (i,j). On a alors pour tout $i \in [\![0, l-2]\!]$ l'égalité :

$$E^{chemin}_{i+1,j} = E_{i+1,j} + \min\left(E^{chemin}_{i,j-1}, E^{chemin}_{i,j}, E^{chemin}_{i,j+1}\right).$$

On a aussi pour tout $j \in [\![0, c-1]\!]$:

$$E^{chemin}_{0,j} = E_{0,j}$$

Dans le cas de la matrice précédente, on obtient :

2	1.5	4.5	2.5	3	1.5	3
7	4.5	6.5	4.5	3	4	7
5.5	8	6	4	5.5	4.5	7.5
7	10	5	10.5	6	7	8
14	11	9	11	9	14.5	15

Figure 5 – Matrice d'énergie minimale des chemins E^{chemin}_{ex}.

6. Écrire une fonction `Energie_chemin(E)` qui prend en argument la matrice d'énergie `E` et qui renvoie la matrice E^{chemin}.

Il reste à reconstruire un chemin d'énergie minimal à partir de cette matrice. Pour cela, il suffit de :

- repérer dans la dernière ligne l'indice du minimum dans la dernière ligne ;
- pour chaque pixel de cooronnée `(i,j)` (pas sur la première ligne) déjà dans le chemin on ajouter `(i-1,j')` où `j'` est choisi dans `j-1, j, j+1` de sorte à ce que l'énergie du chemin en `(i-1,j')` soit minimale ;
- on renverse la liste pour qu'elle soit rangé dans le bon ordre.

Dans le cas de la matrice précédente, on obtient :

2	1.5	4.5	2.5	3	1.5	3
7	4.5	6.5	4.5	3	4	7
5.5	8	6	4	5.5	4.5	7.5
7	10	5	10.5	6	7	8
14	11	9	11	9	14.5	15

Figure 6 – Matrice d'énergie minimale des chemins.

De la même manière on ne stockera que l'ordonnée j.

7. Écrire une fonction `chemin_dyn(E)` qui renvoie `J` la liste des ordonnées du chemin d'énergie minimale. Par exemple, pour la matrice E^{chemin}_{ex}, la fonction renvoie `[5, 4, 3, 2, 2]`.

3. Réduction d'image

Nous allons maintenant pouvoir tester nos diverses méthodes pour réduire la taille des images.

3.1. Algorithme Naïf

Commençons par rappeler la méthode de réduction naïve de la dimension d'une image.

8. Écrire une fonction `reduc_naive(M, e)` qui prend en argument la matrice `M` de l'image et un flottant `e` et qui renvoie la matrice de l'image dont la largeur est égale à `e` fois la largeur de celle de départ.

3.2. Algorithme Glouton

Nous allons maintenant écrire un algorithme permettant de réduire l'image en enlever un à un chemin de faible énergie obtenu par l'algorithme glouton.

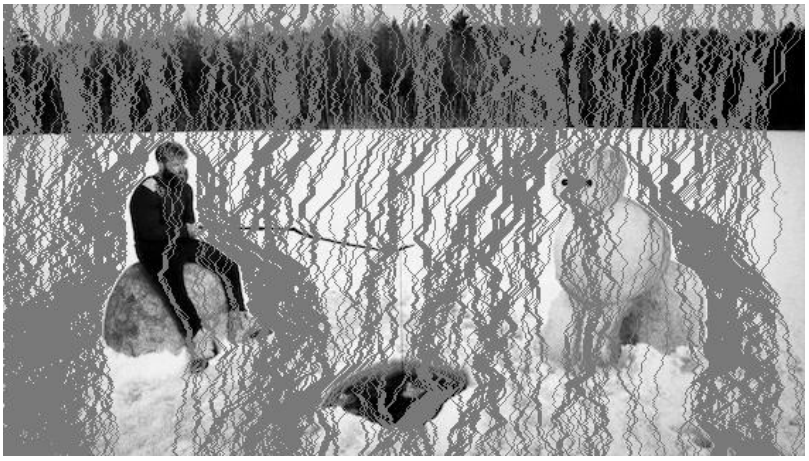


Figure 7 – Représentation des chemins obtenu par l'algorithme glouton pour réduire l'image de 50%.

Nous allons plus précisément à chaque itération :

- Déterminer un chemin de faible énergie ;
- Enlever ce chemin à l'image ;
- actualiser la matrice d'énergie.

Et on répète cette opération le nombre de fois nécessaire pour obtenir le bon ratio.

9. Écrire une fonction `reduc_chemin(M, J)` qui prend en argument une matrice `M` et les ordonnées d'un chemin à enlever `J` et qui renvoie la matrice obtenue une fois ce chemin retiré.
   Si on reprend la matrice E_{ex}, avec pour chemin `[1, 1, 0, 0, 1]`, la matrice renvoyée sera

2	4.5	2.5	3	1.5	3
5.5	5	2	1.5	2.5	5.5
3.5	1.5	1	2.5	1.5	3.5
4.5	1	6.5	2	2.5	3.5
7	4	6	3	8.5	8

Figure 8 – Matrice réduite.

Il reste à actualiser l'énergie pour cela, il n'est pas nécessaire de recalculer toute la matrice `E` puisque la majeure partie reste inchangée. Il suffit de recalculer les énergies qui sont de part et autre du chemin enlevé. Pour actualiser l'énergie, on va donc :

- Enlever le chemin à l'énergie ;
- Pour tout ligne calculer la nouvelle énergie des deux pixels qui sont de part et d'autre du pixel enlevé.

10. Écrire une fonction `actualise_energie(E, M, J)` qui prend en argument la matrice d'énergie `E`, la matrice de l'image `M` (déjà réduite) et les ordonnées des chemins `J` et qui renvoie la nouvelle matrice d'énergie.
    Si on reprend la matrice E_{ex}, après actualisation on obtient :

3	4.5	2.5	3	1.5	3
3.5	3.5	2	1.5	2.5	5.5
2.5	1.5	1	2.5	1.5	3.5
6.5	1	6.5	2	2.5	3.5
5	6.5	6	3	8.5	8

Figure 9 – Matrice d'énergie actualisée.

11. Compléter la fonction suivante pour qu'elle effectue la réduction par algorithme glouton de l'image :

```
def reduc_glouton(M, e) :
    l, c = len(M), len(M[0])
    n = ... # nb d'itération nécessaire
    E = ...
    for k in range(n) :
        J = ...
        M = ...
        E = ...
    return M
```

3.3. Algorithme dynamique

Nous allons maintenant appliquer l'algorithme dynamique.

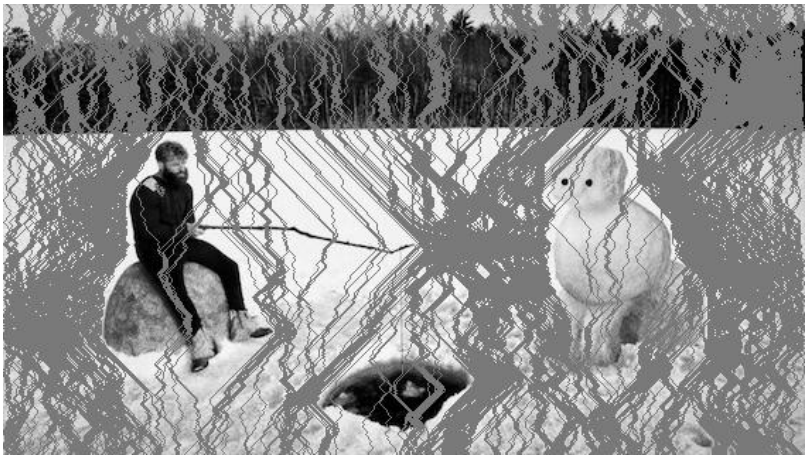


Figure 10 – Représentation des chemins obtenu par l'algorithme dynamique pour réduire l'image de 50%.

La démarche sera la même pour l'algorithme glouton avec l'ajout d'une étape d'actualisation de l'énergie des chemins. Pour actualiser cette matrice, on remarque que les seuls coefficients qui sont susceptibles d'être modifiés sont ceux qui sont dans un « cone » dont le sommet est le premier pixel du chemin enlevé :

2	1.5	4.5	2.5	3	1.5	3
7	4.5	6.5	4.5	3	4	7
5.5	8	6	4	5.5	4.5	7.5
7	10	5	10.5	6	7	8
14	11	9	11	9	14.5	15

2	1.5	4.5	2.5	4	2
7	4.5	6.5	4	4	7.5
5.5	8	6	5.5	5.5	7.5
7	8	8	7.5	8	9
14	14.5	14	10.5	16	16

Figure 11 – En gris clair sont représenté les coefficients qu'il faut modifier.

12. Compléter la fonction `actu_energie_chemin(Ech, E, J)` qui prend en argument la matrice d'énergie des chemins à actualiser `Ech`, la matrice d'énergie déjà actualisée et le chemin à enlever et qui renvoie la matrice d'énergie des chemins actualisée :

```
def actu_energie_chemin(Ech, E, J) :
    Ech = ...
    l, c = len(Ech), len(Ech[0])
    if J[0] < c :
        Ech[0][J[0]] = ...
```

```
    if J[0] > 0 :
        Ech[0][J[0]-1] = ...
    for i in range(1, l) :
        for j in range(..., ...) :
            Ech[i][j] = ...
    return Ech
```

13. Compléter la fonction suivante pour qu'elle effectue la réduction par algorithme dynamique de l'image :

```
def reduc_dyn(M, e) :
    l, c = len(M), len(M[0])
    n = ... # nb d'itération nécessaire
    E = ...
    for k in range(n) :
        J = ...
        M = ...
        E = ...
        Ech = ...
    return M
```