UNIVERSITÉ DE NANTES

Rapport de projet Graphes II
Binarisation d'images

Aniss Bentebib, Camille-Amaury Juge

Respectivement M1 Informatique ATAL et DS

Sommaire :

I - Introduction

Ce rapport a été réalisé dans le cadre du projet de M1 informatique à l'université de Nantes, pour le cours de *Graphes II et Réseaux* tenu par Mme Irena Rusu-Robini.

Nous allons donc commencer par présenter brièvement le contexte du sujet et les spécifications dont vous aurez besoin pour y accéder.

Dans un premier temps, nous avons mis à disposition toute la structure du projet sous github :

https://github.com/camilleAmaury/ProjetGrapheEtReseaux

Vous pourrez y trouver le fichier " ***BinarisationImage_19_20.pdf*** " qui contient tout l'énoncé du projet.

Ensuite, vous pouvez vous référer au fichier Readme.md qui vous donnera toute la procédure quant à la bonne utilisation du programme.

Maintenant que le projet vous est accessible, nous allons pouvoir aborder plus en détails l'implémentation JAVA sous forme de pseudocode.

II - Pseudocode et Utilisation

Ici, nous détaillerons les méthodes afin d'utiliser notre programme ainsi que son implémentation.

A - Utilisation :

Déposer à la racine du projet le fichier texte contenant vos données. Ensuite, grâce à la commande :

java -jar ProjetGraphe2EtReseaux.jar nomFichierTexte nomStructure

Attention, "nomFichierTexte" ne nécessite pas l'extension. De plus il est obligatoire de passer un fichier .txt.
"nomStructure" est un choix entre "list" ou "matrix" en fonction de votre choix d'implémentation voulue.

B - Pseudocode Global :

Toutes les fonctions de l'énoncé sont stockées dans la classe JAVA "***Util***" qui se trouve au sein de notre package "***Classes***".

Ainsi, le but du projet étant bien sûr de pouvoir résoudre le problème BinIm, la fonction "***ResolveBinIm***" ci-dessous est la fonction qui encapsule tout le procédé de l'algorithme.

```
Fonction  ResolveBinIm (Chaine de Caracteres : filename,
    Booleen : logs){
    // Construit le graphe en lisant le fichier texte
       specifie
    Graphe Pondere(matrice d'adjadence, liste de noeud) :
        G <- Util.ConstructionReseau(filename, logs)
    // Procede au calcul du flot maximum (gere le graphe
       residuel etc...) et affiche la valeur du flot
       maximum
    g <- Util.CalculFlotMax(g, logs)
    // Procede au calcul de la coupe minimum
    Coupe de Graphe(X <- Liste d'entiers, Y <- Liste d'
       entiers) cut <- Util.CalculCoupeMin(g, logs)
    // affiche la valeur de la coupe
    Afficher "Coupe X : " + Pour i allant de 1 a Taille(X
       ) Faire Afficher "Node " + X[i]
    Afficher "Coupe X : " + Pour i allant de 1 a Taille(Y
       ) Faire Afficher "Node " + Y[i]
Fin Fonction
```

Cette fonction prend en paramètre une chaîne de caractère correspond au nom du fichier texte à lire pour créer le graphe, mais aussi un booléen qui ne sert uniquement que si vous souhaitez un déroulé plus complet de notre algorithme.

Cette fonction renvoie le graphe dans son état final ainsi que les deux partitions de sommets X et Y, représentant les ensembles de la coupe minimum privé de S et T (sommets ajoutés pour l'algorithme de flot).

Nous détaillerons ici, uniquement la fonction "***CalculFlotMax***" puisque les prochaines seront chacunes détaillées dans leur partie respective.

Sans rentrer dans le code, cette fonction a été implémenté au moyen de l'algorithme de Ford-Fulkerson, qui est moins coûteux en temps mais plus ardu à implémenter que la méthode des préflots, cependant celle-ci ne converge pas forcément.

La fonction prend donc en paramètre un graphe pondéré représentant un réseau de transport (qui dans notre cas est obtenu par la lecture du fichier texte et le réarrangement des données) et un booléen qui ne sert uniquement que si vous souhaitez un déroulé plus complet de notre algorithme.

La méthode affiche le flot maximum trouvé du graphe, puis retourne le graphe. A noter qu'à chaque itération, l'algorithme s'arrête dès qu'il a trouvé un chemin possible dans le graphe résiduel (moins coûteux).

C - Pseudocode "ConstructionReseau" :

```
Fonction parserFileToGraphFile(Chaine de Caracteres :
    filename, booleen : logs):
    // lis le fichier texte et retourne un Graphe Pondere
        correspondant au fichier
Fin Fonction
```

```
Fonction toNetworkGraph(Graphe Pondere g):
    Initialiser Graphe Pondere g_final
    // ajout du sommet S et T, respectivement a l'indice
       0 et a la fin
    Nombre de Noeuds(g_final) = Nombre de Noeuds(g) + 2

    // Pour chaque arcs du fichier texte, on les
       reaffecte aux arcs du nouveau graphe prive de S et
        T
    Pour i allant de 0 a Nombre de Noeuds(g_final) - 1,
       Faire:
        Pour j allant de 0 a Nombre de Noeuds(g_final) -
           1, Faire:
            Si i == 0 ou i == this.graph.length - 1 ou j
               == 0 ou j == this.graph[i].length - 1
                Alors:
                 g_final.Matrice[i][j] = Arc (Flot <- 0,
                    Capacite <- 0)
            Sinon:
                 g_final.Matrice[i][j] = Arc(Flot <- 0,
                    Capacite <- g_final.Matrice[i][j].
                    Penalite)
            Fin Si
        Fin Pour
    Fin Pour
    // On cree un dictionnaire pointant sur les noeuds du
        graphe par id
    Pour i allant de 0 a Nombre de Noeuds(g_final) - 1,
       Faire:
       g_final.Noeuds.ajouter(i, Noeud(id <- i));
    Fin Pour
    // On affecte les probabilites A_ij en capacite a
       tous les arcs possibles (sauf T) ayant pour sommet
        initial S
    Pour j allant de 0 a Nombre de Noeuds(g_final) - 1,
       Faire:
        g_final.Matrice[0][j] = Arc (flot <- 0, capacite
           <- g.Noeuds[j-1].probabiliteA)
    Fin Pour
    // On affecte les probabilites B_ij en capacite a
       tous les arcs possibles (sauf S) ayant pour sommet
        final T
    Pour i allant de 0 a Nombre de Noeuds(g_final) - 1,
       Faire:
        g_final.Matrice[i][Nombre de Noeuds(g_final) - 1]
            = Arc (flot <- 0, capacite <- g.Noeuds[i-1].
           probabiliteB)
    Fin Pour
    // on conserve la taille du jeu de donnees initial
    g_final.TailleInitiale = g.TailleInitiale
Fin Fonction
```

```
Fonction ConstructionReseau(Chaine de Caracteres :
    filename, booleen : logs):
    // On lit le fichier texte, on retourne un objet
        GraphFile contenant les Pij sur les arcs et
    les probabilite Aij et Bij sur chaque Noeud
    Graphe Pondere g = parserFileToGraphFile(filename,
        logs)
    // on converti le graphe du fichier vers un reseau de
         transport
    Graphe Pondere gr = toNetworkGraph(g)
    // on affiche si necessaire
    Si logs Alors
        gr.print()
    Fin Si
    Retourner gr
Fin Fonction
```

La méthode "***ConstructionReseau***" prend en paramètre une chaîne de caractère représentant le fichier texte à parser et un booléen pour afficher plus ou moins de détails sur l'exécution.

Dans un premier temps, celle-ci lit le fichier au moyen de la fonction "***parserFileToGraphFile***" (nous ne la détaillerons pas ici car très volumineuse et peu intéressante).
Cette fonction retourne un Graphe pondéré correspondant à la lecture du fichier. Ainsi les arcs représentent les P_ij (pénalité de séparation) dans la matrice d'adjacence, et chaque noeud (pixel) contient la probabilité A_ij et B_ij.

Ensuite, on convertit ce graphe vers un graphe représentant un réseau de transport. Pour cela on appelle la méthode "***toNetworkGraph***".
Cette fonction prend en paramètre un graphe pondéré lu depuis un fichier texte.

La fonction ajoute deux nouveaux sommets (S et T, d'indice 0 et nombre de noeuds initiaux + 1). Il conserve les anciens arcs représentant les P_ij, cependant il value désormais les arcs entre S et tous les sommets (sauf T) avec la probabilité du sommet concerné (A_ij).
De plus, il value de nouveau les arcs de tous les sommets (sauf S) à T avec la probabilité du sommet concerné (B_ij).

Ces probabilités deviennent les capacités des arcs du réseau.
On précise que les pixels sont représentés par une bijection des pixels dans les entiers :
par exemple le pixel(1,1) sera le noeud 0 du Graphe, le pixel(1,m) sera le noeud m-1, le pixel(n,m) sera le pixel n*m + m-1.

Ainsi lors de la conversion en Graphe pour résoudre notre problème, le changement est léger : le noeud 0 devient le sommet S, puis on décale de +1 tous les anciens noeuds. Le dernier noeud (n*m + m-1 + 2) est T.

Une fois que le réseau de transport est initialisé, la fonction "***ConstructionReseau***" retourne un Graphe pondéré avec le flot initialise à 0 représentant le fichier initial retravaillé pour répondre à un problème de flot.

D - Pseudocode "CalculCoupeMin" :

```
Fonction CalculCoupeMin(Graphe Pondere : graph, booleen :
    logs):
    // Creation du graphe residuel par le graphe de
       transport actuel
    Graphe Pondere gr = g
    // Seule la methode d'evaluation des arcs change (
       voir dans getXRec
    // Recuperation de la tranche X (representee par une
       liste des id des noeuds)
    Liste d'Entiers X = getX(gr)
    // Recuperation de la tranche Y (representee par une
       liste des id des noeuds)
    Liste d'Entiers Y = getY(gr, X)
    // affichage de la valeur de coupe minimale
    Afficher("Minimal Cut : " + g.cutScore(X, Y))
    Retourner Couple(X, Y)
Fin Fonction
```

```
Fonction getX(Graphe Pondere : g):
    // tableau de booleen pour savoir si on a deja visite
       un sommet
    Tableau de booleens : visited,
    Pour i allant de 0 a Nombre de Noeuds(g) - 1 Faire:
        visited[i] = Faux
    Fin Pour

    // retourne uniquement la liste d'entiers
       representant la coupe X
    Retourner getXrec(g, visited, g.getNodeValue(0))[0]
Fin Fonction
```

```
Fonction getXrec(Graphe Pondere : g, Tableau de booleens
    : visited, Noeud : current):
    Si visited[current.id] Alors:
        Retourner Couple(Liste d'entiers vide, visited)
    Sinon:
        // on a visite la node actuelle
        visited[current.getId()] = Vrai
        // Pour chaque successeur de la node dans le
            graphe (c'est-a-dire que la capacite
            residuelle de l'arc est differente de 0), on
            applique recursivement la fonction (interne a
            la fonction getSuccessors)
        Liste d'entiers : successor = g.getSuccesors(
            current.id)
        Liste d'entiers : nodes, initialise a vide
        nodes.ajouter(current.id)
        Pour i allant de 0 a Taille(successor) - 1 Faire:
            Couple(Liste Entiers, Tableau de boolen) :
                res = getXrec(g, visited,
            g.getNodeValue(successor[i]))
            // on concatene les listes de noeuds
                retournes a ceux deja present dans nodes
            nodes.concatenerListe(res[0])
            // on conserve l'etat des noeuds visites pour
                 le retransmettre et eviter des boucles
            visited = res[1]
        Fin Pour
        Retourner Couple(nodes, visited)
    Fin Si
Fin Fonction
```

```
Fonction getY(Graphe Pondere : g, Liste d'entiers : X):
    Liste d'entiers Y vide
    Pour i allant de 0 a Nombre de Noeuds(g) - 1 Faire:
        Si X ne contient pas i, Alors:
            Y.ajouter(i)
        Fin Si
    Fin Pour
    Retourner Y
Fin Fonction
```

```
Fonction cutScore(Liste d'entiers : X, Liste d'entiers :
   Y):
    Reel somme = 0;
    Pour i allant de 0 a Taille(X)-1 Faire:
        Liste d'entiers succ = getSuccesors(X[i]);
        Pour j allant de 0 a Taille(succ)-1 Faire:
            Si X ne contient pas succ[j], Alors:
                somme += g_final.Matrice[X[i]][succ[j]].
                    flot
            Fin Si
        Fin Pour
    Fin Pour
    Retourner somme
Fin Fonction
```

```
// applique uniquement au graphe residuel
Fonction getSuccesors(Graphe Pondere : g, Entier : noeud)
    :
    Liste d'entiers Vide : result
    Pour i allant de 0 a Taille(g.Matrice[noeud]) - 1
       Faire:
        // si la capacite residuelle est differente de 0
            alors l'arc doit etre pris en compte
        Si g.Matrice[noeud][j].capacite - g.Matrice[noeud
            ][j].flot != 0, Alors:
            result.ajouter(j);
        Fin Si
    Fin Pour
    Retourner result
Fin Fonction
```

La fonction "***CalculCoupeMin***" prend en paramètre un Graphe pondéré dont le flot est maximum.
Celle-ci retourne un couple de liste d'entiers, représentant les noeuds dans chaque coupe.

Pour cela on évalue d'abord la coupe X au moyen de "***getX***" et "***getXrec***" dans le réseau résiduel en partant du sommet S et récursivement en accédant aux sommets que nous n'avons pas déjà visité. Ceci forme la partition X. Notons que les arcs existent uniquement si la capacité résiduelle est différente de 0.

Il ne nous reste plus qu'à composer la partition Y au moyen de "***getY***" en prenant le reste des noeuds du graphe non présents dans X.

Nous affichons ensuite la valeur de la coupe minimale "***cutScore***" (tous les arcs allant de X à Y) qui doit être la même que le flot maximal.

III - Réponses aux questions

A - Q1 :

Pour montrer que maximiser q revient à minimiser q'. Il convient d'identifier dans un premier temps ce que représente q et q'. Assez logiquement en lisant l'énoncé on se rend compte que q représente l'équation du flot et q' l'équation de la coupe.

L'objectif ici est de montrer que Q - q(A, B) = q'(A, B)
Pour simplifier les calculs nous prenons l'équation dans l'autre sens en essayant de montrer que q(A, B) + q'(A, B) = Q

$$q(A,B)+q'(A,B) = \sum_{(i,j)\in A} a_{ij} + \sum_{(k,l)\in B} b_{kl} + \sum_{(i,j)\in A} b_{ij} + \sum_{(k,l)\in B} a_{kl} - \sum_{(i,j)\in A,(k,l)\in Bvoisins} p_{ijkl} + \sum_{(i,j)\in A,(k,l)\in Bvoisins} p_{ijkl}$$

$$= \sum_{(i,j)\in A} a_{ij} + \sum_{(k,l)\in B} b_{kl} + \sum_{(i,j)\in A} b_{ij} + \sum_{(k,l)\in B} a_{kl}$$

$$= \sum_{(i,j)\in A}(a_{ij} + b_{ij}) + \sum_{(k,l)\in B}(a_{kl} + b_{kl})$$

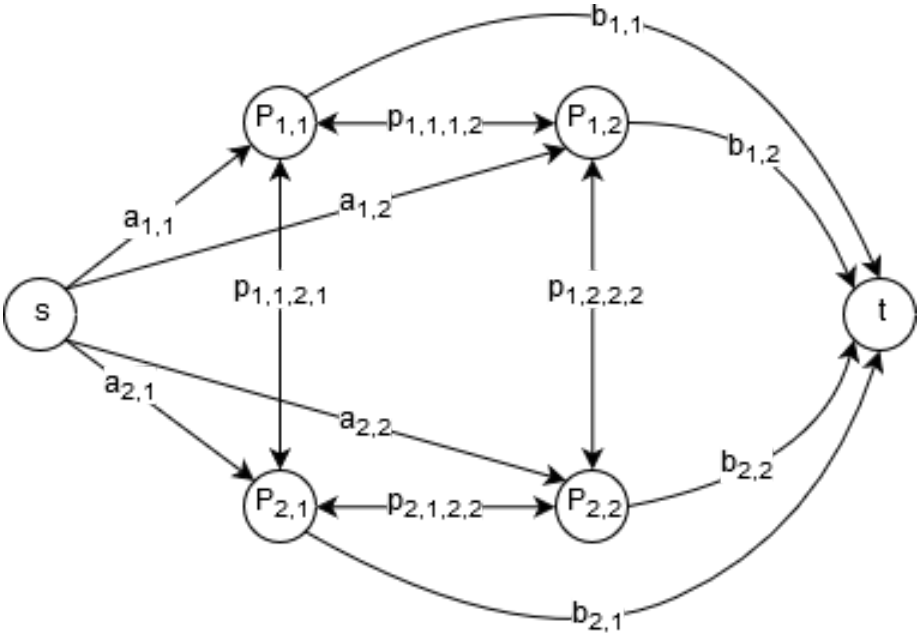
$$= \sum_{(n,m)\in S}(a_{nm} + b_{nm})$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{m}(a_{ij} + b_{ij})$$

$$= Q$$

Donc maximiser q revient à minimiser q' en y faisant intervenir la constante Q.

B - Q2 :



Les valeurs représentées sont des capacités sur les arcs.

C- Q3 :

Si la paire (AU{s}, BU{t}) est une coupe de capacité minimum alors (A, B) est une solution de min(q'(A, B)). Par conséquent, le couple (AU{s}, BU{t}) est une solution du problème de flot maximum car nous savons qu'en minimisant la coupe (q') nous maximisons le flot (q). Comme maximiser le flot revient à résoudre BINIM avec $A \subseteq S$ et $B \subseteq S$ et $A \cap B = \emptyset$ alors la paire (AU{s}, BU{t}) est une solution de BINIM.

IV - Conclusion

L'objectif de ce projet était résoudre le problème de binarisation d'image. Nous avons pu mettre en évidence qu'avec une approche de type réseau de transport, il était possible de trouver une solution à BINIM.