ESIEE PARIS

Traitement d'images "out-of-core"

PRJ-4000

CARNEIRO ESPINDOLA Stela
NEGHNAGH-CHENAVAS Jules
PARIS Édouard
POV Cécile

Suiveur : Jean COUSTY

22 janvier 2019

Sommaire

A. Introduction

Les systèmes d'imagerie récents comme les microscopes électroniques, les tomographes, les scanners médicaux utilisés par exemple dans les laboratoires de recherche et développement en biologiques ou dans les sciences des matériaux sont capables de générer des volumes de données 2D/3D atteignant plusieurs dizaines de Go.

Ces volumes sont destinés à l'analyse et à la quantification via les techniques de traitement d'images. Hors, la conception de ces algorithmes n'a pas été originellement pensée dans le sens d'une gestion mémoire appropriée aux grands volumes de données, c'est-à-dire des données qui ne peuvent être stockées intégralement dans la mémoire principale. Même si une grande partie des algorithmes est compatible avec une approche par blocs, il est nécessaire pour les autres de repenser fondamentalement l'approche algorithmique.

C'est en particulier le cas des algorithmes de segmentation d'images, notamment par ligne de partage des eaux (voir illustrations ci-dessous). L'approche algorithmique pour traiter ces données de grande taille est dite « out-of-core ».

1. Objectif

Les objectifs pour cette projet sont:

- Développer un nouvel algorithme de ligne de partage des eaux <<out-of-core>> ;
- Développer un logiciel <<out-of-core>> de ligne de partage des eaux pour des images de grandes taille ;
- Produire des résultats expérimentaux.

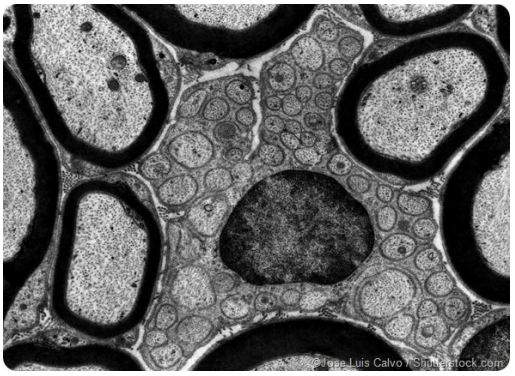
2. Contexte

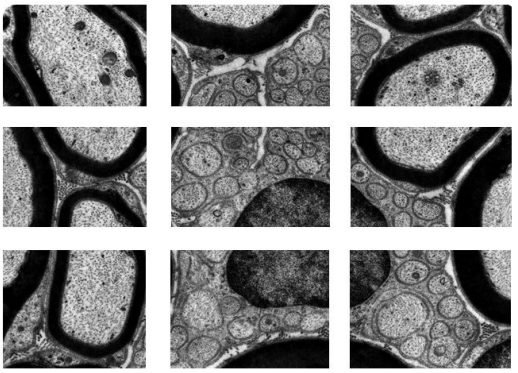
B. Développement

1. La proposition du projet

a. Segmentation par bloc

In order to deal with images of very large sizes, we propose to divided the original image into smaller blocks *B*, that can be loaded into the memory. Below, the images show an example of an medical image which was divided into blocks. This image was not used in the experiments and it is just an example of how the division of the future input images can be read by our algorithm.

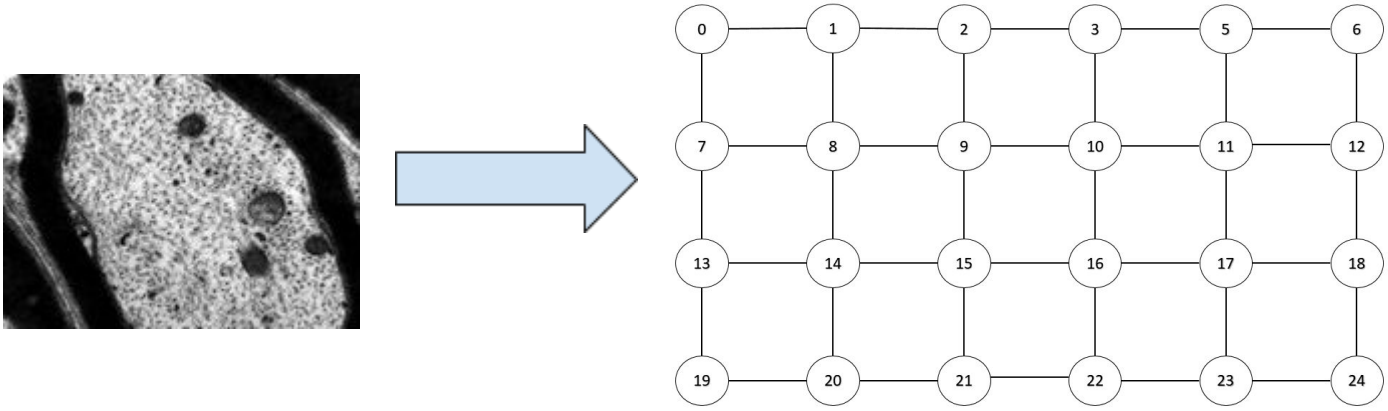




The number of blocks will be determined by the width and the height chosen for the block. We'll call in this report the width of the block as block_x and the height block_y, opposed to the width and height of the image which we'll call img_x and img_y respectively. We will enumerate each block from left to right and then from the top to the bottom like in the image below.

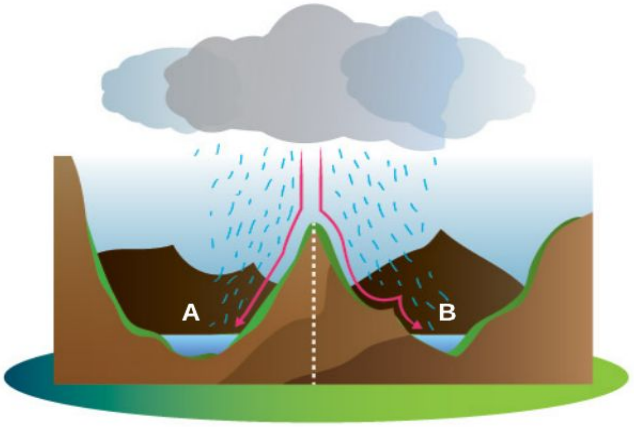
0	1	2
3	4	5
6	7	8

Each block will be treated as an individual image, which will be read by our algorithm as a graph. The graph G(N, E) it's composed by a set of nodes N and of edges E, in which each edge e in E links only to nodes of N. Each x in N represents a pixel on the original image, and each node is connected with its four neighbours. The figure below shows how the graph is built in the memory.



b. Ligne de partage des eaux

Si l'on fait tomber une goutte d'eau sur le flanc d'une montagne, celle-ci ruissellera jusqu'un potentiel « bassin de rétention ». Si l'on fait ruisseler une autre goutte d'eau sur le l'autre flanc de la même montagne, celle-ci rejoindra un autre bassin de rétention. Topographiquement, une ligne de partage des eaux est la ligne séparera les deux flancs. Ainsi si l'on dépose la goutte d'eau d'un côté ou de l'autre de la ligne de partage des eaux, la goutte atteindra un bassin de rétention différent.



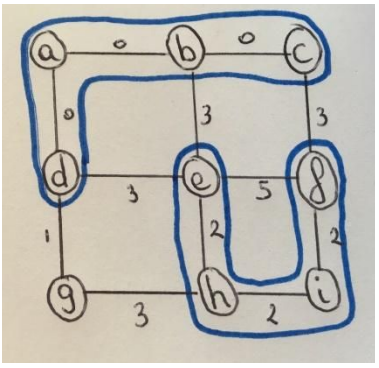
Cette notion de ligne de partage des eaux est intéressante dans le domaine du traitement d'image car elle peut permettre de « segmenter » une image, c'est-à-dire de séparer l'image en plusieurs régions logiquement reliées.

Tout d'abord, nous allons travailler avec des « edge weighted graphs », c'est-à-dire que chaque arc va posséder un poids, et les Sommets n'en auront pas (en tout cas, au début). Mais comment le déterminer, puisqu'une image est une suite de pixels, chaque pixel correspondant à un sommet ? Eh bien c'est à notre bon vouloir, en fonction de ce que l'on veut faire sur l'image. Si l'on travaille sur une image en deux dimension (en noir et blanc), un arc peut être simplement la différence entre la valeurs des deux pixels qu'il relie. Si l'on travaille sur une image en couleur, il y aura probablement une autre technique à définir, etc. Pour le moment nous travaillons avec des images en niveau de gris, et chaque arc correspond à la différence de valeur des sommets(correspondant aux pixels) qu'il joint.

Pour déterminer ce qu'est une ligne de partage des eaux algorithmiquement, expliquons les deux notions suivantes :

Soit F un graphe. Un <u>minimum local de F</u> est un sous-graphe connexe X de F dans lequel tous les arcs ont un poids égal k (réel positif), et pour lequel tous les arcs adjacents de X sont supérieurs à k.
Les minimums d'un graphe F (noté M(F)) est un graphe unissant tous les minimums locaux de F.

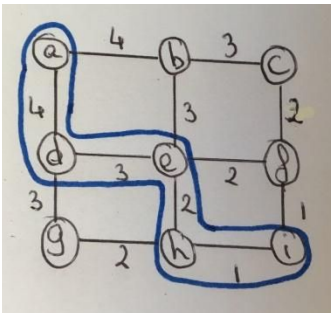
Exemple :



En bleu, les minimas de F

Un <u>« chemin descendant »</u> est une suite d'arcs reliés entre eux (deux arcs sont reliés entre eux s'ils possèdent un sommet en commun) dont le poids est décroissant.

Exemple :



En bleu, un chemin descendant de a à i. il n'existe pas de chemin descendant de i à a

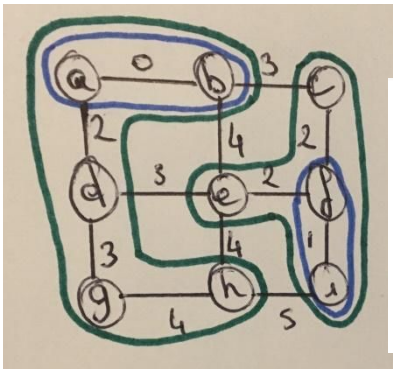
Dans un « edge-weighted graph », (graphe dans lequel les arcs ont un poids (ou une altitude), une ligne des partage des eaux est une suite d'arcs de telle sorte à ce qu'on puisse trouver de part et d'autre deux chemins descendants, menant à deux minimum locaux de F distincts.

Comment peut-on alors déterminer algorithmiquement ces lignes de partage des eaux (ou « watershed cuts ») ?

Notion d'extension d'un graphe : On dit que « Y est une extension de X » si X est inclus dans Y, si Y est connexe, et si tous les sommets et tous les arcs de X sont aussi dans Y.

Si l'on crée des extensions des minimums de F, de telle sorte à ce que chaque sommet de F soient dans une extension d'un des minimums de F un arc reliant deux sommets appartenant à deux extensions différentes est un arc de « coupure ». Intuitivement, il existe au moins une « combinaisons d'extension » de sorte à ce que ces coupures soient des « watershed cuts »

Exemple :

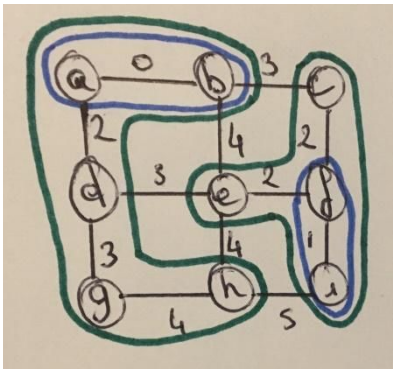


Dans cet exemple, en bleu sont entourés les minimums et en vert une extension possible des deux minimums tels que chaque sommet appartiennent a une extension. On remarque que la coupure est un watershed.

Notion de Foret : En théorie, une forêt est un graphe comportant plusieurs arbres (qui ne peuvent pas contenir de cycle). Cependant, comme les minimas de notre graphe F peuvent eux-même contenir des cycles, on dit qu'une forêt est l'union de plusieurs sous-graphes de F connexe dans lequel seuls les minimas de F peuvent contenir des cycles.

Ainsi, une MSF (pour « minimum spanning forest ») relative aux minimums est la forêt dont la somme des poids des arcs la constituant est la plus petite, et dans laquelle chaque arbre est une extension d'un minima.

Dans l'article, on démontre que si l'on trouve la coupure d'un MSF, celle-ci est elle-même une ligne de partage des eaux. Trouver une ligne de partage des eaux revient donc à trouver une MSF.



Dans cet exemple, on remarque que cette combinaison d'extension est non seulement une MSF, mais également que la coupure correspond a une watershed cut.

Pour trouver cette MSF, il y a trois méthodes. Chaque méthode sera utilisée pour un cas différent. La première peut être utilisée dans le cas de calculs « en parallèle », la seconde ne peut être exécutée en parallèle, mais est en complexité linéaire (O(n)).

Avant d'expliquer chaque méthode, quelques notions sont à expliquer.

Tout d'abord, nous attribuons à chaque sommet de F la valeur du plus petit arc adjacent à ce sommet. On note $F^-(x)$ la valeur ainsi attribuée au sommet x.

Notion de « lowering d'arc ». Le lowering d'un arc u(x,y) correspond au remplacement du poids de u par le minimum de $F^-(x)$ et de $F^-(y)$. En d'autres termes u prendra le poids de l'arc adjacent ayant le poids minimal.

Intuitivement, il est compréhensible qu'on ne puisse pas faire de « lowering » de tous les arcs, ça ne mènerait à rien. Ainsi, pour sélectionner les arcs à « abaisser » nous allons vérifier une propriété sur chaque arc. Si cette propriété retourne « True » sur l'arc u, l'abaissement sera effectué sur u, dans le cas contraire, il ne sera pas fait.

A – Première propriété : Border-thinning

Grace à l'ajout des poids sur les sommets, on peut catégoriser les arcs en 3 sortes.

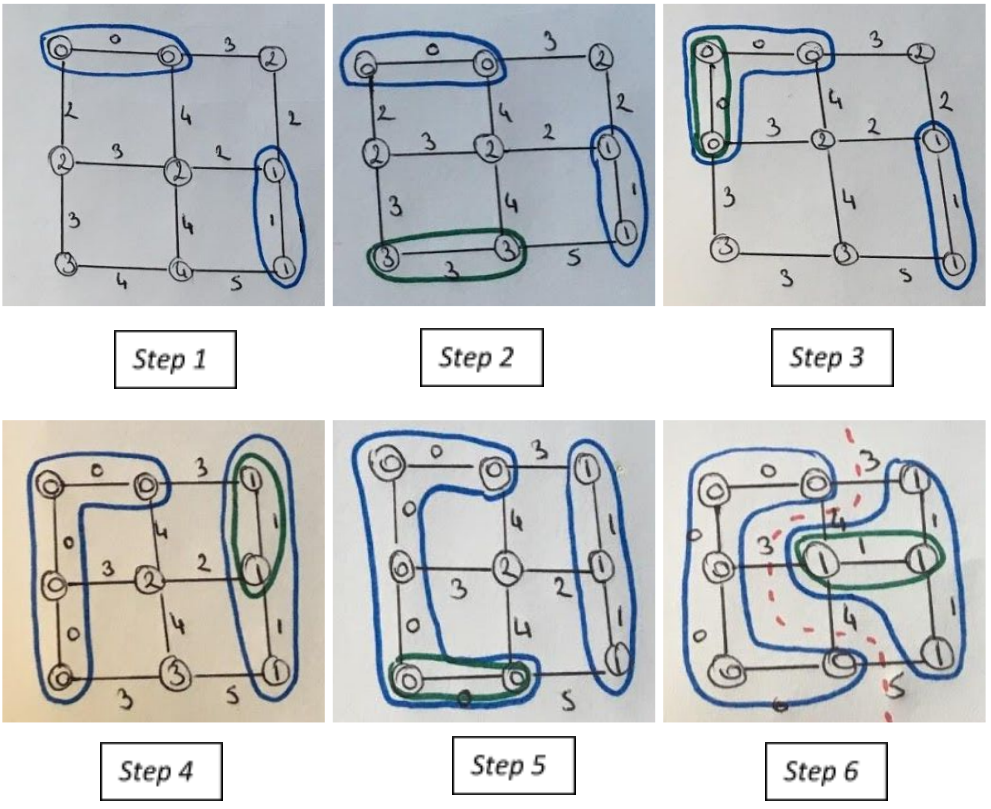
Les arcs « séparant localement » sont les arcs dont le poids est supérieur aux deux sommets qu'ils relient.
Les arcs « internes » sont égaux aux deux sommets qu'ils relient. C'est par exemple le cas des arcs à l'intérieur des minimums de F

Les arcs de bordures sont les arcs qui sont égaux à l'un des sommets qu'ils relient, et supérieur à l'autre sommet.

Puisque le poids des sommets est le minimum des arcs adjacents à ce sommet, un arc ne peut pas être inférieur au poids d'un sommet adjacent. Les 3 catégories ci-dessus couvrent donc bien l'entièreté des possibilités.

La propriété de lowering est donc la suivante. Un arc devra être « abaissé » si et seulement si il appartient à la catégorie « arc de bordure ». Tant qu'il existe au moins un arc qui respecte cette propriété (c'est-à-dire, tant qu'il existe un arc appartenant à la catégorie « arc de bordure ») , le processus de lowering devra être poursuivi. Ce processus est appelé le *B-Thinning* (pour *Border-Thinning*) et lorsqu'il sera terminée, graphe obtenu, qui ne pourra plus être abaissé d'avantage, est un graphe dont l'union des minimum est appelé *B-Kernel*. (l'abaissement successif aura abaissé les poids d'un grand nombre d'arc de telle sorte à ce que tous les sommets fassent partie d'un B-Kernel). Le B-Kernel est tout simplement le MSF de F.

Exemple en étapes : pour chaque étape : en bleu les minimas, en vert, l'arc de bordure « abaissé »



Cette méthode est intéressante du fait qu'elle est executée en local (sur chaque arc, sans se soucier du reste). Elle peut ainsi être exécutée en parallèle.

Cependant, si l'exécution en parallèle n'est pas désirée, on se rend compte que la complexité de l'algorithme est est bien trop élevée ($O(E^2)$). E (pour « edge ») étant le nombre d'arc du graphe.

B – Deuxième propriété : Minimum-thinning

Cette propriété est un cas particulier de B-Thinning. Si l'on exécute « à la main » un B-Thinning, on se rend compte très rapidement que si on vérifie la propriété (d'arc de bordure) seulement sur les arcs adjacents aux minimums de F, le lowering ne sera effectué qu'au plus 1 fois sur chaque arc...

Par exemple, sur l'exemple précèdent (B-Thinning), on comprend rapidement que l'étape deux est de trop, qu'elle ne sert pas réellement a quelque chose, puisque l'arc est a nouveau modifié à l'étape 5...

Ainsi, l'efficacité du *B-thinning* est augmentée est l'algorithme ne parcourra qu'une seule fois tous les arcs de F. On obtiendra donc un algorithme en complexité linéaire, pour des résultats similaires au *B-Thinning.*

Cet amincissement s'inscrit donc parfaitement dans une optique de traitement séquentiel de l'image. En revanche l'amincissement B-Thinning est un amincissement local, et peut être utilisé en parallèle, c'est celui que nous préférerons et que nous utiliserons au cours de ce projet.

Par ailleurs, un enjeu majeur du projet est de réaliser cet algorithme d'amincissement "par bloc". En effet, si nous avons une image de taille énorme, ne rentrant pas en un seul bloc dans la mémoire vive, nous devrons traiter l'images en plusieurs parties d'images (plusieurs "blocs") pour ensuite fusionner tous les morceaux.

Mais n'oublions pas que nous travaillons dans un edge-weighted graphe. En effet, chaque bloc est lié avec des arcs ayant un poids, et ce poid est "mandatory" au cours du processus de fusion des blocs.

Tout d'abord, ce que nous remarquons à la fin des deux processus d'amincissement cités précédemment, c'est qu'à la fin de celui ci, il n'y aura plus aucun arc de bordure. Il ne restera plus que des arcs internes (qui constitueront les minimas) ainsi que des arcs séparants.

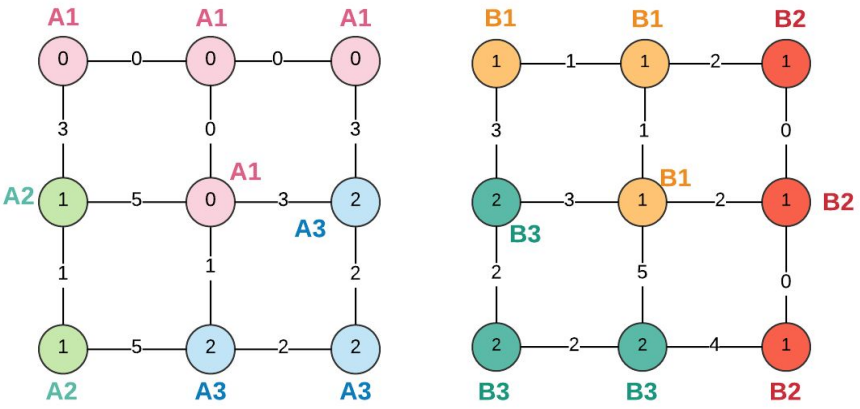
A quoi peut nous servir cet information triviale ? Eh bien, au cours du processus de fusion des deux blocs (qui ont, je le rappelle, déjà été amincis de leur côté) nous allons abaisser uniquement les arcs de bordures appartenant à la coupure entre les deux blocs. Mais ce n'est pas tout ! Une fois que l'arc a été abaissé, il faut remettre à jour les deux bloc et ré-effectuer un B-Thinning sur les deux blocs et cela, pour être sur qu'un minimum dépassant d'un bloc soit sur un même plateau ! (C'est à dire : que chaque arc sur un même minima possède le même poid, peu importe le bloc dans lequel il se trouve).

c. Étiquetage des composantes connexes

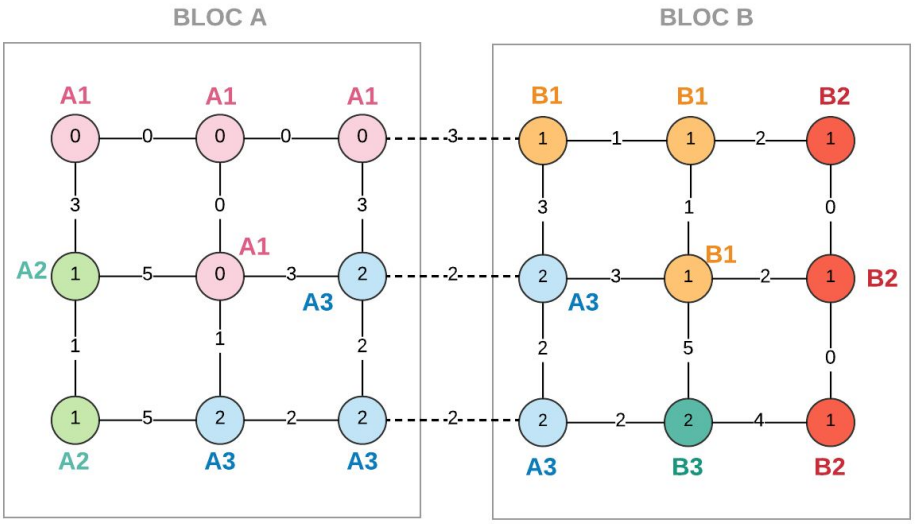
Une composante connexe d'un graphe est un sous-graphe dont, pour tout sommet appartenant à cet ensemble, il existe un chemin permettant d'aller à tout autre noeud de ce sous-graphe.

Soit 2 blocs pour lesquels on a calculé indépendamment l'un de l'autre les composantes connexes. Dans le cas du B-Thinning, les composantes connexes sont les minimas étendus.

Les labels sont uniques.

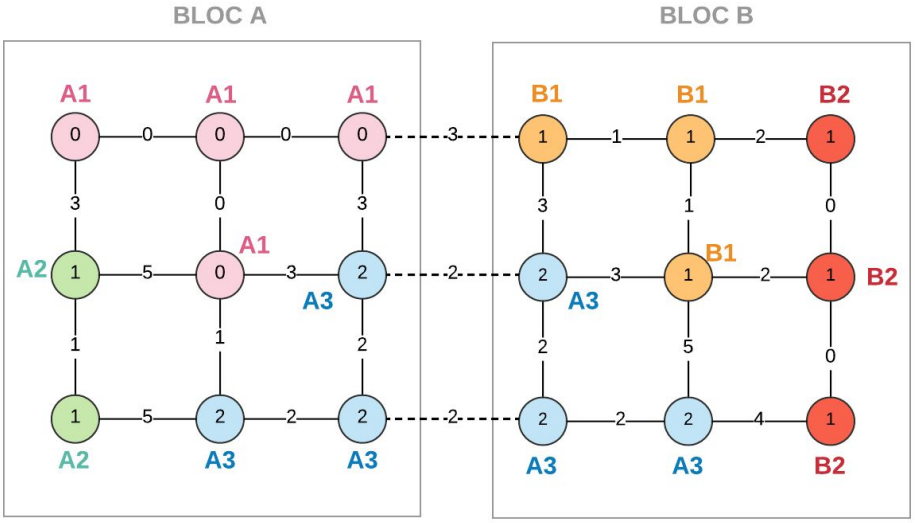


On cherche à merger les deux blocs. On merge les labels des arcs internes de la coupure. Si l'arc est un acc interne, alors le label de l'un des deux noeuds adjacent à l'arc prend la valeur de l'autre noeud adjacent.

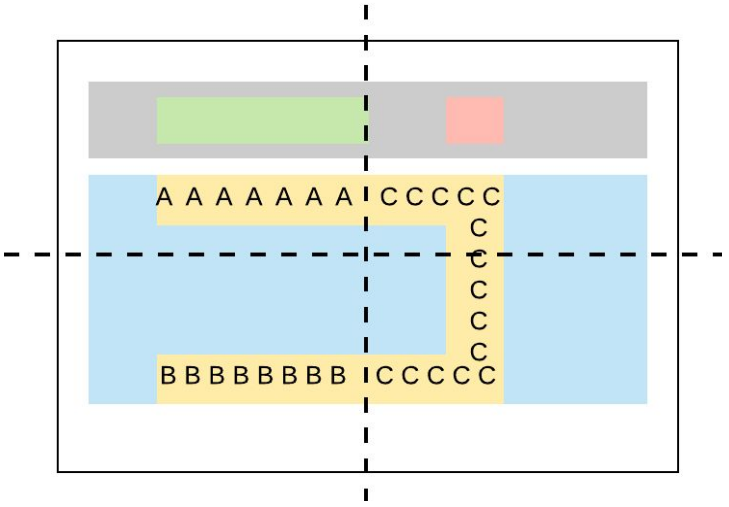


On mémorise les changements dans une table, puis on parcours le bloc du label qui a été

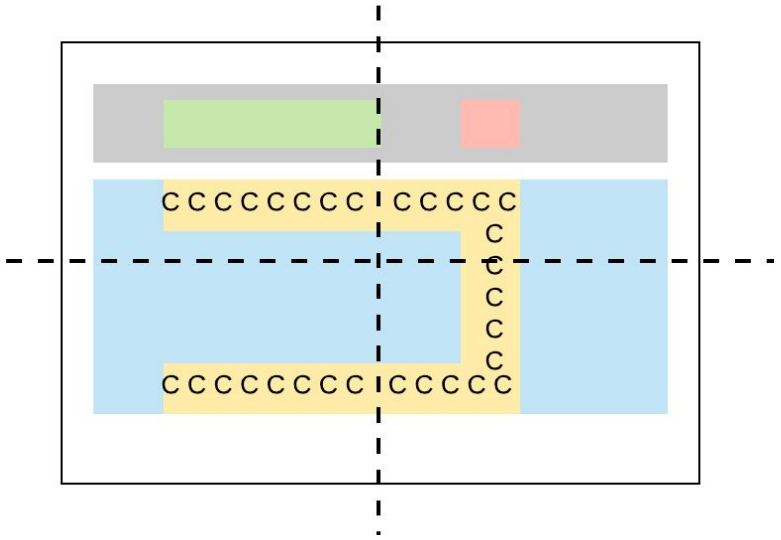
modifié. On actualise les labels avec les informations contenues dans la table. Dans notre cas, on a retenu dans la table qu'il fallait modifier tous les B3 en A3 dans le bloc B.



Cette technique, bien que peu efficace en terme de temps de calcul car on ré-actualise la totalité des labels, fonctionne toujours. Par exemple, dans l'exemple ci-dessous, le merging du bloc supérieur gauche et inférieur droit aurait montré que les labels A et C sont différents.



Cependant, on constate, en les fusionnant avec les deux autres parties, que les labels A, B et C forment un seul et même label et donc une seule et même composante connexe :


CCCCCCCCC CCCCC
C
C
C
C
C
CCCCCCCCC CCCCC

d. QBT

A: Obtenir QBT

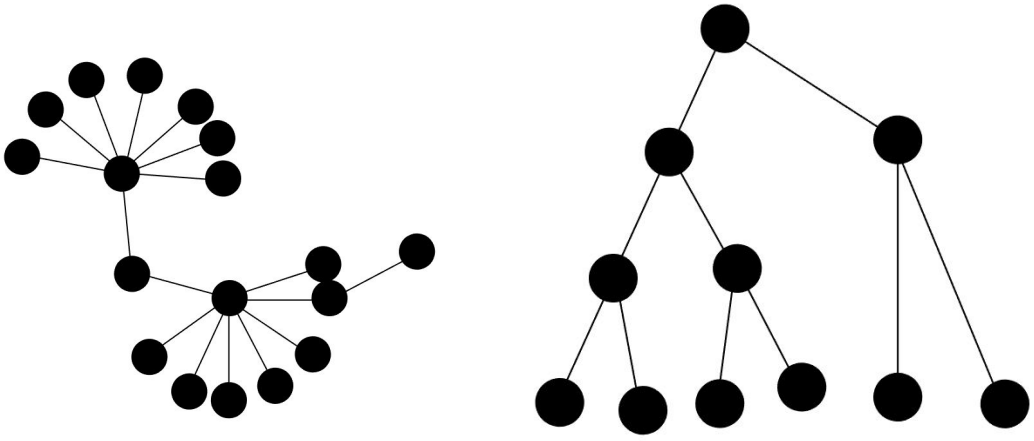
Un autre moyen d'obtenir une ligne de partage des eaux est de construire un arbre représentant l'image appelé QBT. La méthode pour produire ce graphe sera présenté ici.

QBT permet principalement quatres choses:

- Sa complexité est quasi linéaire, il est donc très rapide
- Il est possible d'obtenir la hiérarchie des zones quasi-plates à partir de cet arbre linéairement
- Il est possible de trouver les arêtes appartenant à la ligne de partage des eaux en temps linéaire
- Il est possible de calculer QBT par blocs

Tout d'abord, il est utile de réduire la dimensionnalité du problème. Nous voulons éliminer le plus d'arêtes que possible tout en conservant les informations nécessaires pour la suite. Cela est fait en utilisant Kruskal. Mais cet algorithme présente un problème de taille, sa complexité trop importante.
Heureusement pour nous, il est possible de calculer QBT de deux manières:

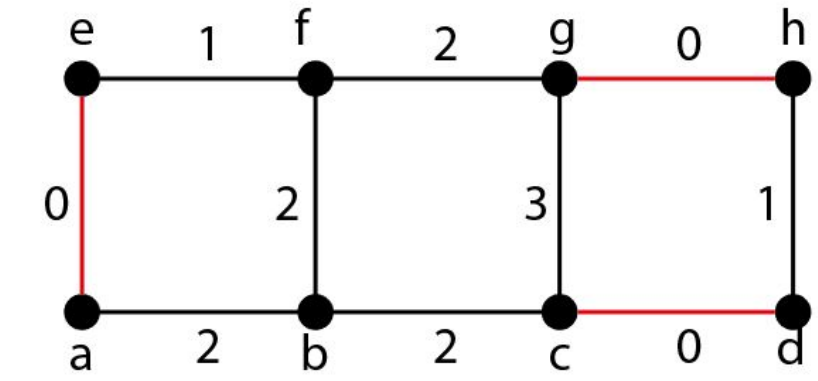
- A l'aide de l'algorithme Union-Find, nous obtenons un arbre contenant des informations utiles, mais son calcul est relativement lent. O(V^2)
- En utilisant la structure de données Tarjan Union-Find, l'arbre obtenu n'est pas utile. Son calcul est très rapide. ~O(n)



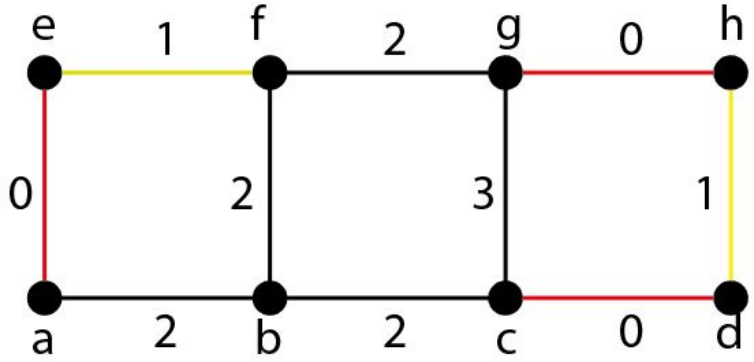
A gauche la structure Tarjan Union-find, à droite QBT. Nous pouvons voir qu'il est difficile d'extraire des informations du premier graphe.

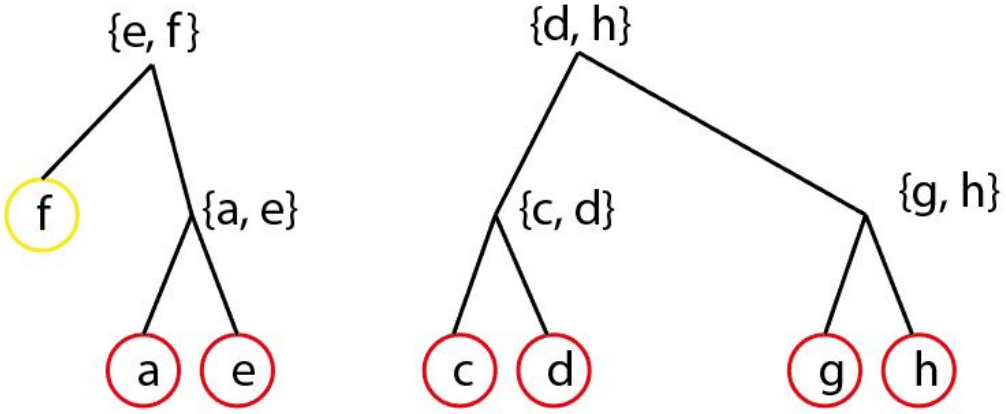
Nous voulons un algorithme rapide et donnant un arbre utile. Nous combinons donc les deux algorithmes pour ne garder que le meilleur des deux mondes, au prix d'un encombrement mémoire plus important. Deux arbres vont être créés, dont QBT, qui nous est utile.

B) Exemple de calcul de QBT

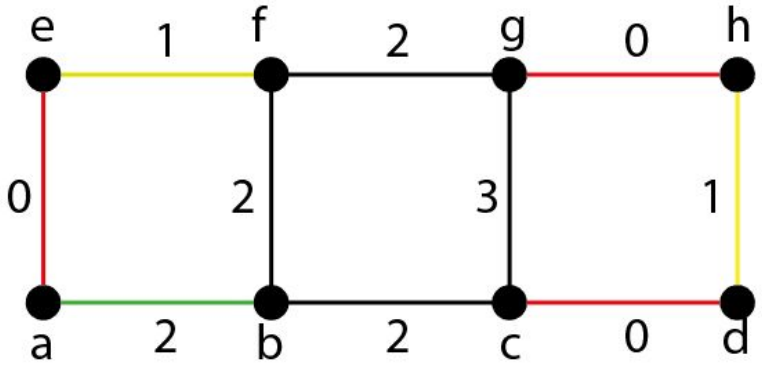


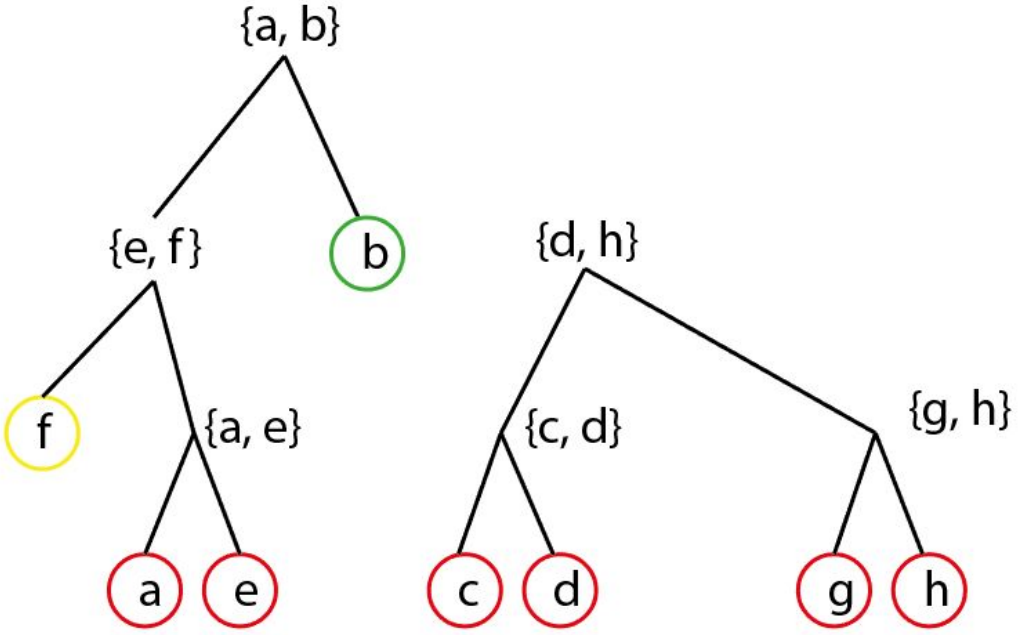
1) Ci dessus, on applique kruskall, des nodes sont créés pour chaque noeud du graphe si celui-ci n'est pas déjà présent dans le graphe.



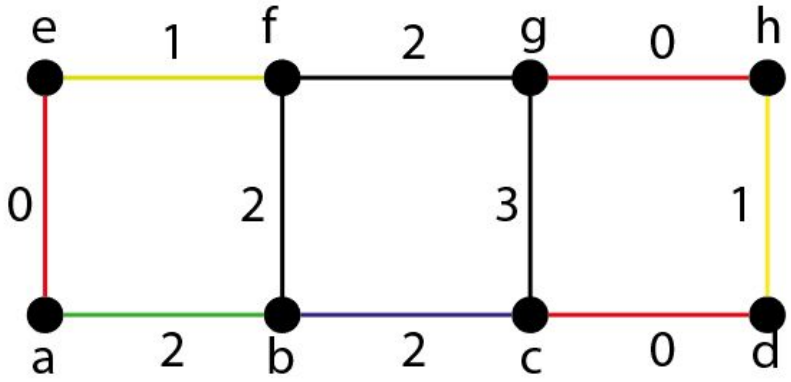


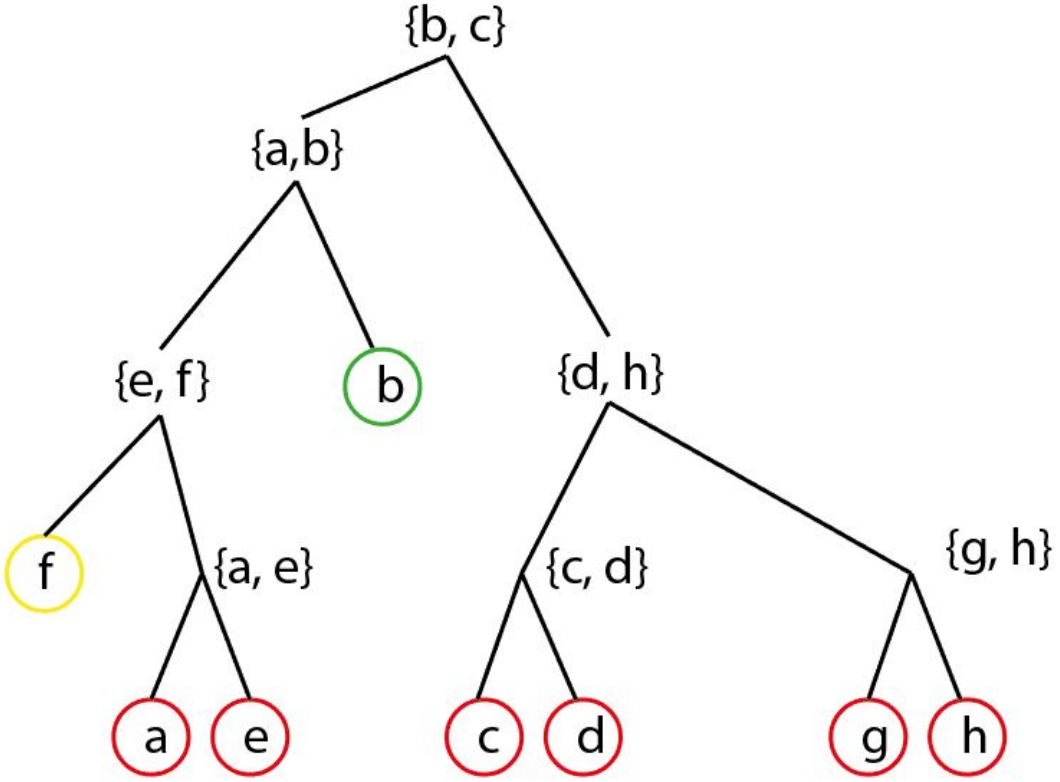
2) Ci dessus, on ajoute les arêtes {e, f} et {d, h} . f n'étant pas dans notre arbre, nous l'ajoutons. d et h étant présents dans QBT, nous faisons en sorte que {d, h} soit le père des arbres contenant d et h.


e
1
f
2
g
0
h
0
2
3
1
a
2
b
2
c
0
d

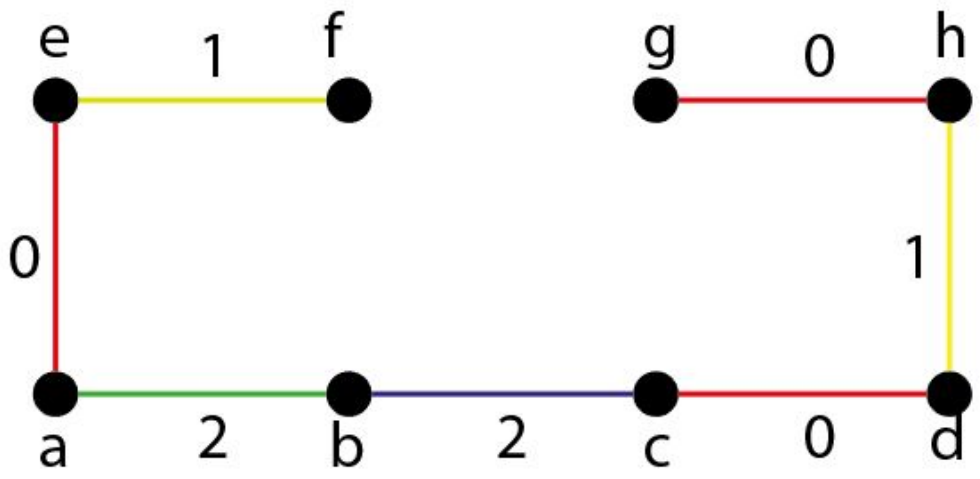

{a, b}
{e, f}
b
f
{a, e}
a
e
{d, h}
{c, d}
{g, h}
c
d
g
h

3) Même chose pour {a, b}.





3) {b, c} est ajouté à QBT, il est impossible de rajouter une arête sans créer de cycles, nous nous stoppons donc.



4) Nous voyons que beaucoup d'arêtes ont étés enlevées de notre arbre QBT.

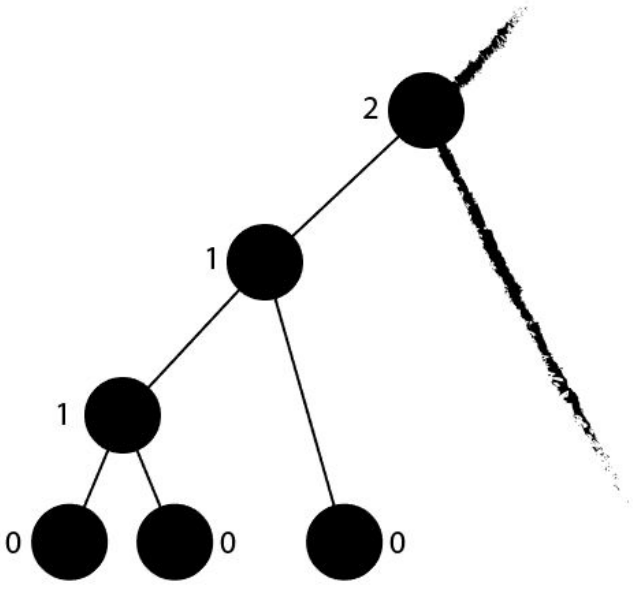
C: Obtenir une hiérarchie des zones quasi plates

La hiérarchie des zones quasi plates nous permet d'obtenir une ligne de partage des eaux hiérarchique, mais nous n'avons pas encore implémentés la partie hiérarchique.

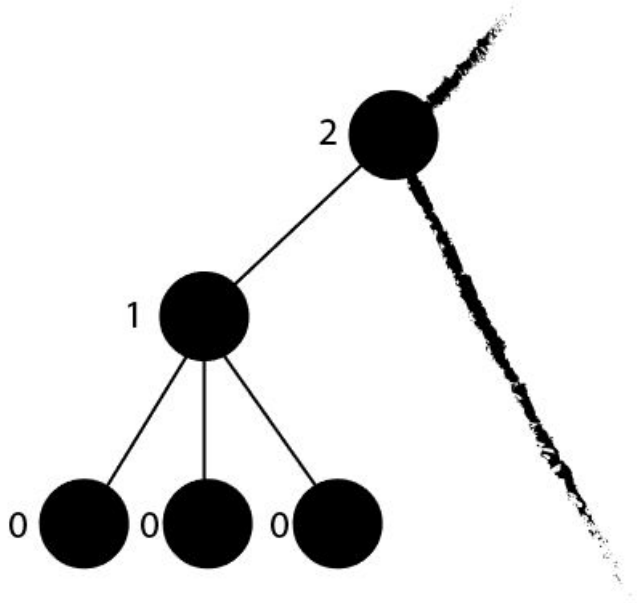
Pour calculer cet arbre, il faut savoir qu'à chaque noeud de QBT, un poids est assigné (0 pour les feuilles, …). Il suffit alors de "réunir" ces noeuds de même poids.

Voici un exemple:

Partie de QBT

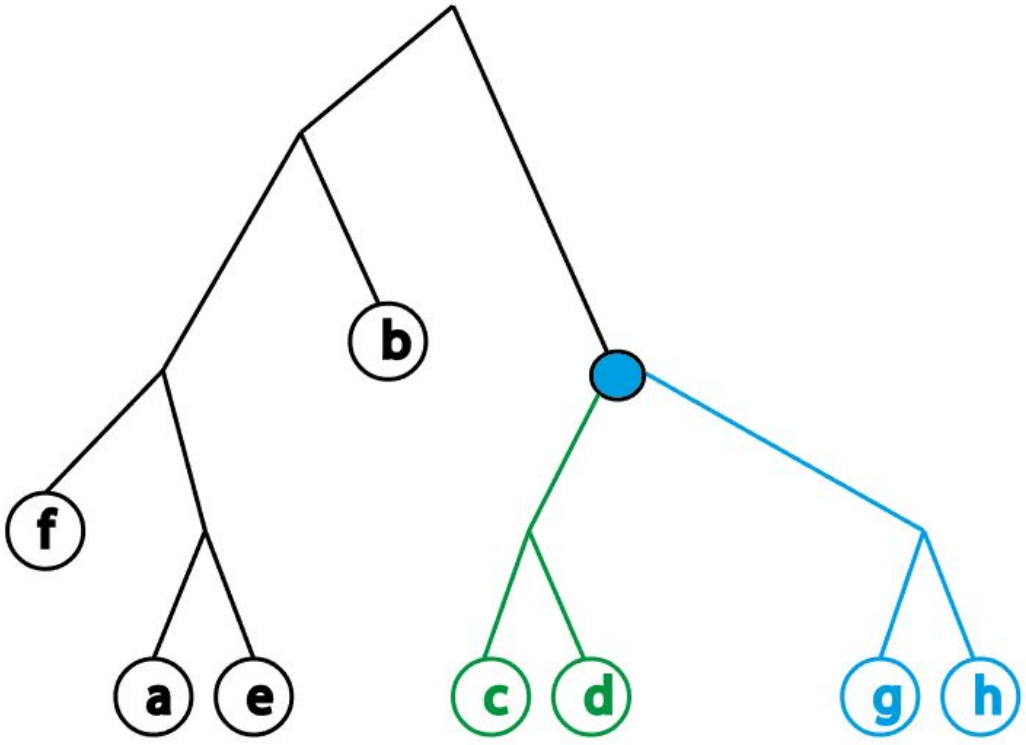


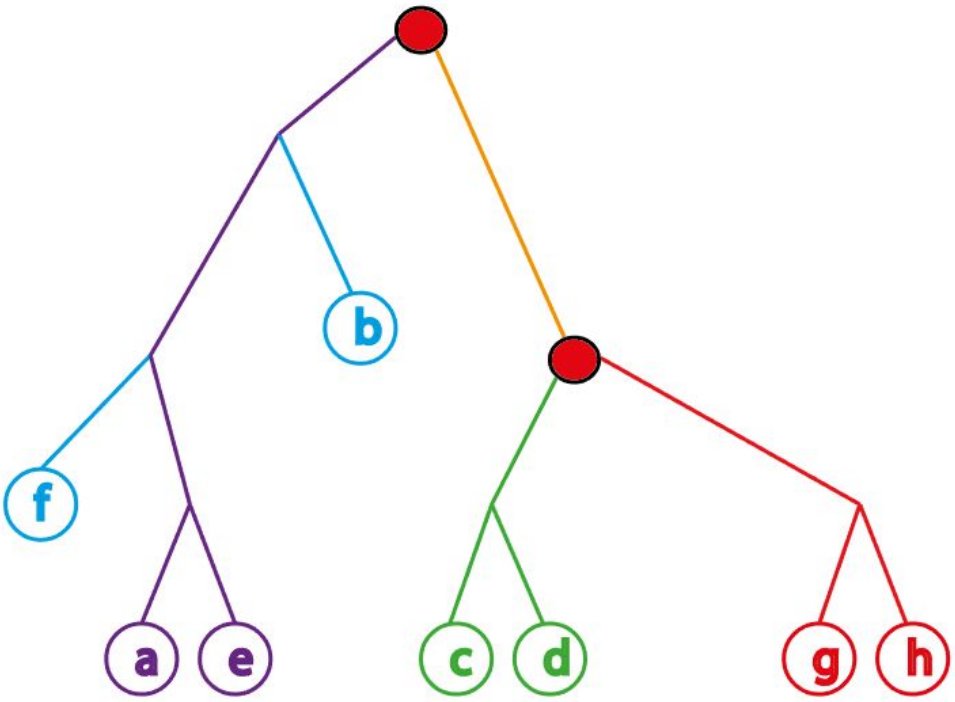
Après réunification des nodes de même poids:



D) Obtenir un wateshed par flooding

Pour obtenir ce watershed, nous partons des minimas ⇔ “tout en bas de notre arbre”, puis nous “augmentons le niveau de l’eau” progressivement. Ici, les groupes C,D et G,H sont deux bassins différents. Ces bassins se rejoignent sur une arête du watershed cut. Nous remontons donc le graphe de cette manière.



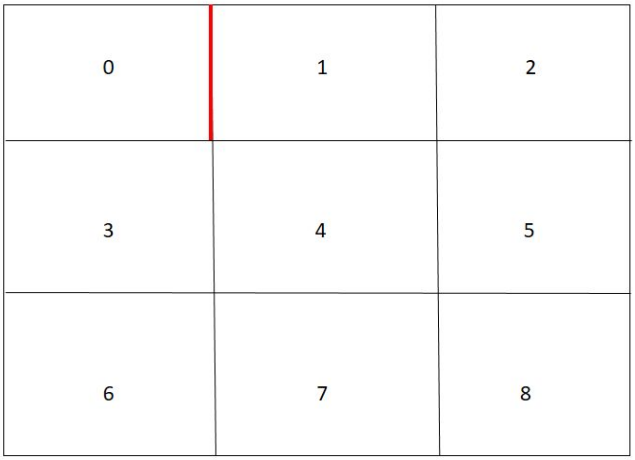


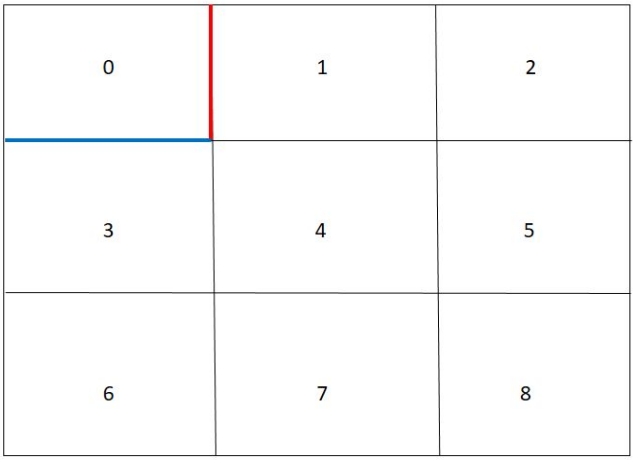
Ici en rouge les deux arêtes du watershed cut du graphe obtenu via QBT

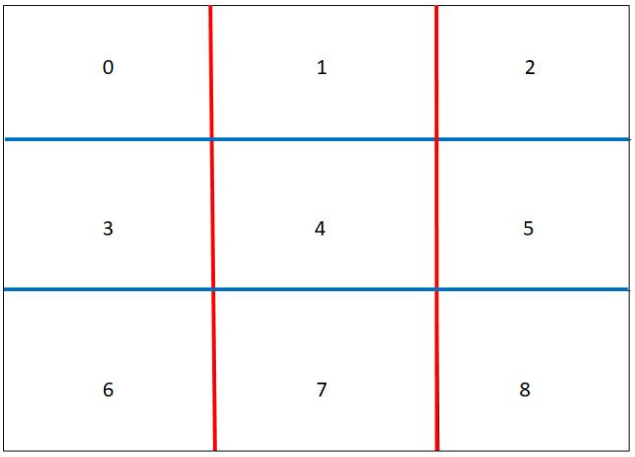
e. Merging of the blocks

After producing the hierarchy of each block by using the QBT method, we are now ready to merge the blocks together.

We created a process called "Server" which manages the merging of all blocks. It will initiate the merging process by merging block 0 with its neighbours on the right and on the bottom. After, we do the same for block 1 and its neighbours and so on. We do this merging one by one until all blocks are merged together.

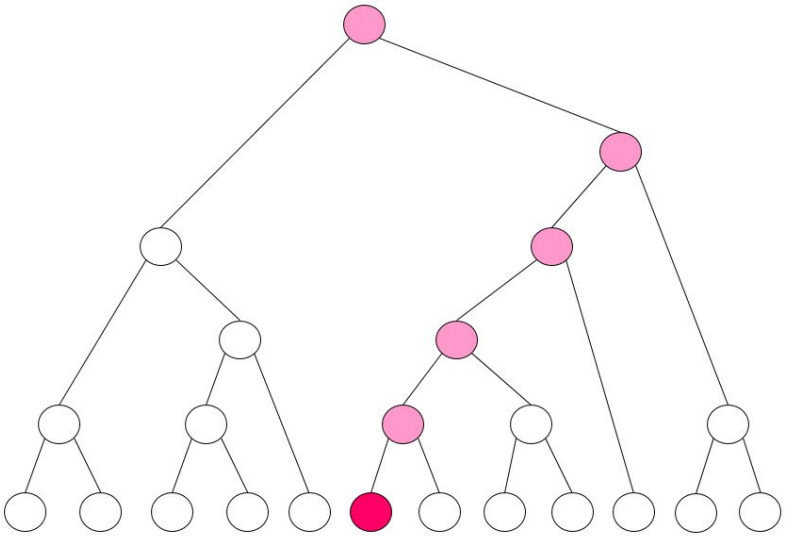




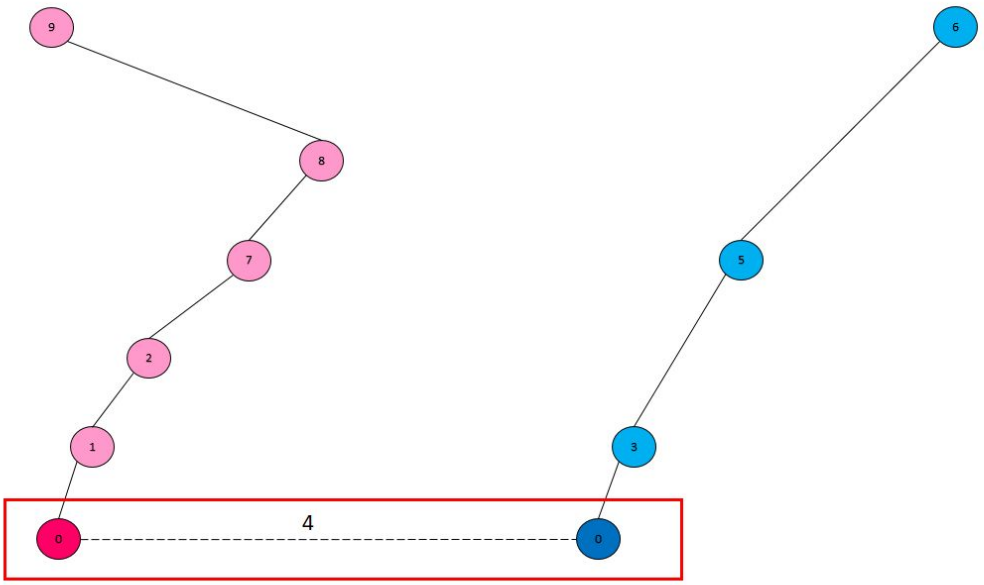


The merging of two blocks happens on their touching border. We select two nodes, x and y, one of each block, that are connected by the border we are now processing. For each of the nodes, we generate the their relative border tree (BT) and give both of these to the "Server".

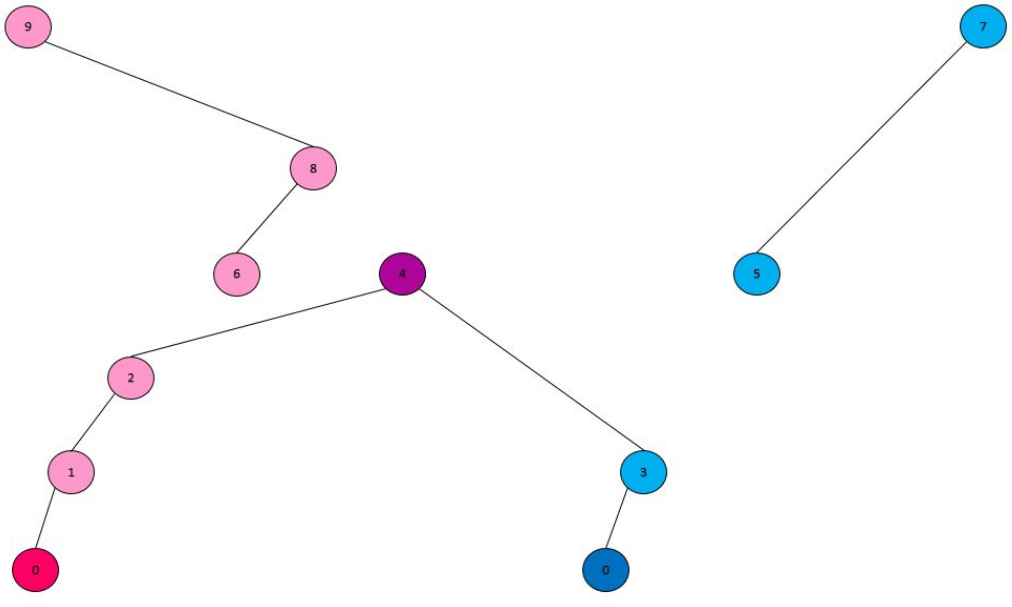
The boundary tree is the subtree of the hierarchy of the block which includes only the leaf we are searching, in this case, x or y, and its parent, and the parent of its parent. We can see in the image below the construction of the boundary tree of the node in pink.



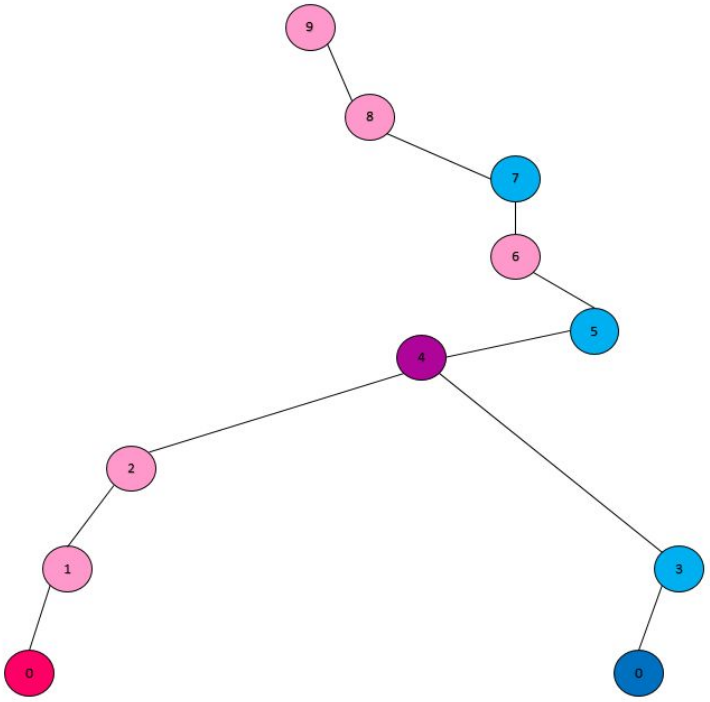
We obtain both of the boundary trees we need and the server has the information of the altitude of the merging. We see in the image below two boundary trees (blue and pink) and the values inside each node represents its altitude. The altitude of the merging of the two nodes will be four.



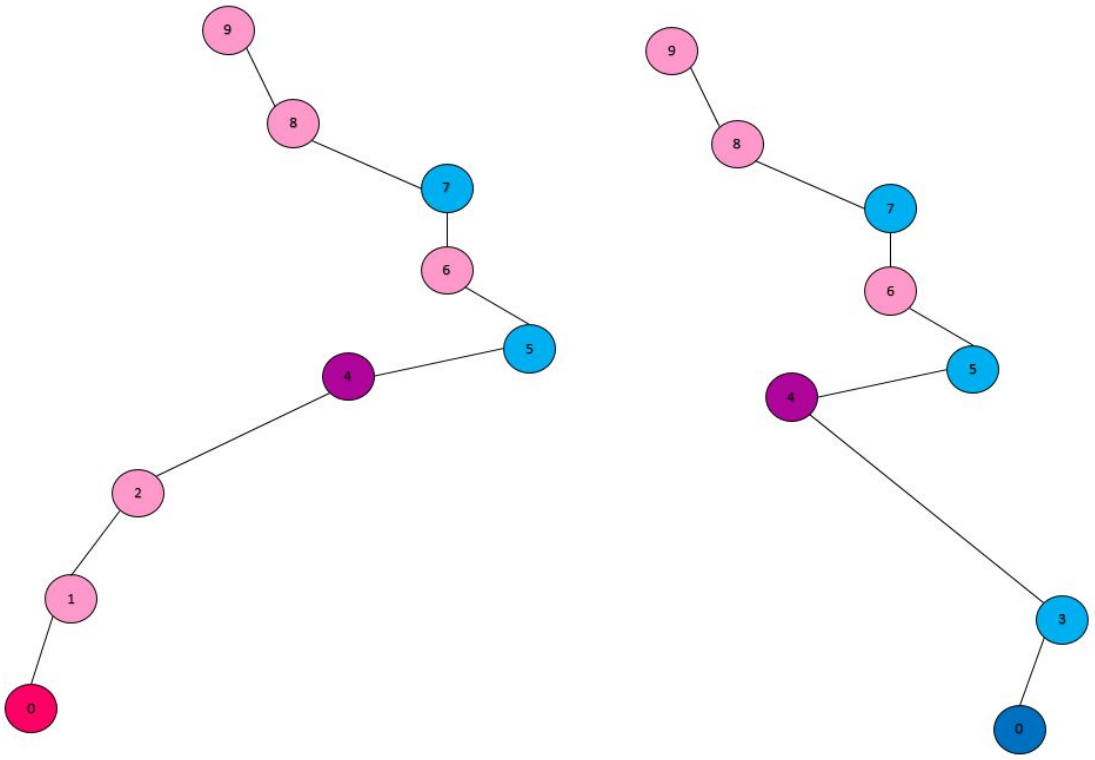
The first part of the merging is to find where the new node is going to fit. When we reach an altitude which is higher than the altitude of the merging, we select its children to be the children of our new node. The new node is represented in purple and its altitude is four.



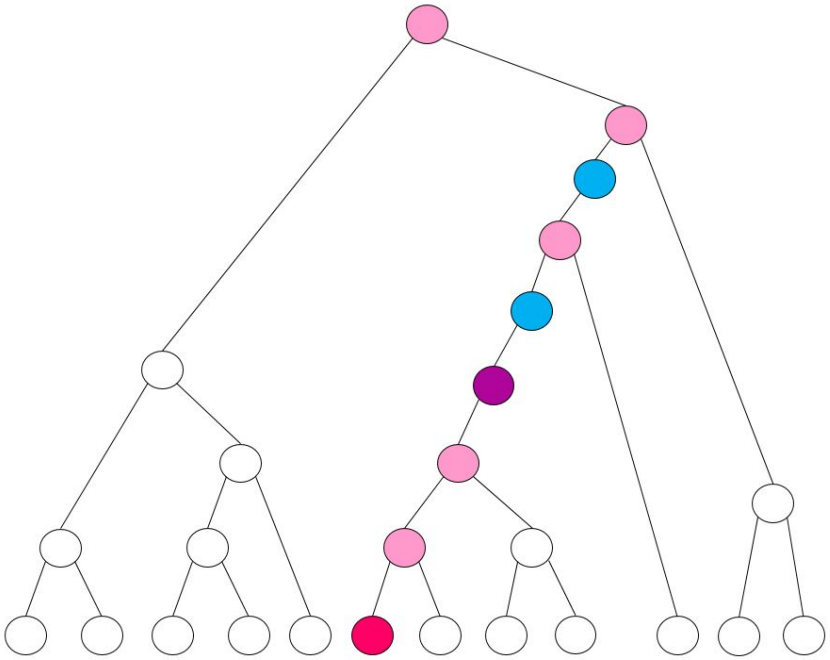
After that we merging the remaining nodes one by one in order of its altitude until we reach a new root. The final result MT will look like this:



With this result, we now generate a new boundary tree for each of the starting nodes, but now they are related to the previous result tree MT. We can see both of the boundary trees below.



Now, we have to send this information back to the original blocks in order to update their trees. Then, the last part of the merging algorithm is the update of the blocks. We send the new boundary tree to the original blocks and we update according to the changes detected. If we take the original tree from the pink node, we can do the updating using this new boundary tree and the result will be as shown below.



We repeat this process for each edge in the border until we have all of the borders covered. In the end each block will have in its tree all the information that it needs.

2. Description des tâches

3. Management

Avec l'aide de github et d'une gestion des issues ainsi que des plannings de préparations, nous avons pus, semaines par semaines gérer les innovations à intégrer au projet. De plus, nous avons mis en place un script Qualité permettant de détecter les erreurs dans le code via des tests unitaires.

De plus, nous nous voyons chaque semaine via discord ou en vrai, pour travailler en équipe (peer programming).

C. Conclusion

Pour conclure, on peut dire que ce projet nous a permis d'acquérir des connaissances en traitement d'image et en théorie des graphes. Par ailleurs c'est un projet particulièrement intéressant, parce que l'algorithme que nous produisons est innovant.

A ce stade du projet, nous sommes capables de :

- segmenter une image par amincissement itéré ;
- Fusionner deux "blocs d'images" segmentés séparément, et labelliser leur composantes connexes ;
- Générer l'arbre QBT par bloc ;
- Faire la hiérarchie des zones quasi-plates (QFZ).

D'un point de vue organisationnel, nous avons divisé les tâches par binôme :

- Jules et Stela sur la partie "Hiérarchisation et Fusion" ;
- Cécile et Edouard sur la partie "Ligne de partage des eaux par blocs".

Au semestre deux, un membre de l'équipe ne fera plus partie de l'équipe de projet. Il faudra donc d'ici là s'assurer que les connaissances qu'il a acquises soient bien transmises aux membres actuels, de telle sorte à ce que les nouveaux arrivants puissent intégrer le projet sereinement.

Sources

- [1] J. Cousty, L. Najman, Y. Kenmochi, S. Guimarães, *Hierarchical segmentations with graphs: quasi-flatzones, minimum spanning trees, and saliency maps*, Journal of Mathematical Imaging and Vision, Springer Verlag, 2017.

- [2] J. Cousty, G. Bertrand, L. Najman, M. Couprie, *Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Institute of Electrical and Electronics Engineers, 2009.

- [3]L. Najman, J. Cousty, B. Perret, *Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs*, C.L. Luengo Hendriks, G. Borgefors, R. Strand. International Symposium on Mathematical Morphology, May 2013, Uppsala, Sweden.