TP 9 - Graphe d'une image et algorithme de binarisation

Des algorithmes sur les graphes ont été appliqués avec succès pour résoudre des problèmes de vision par ordinateur et de traitement d'images. Nous nous intéressons à l'application des algorithmes de coupes de graphes pour segmenter les images de niveau de gris en deux parties : les pixels sombres d'une part, et les pixels clairs d'autre part, sachant que la notion de clarté d'un pixel est relative à son contexte dans l'image.

D'une image à un graphe et à un réseau

On peut considérer les pixels d'une image comme les nœuds d'un graphe orienté où chaque pixel est relié par un arc sortant à ses 4 pixels voisins. Comme les pixels sont organisés en une grille rectangulaire, il est possible de coder le graphe de l'image comme un tableau 2D où l'on sait où trouver les voisins de chaque nœud sans avoir à coder cette relation de voisinage. Ainsi, on sait que le pixel $(3, 7)$ est voisin des pixels $(3, 6)$, $(3, 8)$, $(2, 7)$ et $(4, 7)$, par simple calcul sur les coordonnées du pixel. Dans le tableau modélisant l'image, chaque pixel stockera son information d'intensité lumineuse mais aussi les valuations de chacun de ses 4 arcs sortants (vers les voisins Ouest, Est, Nord et Sud respectivement). Si le pixel est sur le bord de l'image, il aura moins de voisins, et les valuations d'arcs correspondantes seront alors sans importance. Entre les pixels p et q, il y a l'arc (p, q) et l'arc (q, p).

Nous allons transformer le graphe de l'image en un réseau dans lequel nous établirons un flot alimenté par une source placée au dessus de l'image et s'écoulant vers un puit placée en dessous de l'image (voir Figure 1). En plus des nœuds correspondant aux pixels, votre structure de graphe sera donc également composée de 2 nœuds de nature différente, appelés la source S et le puit P. Il existe un arc entre S et l'ensemble des pixels, et un arc entre chaque pixel et P. Les valuations de ces arcs supplémentaires pourront également être stockées dans les pixels, sous forme de propriétés supplémentaires, si bien qu'en définitive le graphe pourra intégralement être codé sous forme du tableau, de son nombre de ligne L et de son nombre de colonnes C.

Pour l'arc orienté (p, q) entre les nœuds p et q, il y aura deux valuations positives, sa **capacité** $Cap(p, q)$ et son **flot** $F(p, q)$, avec la contrainte que $F(p, q)$ doit être inférieure à la capacité $Cap(p, q)$. En tout nœud p du graphe, la somme des flux entrants doit être égale à la somme des flux sortants (loi de conservation du flot), sauf pour la source S et le puit P pour lesquels le flot sortant de S doit correspondre au flot entrant dans P.

$$F(p, P) + \sum_{q \in voisinage(p)} F(p, q) = F(S, p) + \sum_{r \in voisinage(p)} F(r, p)$$

A l'initialisation le flot est nul dans chacun des arcs, et nous verrons dans la section suivante comment calculer les capacités des arcs à partir des intensités lumineuses.

Travail à faire : On se propose de mettre en œuvre un module graphe d'une image, en utilisant uniquement un tableau 1D de taille $L \times C$. Le pixel $p = (i, j)$ $(0 \leq i \leq L - 1,\ 0 \leq j \leq C - 1)$ est situé dans la case $i * C + j$ du tableau, son intensité lumineuse notée $I(p)$ ou $I(i, j)$ est une valeur entière comprises entre 0 et 255. Les voisins d'un nœud (i, j) sont les nœuds situés à sa gauche $(i, j - 1)$, à sa droite $(i, j + 1)$, au dessus $(i - 1, j)$ et en dessous $(i + 1, j)$ de lui (on les désignera sous l'appellation voisins Ouest, Est, Nord et Sud). Les pixels stockent les valuations F et Cap

des arcs sortants vers leurs voisins, ainsi que les valuations de l'arc entrant issu de la source S et de l'arc sortant allant vers le puit P.

Parmi les procédures d'initialisation d'une Image, en prévoir une pour charger un graphe sauvegardé dans un fichier (supprimer les commentaires pour une lecture de fichier plus rapide, et regarder les exemples fournis sur la page web du cours pour des rappels sur les lectures de fichiers). A l'initialisation, les flots dans les arcs sont mis à 0 et nous verrons plus tard comment établir leur capacité.

Format de stockage d'une Image dans un fichier (format pgm ascii) :

```
P2       # PGM ASCII
15 24    # Dimensions du graphe C (largeur) et L (hauteur)
255     # Intensité maximale
12 34 5 .... //Intensités des C*L pixels listés ligne par ligne
               // (de gauche à droite et du haut vers le bas)
```

En plus des opérations classiques d'initialisation, d'affectation et de testament du graphe, prévoir des fonctions d'accès à l'indice global d'un nœud positionné sur la ligne i et la colonne j, d'accès à l'indice global du voisin Nord (resp. Sud, Est et Ouest) d'un nœud (avec pour précondition que ce voisin existe), d'accès aux propriétés d'un pixel (intensité, flot et capacité des arcs vers les voisins) ainsi que des procédures de modification du flot dans un arc. Prévoir également une procédure d'affichage de la grille ou de sauvegarde dans un fichier. Un nœud peut être désigné par son indice global dans le tableau ou par ses coordonnées de ligne et de colonne (i, j). Un arc est désigné par le nœud dont il est issu ainsi que par une direction.

Binarisation de l'image par établissement du flot maximal dans le réseau

L'image à traiter peut être vue comme une version bruitée de l'image binarisée que l'on cherche à retrouver, et dans laquelle il n'y a plus de pixels gris, uniquement des pixels noirs ou blancs.

La capacité des arcs du réseau sont établies de manière à quantifier la similarité entre deux pixels. On mesure cette similarité à l'aide d'une Gaussienne d'écart-type σ où σ est un paramètre de l'approche interprétable comme le niveau de bruit présent dans l'image. Ainsi, la capacité d'un arc (p, q) entre deux pixels voisins p et q correspond à $\exp(-\frac{(I(p)-I(q))^2}{2\sigma^2})$ où σ est le bruit qui caractérise l'image. La capacité de l'arc entre la source S et un pixel p est égal à $-\alpha ln(\frac{255-I(p)}{255})$ où α est un autre paramètre du programme. De même la capacité de l'arc entre le pixel p et le puit P est égal à $-\alpha ln(\frac{I(p)}{255})$.

On se propose de binariser l'image à l'aide d'un algorithme dit de coupe de graphe. L'idée est de catégoriser chaque pixel de l'image de telle sorte qu'il soit associé soit à la source S (intensité 255), soit au puit T (intensité 0). Pour cela, on augmente itérativement le flot maximal dans le réseau associé à l'image. Pour augmenter le flot, **on cherche à établir un chemin** améliorant entre la source S et le puit P **en utilisant uniquement des arcs non saturés** i.e. des arcs pour lesquels le flot n'a pas encore atteint la capacité. Si un tel chemin existe, **on augmente le flot courant d'une même quantité sur tous les arcs composant le chemin** améliorant, moyennant une petite subtilité à bien prendre en compte : **la valeur de l'augmentation du flot s'obtient en recherchant la plus petite valeur de** $Cap(p, q) - F(p, q) + F(q, p)$ **sur les arcs du chemin améliorant**. Attention toutefois que **lorsqu'on augmente le flot** courant dans l'arc joignant deux pixels voisins p et q, **si on dépasse la capacité** $Cap(p, q)$ **il convient de reporter l'excédent en diminuant le flot dans l'arc inverse** joignant les pixels q et p. Cela permet de maintenir la loi de conservation du flot à chaque nœud. Ainsi si le flot courant est de 5 dans un arc de capacité 7, et qu'on veut l'incrémenter de 3, alors on établit le flot courant à 7 et on décrémente de 5+3-7 = 1 le flot dans l'arc inverse. Lorsqu'il n'y a plus de chemin améliorant

entre la source S et le puit P on dit que le flot est maximal.

Il existe un théorème établissant l'équivalence entre le flot maximum et la coupe de coût minimum dans un graphe. Cela signifie que si on coupe tous les arcs saturés et les arcs de flot nul, on obtient deux composantes dans le graphe, une composante composée de pixels clairs connectés à la source S et une composante de pixels sombres connectés au puit P. Cette coupe est minimale au sens où elle minimise la somme des coûts des arcs coupés pour partionner le graphe en deux parties. Au démarrage de l'algorithme, si on veut établir des contraintes sur l'affectation d'un pixel donné à la zone des pixels clairs ou des pixels foncés, il suffit de ne le connecter qu'à la source S ou qu'au puit P suivant le cas.

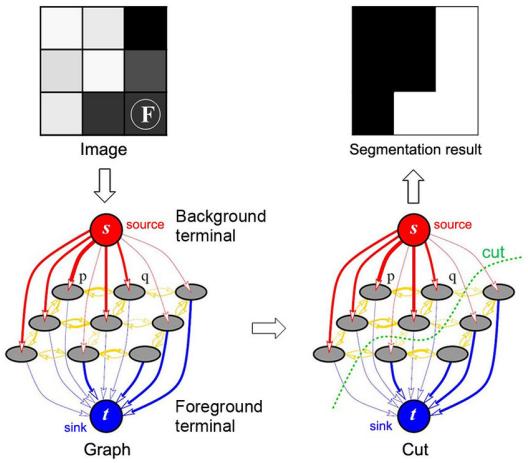

FIGURE 1 – Binarisation d'une image 3*3 par coupe de graphe (adapté de Boykov et Funka-Lea, 2006). La source s'appelle S et le puit T. On a ici ajouté la contrainte que le pixel F doit être classé comme étant sombre en ne le connectant pas à la source. La capacité d'un arc mesure la similarité locale entre deux pixels ainsi que la propension locale d'un pixel à être classé comme clair ou sombre.

La binarisation finale de l'image s'obtient en coloriant en noir (0) l'ensemble des nœuds qui sont connectés au nœud P et en blanc (255) l'ensemble des nœuds qui sont connectés au nœud S.

Modifier l'image avec ces nouvelles intensités et sauvegardez-la.

Travail à faire et pistes de travail : Coder l'algorithme de coupure de graphe d'une image, par recherche de flot maximal. Pour rechercher un chemin entre la source et le puit, on pourra utiliser l'algorithme de parcours en largeur du graphe à partir de la source S ou bien l'algorithme de Dijkstra pour chercher les plus courts chemins issus de S. On peut stocker les infos relatives au parcours du graphe dans des tableaux de taille $L \times C + 2$, dont les $L \times C$ premières cases sont associées aux pixels de l'image et les deux dernières cases sont associées aux sommets S et T. Ainsi on pourra avoir des tableaux pour stocker les couleurs et les prédécesseurs. Vous essayerez plusieurs valeurs de σ (entre 10 et 30 environ) et d'α pour voir la sensibilité de la binarisation à ces valeurs.

1 Conditions de rendu

Vous placerez dans Tomuss une archive de votre travail $Nom1\_Nom2.tgz$ ($Nom1$ et $Nom2$ étant les noms des 2 étudiants composant le binôme) avant le **mardi 5 décembre 2023** à 22h.

Cette archive doit contenir un répertoire $Nom1\_Nom2$ avec les sources de votre application. Les sources sont bien évidemment tous les fichiers *.cpp* et *.h*, mais aussi le fichier $Makefile$, ainsi que tout autre fichier ($README$, etc.) utile à la compréhension de votre programme, ainsi qu'à sa compilation et à son exécution.

Le fichier $Makefile$ doit modéliser clairement les dépendances entre les fichiers et permettre une recompilation partielle, en cas de modification partielle.

Attention !

- Votre archive NE DOIT PAS contenir de fichiers objets (.o) ni d'exécutable.
- Le travail rendu doit être issu de la collaboration entre 2 personnes et toute récupération flagrante du code d'autrui sera sanctionnée.
- Votre programme doit pouvoir être compilé sans erreur ni avertissement avec $g++$ **dans une des salles de TP Linux de l'Université**, et sans nécessiter l'installation de bibliothèques supplémentaires.
- une attention particulière devra être consacrée à la mise en œuvre d'une programmation modulaire débouchant sur la proposition de véritables types abstraits.
- l'interface de vos modules doit offrir toutes les informations utiles à l'utilisation des types et des fonctionnalités offertes.
- l'exécution de votre main doit illustrer le bon fonctionnement de l'ensemble des éléments du projet.

Le **mercredi 6 décembre 2023**, vous serez amenés à répondre à quelques questions sur papier concernant votre travail, puis à faire une petite présentation au cours de laquelle des questions seront posées à chacun des 2 membres du binôme.