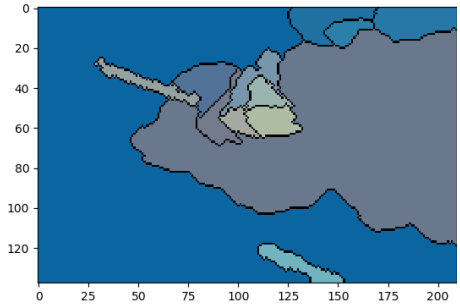
TIPE d'informatique pratique

Segmentation d'image
Algorithme de Ligne de Partage des Eaux par Inondation
sous le langage *Python*

Objectif : unifier les zones à peu près homogènes de l'image à segmenter (soit **n** son nombre de pixels)

Motivation : se confronter aux difficultés soulevées par la programmation d'un algorithme dont le principe est assez intuitif : inondation d'un relief, et placement de frontières à la jointure entre deux bassins



Note : La programmation des segmentation et fusion avant amélioration ont fait l'objet d'un travail de groupe, je me suis particulièrement intéressée aux questions de complexité.

1. De l'intuition à la programmation

La segmentation repose sur la représentation topographique de l'image en fonction du gradient : une valeur affectée à chaque pixel traduisant la variation de couleur à cet endroit.
On travaille à partir d'une **image de gradients** qui informe sur la répartition géographique des gradients, et d'**étages** qui informent à l'inverse sur la répartition des pixels en intervalles de valeurs prises par le gradient. Les pixels y sont ordonnés par gradients croissants.

L'algorithme crée des bassins et y affecte les pixels en suivant les étapes détaillées par P. Soille et L. Vincent dans *Watersheds in digital spaces : An efficient algorithm based on immersion simulations*. La première partie du travail a consisté à comprendre la pertinence de leur procédure, en particulier :
- l'expansion des bassins de proche en proche, puis la considération des pixels par gradient croissant au sein d'un étage, pour ne pas créer plus d'**un bassin par zone homogène** continue ;
- l'affectation des pixels par étages croissants, pour limiter l'extension des bassins sur les zones homogènes et permettre le placement des **frontières sur les frontières** entre deux teintes de l'image initiale

➢ Notons la dépendance du résultat en la hauteur des étages **h** et le coefficient initial de floutage **f** (cf. page suivante)

2. Optimisation du coût de la segmentation

Au cours du traitement, chaque pixel est successivement dans un **étage**, dans la **file d'attente** pour intégrer le bassin en cours de remplissage, puis affecté à un **bassin**. Il faut ôter le pixel de chaque ensemble lorsqu'on le place dans le suivant.

➔ Se pose la question du **coût de ces suppressions**.

1. En modélisant les étages et la file d'attente par des listes Python, on aboutit à une **complexité quadratique** en le nombre total de pixels, car la suppression est linéaire en la longueur de la liste avec la **méthode *.remove***.

2. L'accès au $k^{ième}$ élément en temps quasi-constant m'a permis remplacer les suppressions dans la file d'attente par un **compteur** qui n'avait qu'à s'incrémenter pour indiquer l'indice du pixel suivant à considérer. Mais cela ne peut être mis en place pour les suppressions dans l'étage, où les pixels à supprimer n'ont pas de position privilégiée, du fait du traitement par voisinage et non indice croissant.

3. Enfin, les **tables de hachage** ont permis un accès en temps constant à chaque élément dans l'étage et dans la file d'attente sans surconsommation de mémoire. La clé est un pixel, la valeur est le couple (pixel_précédent, pixel_suivant). Pour ôter n'importe quel élément de la liste, il suffit de changer les valeurs de pixel_suivant et pixel_précédent de ses voisins de gauche et droite respectivement.

➔ **complexité linéaire** (cf. page suivante)
la durée de segmentation est passée de 39.0 secondes à 3.5 secondes pour notre image : n= 28 288, h = 5, f= 8

3. Optimisation du résultat : fusion des petits bassins

Pour corriger la sur-segmentation, on a voulu rattacher les bassins négligeables à des bassins voisins de couleur proche.

La fusion des petits bassins force à mémoriser explicitement les pixels contenus dans chaque bassin, identifié par un numéro appelé **label**. On note **nbi** le nombre initial de bassins. Un **tableau « effectifs »** contient pour chaque label le nombre de pixels contenus dans le bassin associé. Il est calculé en O(n).

La fusion des bassins fonctionne de la manière suivante :

Tant qu'il existe un bassin à fusionner (d'effectif <**eff_max** et pas dans la liste des bassins non traitables, en O(nb_bassins_initial)) :

Si son plus proche bassin voisin est suffisamment proche :

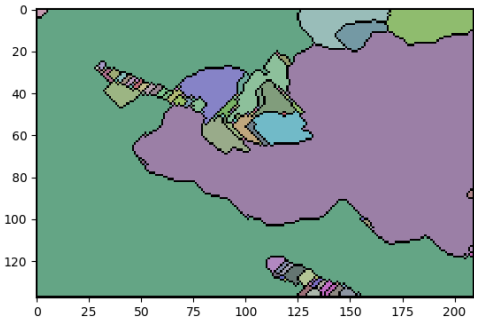
(la norme carrée de la différence des composantes RVB moyennes des bassins est minimale et < **diff_max**, calculs en O(n))

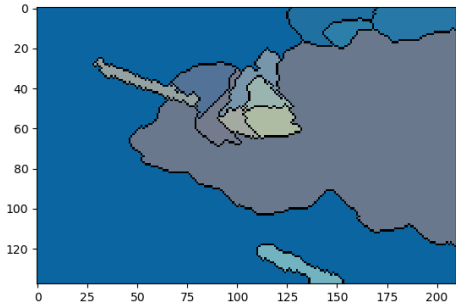
fusion des deux bassins en un nouveau bassin de label inédit sur l'**image-résultat**

(changement du label des pixels des deux bassins initiaux et de la frontière, sur l'image-résultat, en O(n))

mise à jour des **effectifs** de chaque bassin (calcul du tableau effectifs en O(n))

Sinon : son label est ajouté à la liste des bassins non traitables





n=28 288 pixels

Paramètres :
h = 5
f = 8
eff_max = 120
diff_max = 180

Résultats :
nbi= 126 bassins initialement
14 bassins après fusion

4. Optimisation du coût de la fusion

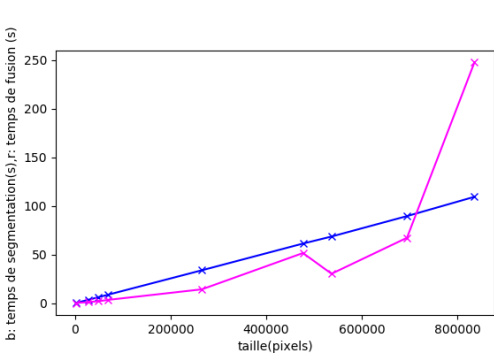
La complexité de la boucle est O(n), et elle est appelée au plus nbi fois, donc la fonction est un O(n*nbi) = O(n²).

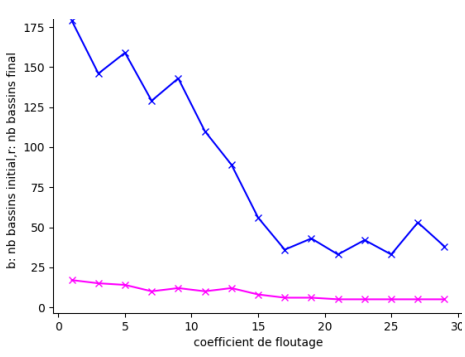
➔ Il s'agit de **réduire la constante** du O(n) de la boucle.

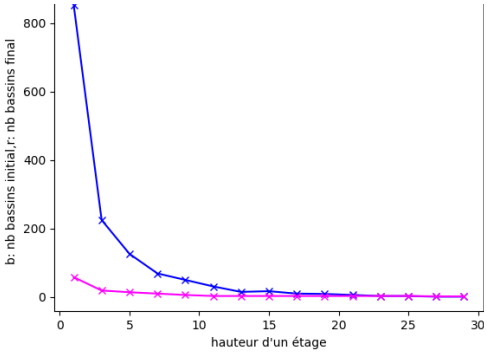
On s'appuie sur un **tableau « bassins »** répertoriant pour chaque bassin **[effectif E*(int)*, contenu C*(int*int list)*, [Rmoy, Vmoy, Bmoy] *(int list)*, est_traitable *(bool)*]**, créé en O(n).
Au lieu de le recalculer, chaque fusion vide les 2 bassins fusionnés et ajoute une ligne correspondant au nouveau bassin, en O(effectif_nouveau_bassin) : [E1+E2, C1 ∪ C2, [int(R1*E1 + R2*E2)/(E1+E2), V, B], true], rendant l'accès à de nombreuses informations en temps constant.

➔ **performance temporelle fortement améliorée**, même si la complexité est toujours O(n²)
Sur notre exemple, la durée de fusion passe de 61.5 secondes à 1,2 secondes

➔ **réduction drastique de l'influence de la tolérance et du floutage** sur le nombre de bassins finaux







➢ Notons la dépendance du résultat en l'effectif maximal d'un bassin fusionnable **eff_max** et la différence de couleur maximale entre deux bassins fusionnés **diff_max** (cf. 5)