IG 4 - Graphes 2

TP : application des flots au débruitage d'images

Avant de commencer

Télécharger les fichiers fournis. Ces fichiers contiennent du code à trou qui compile, mais qui ne fait rien pour l'instant. Ne regardez pas le code pour l'instant, et commencez par la lecture des sections ci-dessous qui présentent le sujet globalement. L'ordre pour compléter les morceaux de code est indiqué en Section 3.

1 Introduction

Le problème du flot maximum (ou de la coupe minimum) a de nombreuses applications dans le traitement d'images. Une des applications est le problème de la segmentation d'image, consistant à supprimer d'une image la partie qui correspond à l'arrière plan. Dans l'article "Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images" (Yuri Y. Boykov and Marie-Pierre Jolly), les auteurs traitent ce problème de segmentation en se réduisant au problème du flot max.
L'objectif de ce TP est de résoudre une version simplifiée du problème de segmentation, en utilisant également une réduction vers le problème du flot max. Le problème que nous considérons est celui de débruitage d'image dans une image uniquement constituée seulement de pixels noirs ou blancs[1], et se formalise de la façon suivante.
L'entrée du problème est

- un objet "im" de la classe Img, avec l'hypothèse que pour tout pixel de coordonnée (i,j), im.get(i,j) vaut soit 0 (pixel noir), soit 255 (pixel blanc).
- une valeur $\alpha > 0$

On fixe l'orientation suivante : i représentera le numéro de colonne, j celui de ligne, la colonne 0 étant à gauche, et la ligne 0 en haut.
La sortie du problème est une image "im2" ayant les mêmes dimensions que im, et elle aussi constituée uniquement de pixels noirs ou blancs. Afin de définir la fonction à minimiser nous avons besoin des notations suivantes. On dénote $\mathcal{P}$ l'ensemble des pixels, et $\mathcal{E}(\mathcal{P})$ l'ensemble des pairs de pixels voisins, avec la convention qu'un pixel a pour voisins les pixels à distance 1 dans l'image. Un pixel au milieu de l'image aura donc 4 voisins (haut, bas, gauche, droit), un au bord aura 3 voisins, et un dans un coin 2 voisins. Le but est de minimiser $f(im2)$, avec $f(im2) = \sum_{(i,j)\in\mathcal{P}} d(im2.get(i,j), im.get(i,j)) + \alpha \sum_{(i,j),(i',j')\in\mathcal{E}(\mathcal{P})} d(im2.get(i,j), im2.get(i',j'))$, où $d(x,y) = 1$ ssi $x \neq y$, et 0 sinon. On paye donc 1 pour chaque pixel ayant changé par rapport à l'image d'origine, et α pour chaque couple de pixels voisins ayant une couleur différente.
L'idée de cette formalisation est la suivante. L'image d'origine "im" sera typiquement une image ayant été bruitée : elle contiendra donc de grandes zones toutes noires ou toutes blanches, et des petites zones "bruitées" où les pixels ont été inversés. Le but est de débruiter l'image "im" en calculant une image "im2" qui soit à la fois proche de "im", mais également plus "unifiée/lissée", au sens où l'on souhaiterait limiter le nombre de changements de couleurs de pixels voisins. Plus α sera grand, et plus les solutions optimales se contenteront d'obtenir de grandes zones monochromes, quitte à ne pas respecter les couleurs d'origine. Voir la Figure 1 ci-dessous pour un exemple d'une image "im" (à gauche), et d'une image "im2" (à droite) de coût $2 + 3\alpha$.

2 Résolution

Pour résoudre ce problème, nous allons procéder ainsi.

[1] Ce problème est exactement celui traité dans http://mpechaud.fr/scripts/maxflow/index.html

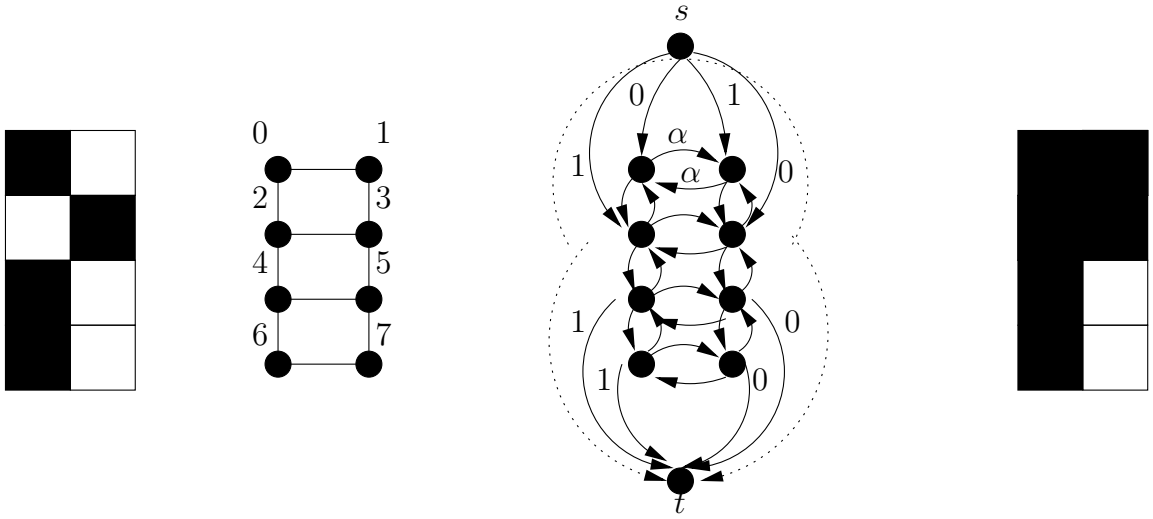

Figure 1: De gauche à droite. Une image en entrée im. Le graphe dans l'instanceDebruitage associé à im. Le réseau associé à l'instanceDebruitage. La sortie im2 correspond à la solution $S = \{5, 7\}$ de l'instanceDebruitage. La solution S correspond à la coupe $\{s, 5, 7\}$ dans le réseau. Le théorème 1 nous dit que $f(S) = 2+3\alpha = c(\{s, 5, 7\}$ ($c(X)$ est le coût de la coupe X).

Création d'une InstanceDebruitage La première étape consiste à associer à une entrée du problème un objet de la classe InstanceDebruitage. Plus précisément, étant donné un objet Img (de hauteur H, largeur L, et ayant $|\mathcal{P}| = HL$ pixels) et un α, nous allons créer un objet de type InstanceDebruitage (grâce au constructeur InstanceDebruitage(Img im) fourni) constitué de :

- un graphe g à $n = |\mathcal{P}|$ sommets. Les pixels de la première ligne seront associés aux sommets 0 (pour pixel en haut à gauche), 1, ..., $L-1$, puis pour la deuxième ligne L, $L+1$, etc. Le pixel de coordonnées (i, j) sera associé au sommet d'indice $calculIndice(i, j)$ (calculIndice est une méthode de Img).
- un tableau b à n cases. Le niveau de gris (0 ou 255 ici) du pixel (i, j) sera stocké dans $b[calculIndice(i, j)]$.
- la valeur α.

On reformule donc le problème précédent de la façon suivante : étant donné une entrée (g, b, α), le but est de trouver un sous ensemble $S \subseteq V(g)$ qui minimise $f(S)$, où la fonction f est définie de manière analogue au paragraphe précédent (S représente l'ensemble des sommets du graphe à colorier en blanc). Le but final sera donc de coder la méthode calculOpt de InstanceDebruitage qui retournera une solution optimale S. On remarque au passage que, même si pour notre application les graphes g considérés seront des grilles, le problème de minimisation de f a du sens pour tout graphe g, et calculOpt sera capable de trouver une solution optimale pour un graphe quelconque. Nous allons maintenant détailler la stratégie adoptée par calculOpt.

Stratégie adoptée par calculOpt La première étape consiste, étant donnée une InstanceDebruitage (g, b, α), à créer un réseau $R(g, b, \alpha)$ comme indiqué sur la figure ci-dessus. La propriété clef est la suivante.

Theorem 1. *Pour tout (g, b, α) et t,*
si S est une solution de (g, b, α) telle que $f(S) = t \Leftrightarrow$ alors $\{s\} \cup S$ est une coupe de $R(g, b, \alpha)$ de valeur t.

On déduit du théorème précédent que pour résoudre optimalement une InstanceDebruitage (g, b, α), on va créer le réseau $R(g, b, \alpha)$, et calculer une coupe minimum dans ce réseau.

3 Travail à faire

3.1 Img et InstanceDebruitage

1. Dans la classe MainGraphe, écrire un main effectuant les actions suivantes :

- création d'un int[][] data et remplissage de data pour avoir l'image de gauche dans la Figure 1
- création d'un objet im de type Img grâce au tableau data

- écriture de im dans un fichier via la méthode creerImage (on veut obtenir l'image de gauche de la Figure 1)
- création d'un objet instDeb de type InstanceDebruitage à partir de im
- affichage dans un terminal de instDeb (et contrôler que ce vous voyez est ok!)

2. Dans la classe Reseau, écrivez le code du constructeur prenant en paramètre une InstanceDebruitage. Vérifier votre méthode en affichant le réseau obtenu pour l'exemple construit dans la question 1.

3.2 Ecriture de l'algo de flot max

3. Dans la classe Flot, écrire la méthode créerReseauResiduel. Pour tester, créez un Reseau à la main sans digons! (donc ne pas utiliser le réseau précédent)

4. Dans la classe Reseau, écrire la méthode trouverChemin. Testez les deux cas (selon qu'il existe un chemin ou non).

5. Dans la classe Flot, écrire la méthode modifieSelonChemin.

6. En utilisant les questions précédentes, écrire la méthode flotMax de la classe Réseau. Testez!

3.3 Résolution du débruitage

7. Dans la classe InstanceDebruitage, écrire la méthode calculOpt. Attention, une fois le réseau r créé à partir de l'instanceDebruitage comme indiqué dans la figure ci-dessus, il ne faut pas directement appliquer la méthode flotMax sur r, puisque flotMax suppose qu'il n'y a pas de digons. Ainsi,

- utilisez d'abord la méthode supprimerDigons sur r : de nouveaux sommets (numérotés à partir de n, où n étaient le nombre de sommets de r avant modification, voir là specification de supprimerDigons) vont être ajoutés à r
- calculer une coupe min S sur r
- conservez seulement dans S les sommets d'indice strictement inférieur à n

8. En utilisant les méthodes calculFiltre et appliquerFiltre de la classe Img, résolvez le problème de départ! Vous pouvez tester le cas suivant. Pour la grille 5x5
bnnnb
nbnbn
bnbnb
nbnbn
bnbnb

avec $\alpha = 0.4$ on doit obtenir
bnnnb
nnnnn
nnnnn
nnnnn
bnnnb

avec $\alpha = 0,3$ on doit obtenir
bnnnb
nnnnn
bnnnb
nnnnn
bnbnb

4 Bonus

Des directions possibles :

- gérer les niveaux de gris, et pas simplement noir ou blanc
- coder la segmentation d'image (voir par ex https://julie-jiang.github.io/image-segmentation/, qui reprend les points principaux de l'article)