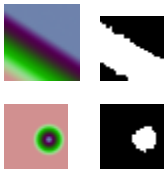
Harry Jandu

Binarisation d’image



Contents

1. Introduction

La binarisation d'image consiste à diviser les pixels d'une image en 2 classes – 0 ou 1 – dont vient le nom « binarisation ». Les applications de binarisation peuvent se trouver dans plusieurs domaines comme la détection des visages, traitement et analyse des images médicaux, etc. Dans ce projet, le problème du flot max sera utilisé afin de trouver deux partitions d'une image.

2. Données

Le programme prend en entrée un fichier contentant le suivant en ordre qu'ils apparaissent :

1. *n m* : les dimensions de l'image
2. Les probabilités $a_{ij} > 0$ formée de *n* lignes et *m* colonnes.
3. Les probabilités $b_{ij} > 0$ formée de *n* lignes et *m* colonnes.
4. Les pénalités horizontales formée de *n* lignes et *(m – 1)* colonnes.
5. Les pénalités verticales formée de *(n – 1)* lignes et *m* colonnes.

Exemple du fichier d'entrée :

```
2 2

1 9

1 12

8 1

9 1

2

3

15 19
```

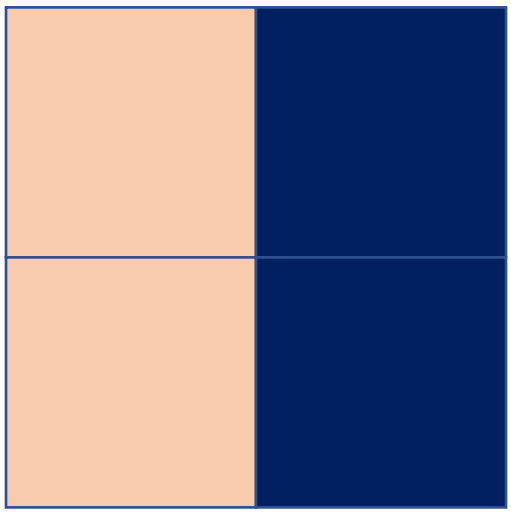


Figure 1: Image pour les données a gauche

3. Modélisation

On a des pixels P_{ij} *i <= n, j <= m*.

Le graphe G = (V, E) non orienté sera définie par V = P l'ensemble des pixels et une arrête *e = {les voisins à distance d'un pixel de toute direction}*.

On ajoute une source *« s »* et un puit *« t »* telle que :

- *s* aie des arcs sortant vers toutes les pixels P_{ij} sauf *t* (s->P_{11}, *s->*P_{nm}) avec les capacités de a_{ij}
- *t* aie des arcs entrant de toutes les pixels P_{ij} sauf *s* (P_{11}*->t*, P_{nm}*->t*) avec les capacités de b_{ij}
- *toutes les autres arrêtes en P deviendra des arrêtes bidirectionnelles avec les pénalités horizontales et verticales*

4. Fonctionnement de l'application

DirectedGraph.java
G = (V, E) avec V : Integer et E : Edge

```
BFS_(u, v : Nœud) : HashMap< Nœud, Edge>
        Visité = {}             // hashmap de nœud et boolean ; dire si un nœud est déjà visité
        Chemin = {}             // hashmap de nœud et edge ; contiendra toutes les nœuds et leurs edges
        Queue = []              // pour ajouter et enlever les nœuds
        While (queue != null)
                Enlever premier element
                Parcourir tous les voisins et ajouter remplir visité
                Remplir chemin avec (voisin, arrete entre nœud courant et voisin)
        Renvoyer le chemin
```

```
// créer un chemin de u et v et calculer le flot minimum qui peut passer par ce chemin
Make_path(chemin : HashMap(renvoyé par bfs_), u, v)
        ret = {}  // un ensemble vide de Edge
        parent = v;       // commencer par le dernier en allant en reverse
        flot_min = infinité
        While (arrete = chemin.get(parent) != null)
                Si flot_courant de arrete < flot_min alors flot_min = flot_courant
                Ajouter(ret, arrete);
        Renvoyer ret
```

```
edmondsKarp(u, v : Nœud)
        h = (V, E) copie de graph original
        r = residuel de graph h
        // on cherche les Chemins dans le graph residuel
        parent_array = r.bfs_(u, v)
        while (path = make_path(u, v) != null)
                //mettre à jour le flot dans le graph h
```

h.updateNetwork
r = nouveau residuel graph de h
parent_array = r.bfs_(u, v)
calculer la somme de flot entrant de s et sortant dans t
verifier que c'est egale (flow in = flow out)
renvoyer flot et graph h avec les flots mis-à-jour

Edge.java
Edge composé de U, V, flow, capacité : Integer

⇨ *G = (V, E, f, c) avec f, c = flow, capacité*

Application.java
Readfile(jeu_de_données);
Lire premier ligne et extraire les dimensions d'image
(dans le boucle, y a 4 cas – probabilités de a, b, pénalités horizontales et verticales respectivement
While (ligne lu n'est pas nul):
Lire la ligne et séparer la ligne par les espaces et vérifier dans quel cas on est et remplir le graphe

5. Instruction de compilation

Avec maven – se retrouver dans la base répertoire du projet avec le pom.xml

mvn package

java -cp target/Image-segmentation-0.0.1-SNAPSHOT.jar image.segmentation.Application

6. Exécution de l'application pour trouver les deux ensembles A et B

1. *L'exemple dans l'énoncé*

```
<terminated> Application (1) [Java Application] C:\Program Files\Java\jdk1.8.0
vertex count: 18
edge count: 80
t_edges: [(16, 18, 1, 20), (9, 18, 6, 20), (2, 18, 1, 20),
time taken to calculate max flow and set A and B: 46ms
max flow: 49
background: [7, 8, 11, 12]
foreground: [2, 3, 4, 5, 6, 9, 10, 13, 14, 15, 16, 17]
```

Figure 2: terminaison avec max flow de 49

2. *Exemple avec des images 32x32*

Figure 3.1: original

Figure 3.2: binarisé

```
<terminated> Application (1) [Java Application] C:\Program Files\Java\jdk1.8.0_202\b
vertex count: 1026
edge count: 6016
t_edges: [(517, 1026, 1, 1), (2, 1026, 1, 1), (519, 1026, 1, 1),
time taken to calculate max flow and set A and B: 144824ms
max flow: 3731
background: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
foreground: [167, 168, 169, 170, 171, 172, 181, 182, 183, 199, 2
```

Figure 4: exécution de binarisation de Figure 3.1

Figure 5.1: original

Figure 5.2: binarisé

```
<terminated> Application (1) [Java Application] C:\Program Files\Java\jdk1.8.0_
vertex count: 1026
edge count: 6016
t_edges: [(517, 1026, 24, 28), (2, 1026, 1, 1), (519, 1026,
time taken to calculate max flow and set A and B: 171831ms
max flow: 4404
background: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
foreground: [101, 102, 109, 110, 111, 112, 121, 122, 123, 12
```

Figure 6: execution pour binariser Figure 5.1

7. Conclusion

La segmentation d'image en utilisant les coups minimums ne marchent que quand le nombre de nœuds et arrêtés n'est pas trop large vu que la complexité de la plupart des algorithmes de flots est de *O(VE)*. On peut essayer d'autres algorithmes pour binariser une image comme l'algorithme de KNN ou Deep Learning.

Bibliographie

https://madoc.univ-nantes.fr/pluginfile.php/931845/mod_resource/content/13/FlotsAvecPreflotORO_2017.pdf

https://www.coursera.org/lecture/advanced-algorithms-and-complexity/image-segmentation-refBK

https://www.coursera.org/lecture/image-processing/7-interactive-image-segmentation-duration-21-13-zEn0I

http://bigwww.epfl.ch/chaudhury/ME_thesis_KunalNC.pdf

https://www.youtube.com/watch?v=lYQQ88nzxAM

https://math.stackexchange.com/questions/677743/finding-the-max-flow-of-an-undirected-graph-with-ford-fulkerson

https://en.wikipedia.org/wiki/Maximum_likelihood_estimation

https://www.slideshare.net/UlaBac/lec10-medical-image-segmentation-as-an-energy-minimization-problem

https://www.inf.u-szeged.hu/~kato/teaching/emm/multi-layer-mrf.pdf

http://www.csd.uwo.ca/faculty/yuri/Papers/iccv01.pdf

http://lvelho.impa.br/ip13/reading/ijcv06.pdf