Rapport projet de $3^{ème}$ année : Implémentation d'un algorithme multi-niveaux pour le problème des k-moyennes

15 décembre 2012

Table des matières

Table des figures

1 Définition du problème

1.1 Le problème de partitionnement

Soit un ensemble de n observations $V = (V_i)_{i=1..n}$, une partition Π de V en m classes est la donnée de m sous ensembles $(C_{k=1..m}) \subseteq V$ tel que

$$\begin{aligned} C_k \cap C_{k'} &= \emptyset \quad \forall k \neq k' \\ V &= \bigcup_{k=1}^{m} C_k \\ C_k &\neq \emptyset \qquad \forall k \end{aligned}$$

Soit f un élément de $\mathbb{R}^{|V|} \longmapsto \mathbb{R}$ que nous appelrons fonction de coût, le problème de partitionnement consiste à déterminer une partition de V minimisant f, c'est à dire :

$$\min_{\Pi=(C_1,\ldots,C_m)} \sum_{k=1}^{m} f\left(C_k\right)$$

Dans la suite, un sous-ensemble, une classe, un cluster sont des synonymes et nous noterons $\mathcal{P}_V^m$ l'ensemble des partitions de V en m éléments.

1.2 Problème des k-means

Le problème des k-means est un problème de partitionnement où :

$$\begin{aligned} V_i &\in \mathbb{R}^q \quad \forall i = 1..n \\ f^{kmeans}\left(C\right) &= \min_{Y \in \mathbb{R}^q} \sum_{V_i \in C} \left\|V_i - Y\right\|^2 \end{aligned}$$

Le problème des k-means se formule donc ainsi :

$$\min_{\Pi=(C_1,\ldots,C_m)} \sum_{k=1}^{m} \min_{Y \in \mathbb{R}^q} \sum_{V_i \in C} \left\|V_i - Y\right\|^2$$

Déjà bien étudié dans la littérature, nous pouvons affirmer les choses suivantes :

Théorème 1. *$\forall C \subseteq V \backslash \emptyset$ et $Y^* = \frac{1}{|C|} \sum_{V_i \in C} V_i$, $Y^* = arg \min_{Y \in \mathbb{R}^q} \sum_{V_i \in C} \left\|V_i - Y\right\|^2$*

Autrement dit,

$$f^{kmeans}\left(C\right) = \begin{cases} \frac{1}{|C|} \sum_{V_i \in C} V_i & C \neq \emptyset \\ 0 & C = \emptyset \end{cases}$$

Le problème des k-means s'exprime alors comme :

$$\min_{\Pi=(C_1,\dots,C_m)} \sum_{k=1}^{m} \min_{Y\in\mathbb{R}^q} \sum_{V_i\in C} \|V_i - Y_k\|^2$$
$$Y_k \cdot |C_k| = \sum_{V_i\in C_k} V_i$$
$$\Pi \subset \mathcal{P}_V^m$$

Nous pouvons aussi remarqué que pour chaque élément V_i, Y_k :

$$\|V_i - Y_k\|^2 = \|V_i\|^2 - \frac{2}{|C_k|}\sum_{j\in C_k}\langle V_i, V_j\rangle + \frac{1}{|C_k|^2}\sum_{j,j'\in C_k, j\leq j'}\langle V_j, V_{j'}\rangle$$

Ainsi, il est important de noter que seuls les produits scalaires $\langle V_i, V_j\rangle$ entre les élements de V interviennent dans $KMEANS(V,K,m)$. Nous noterons K la matrice telle que :

$$K_{ij} = \langle V_i, V_j\rangle$$

Enfin, nous introduisons des variables $x_{ik} \in \{0,1\}$ associées à la présence de V_i dans la classe C_k.

$$KMEANS(K,m)^* = \sum_{i=1}^{n} K_{ii} + \min_{\Pi=(C_1,\dots,C_m)} \sum_{k=1}^{m} -\frac{2}{|C_k|}\sum_{j\in C_k} K_{ij} + \frac{1}{|C_k|^2}\sum_{j,j'\in C_k} K_{jj'}$$
$$\Pi \in \mathcal{P}_V^m$$

Théorème 2. $\forall m \leq m^{'}$, $KMEANS(V,K,m)^* \leq KMEANS(V,K,m^{'})^*$

1.3 Partitionnement de graphes

Soit $G = (V, E, W)$ un graphe où V est l'ensemble des noeuds, E est l'ensemble des d'arêtes et W est une matrice telle que W_{ij} est le poids de l'arêtes (i, j)et vaut 0 si $(i, j) \notin E$. Le partionnement du graphe G est un problème de partitionnement dont le critère d'évaluation est une fonction utilisant les arrêtes.

Soit deux classes C_k, $C_{k'}$, nous définissons

$$links\,(C_k, C_{k'}) = \sum_{i \in C_k, j \in C_{k'}} W_{ij}$$

Pour illustrer cette définition :

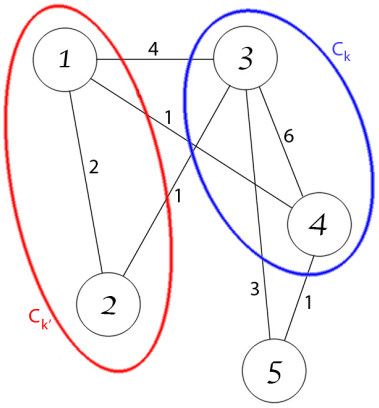

FIGURE 1 – Graphe pour le calcul de $links\,(C_k, C_{k'})$

Les arêtes entrant en considération pour le calcul de $links\,(C_k, C_{k'})$ sont les arêtes reliant les sommets 1 et 3, 1 et 4, 2 et 3. De là on déduit que $links\,(C_k, C_{k'}) = 4 + 1 + 1 = 6$.

Nous definissions maintenant la fonction

$$f^{RAssoc}\,(C) = \frac{links\,(C, C)}{|C|} = \frac{\sum_{i \in C, j \in C\prime} W_{ij}}{|C|}$$

qui est un des critères des problèmes de partitionnement de graphes. Nous noterons $RASSOC\,(W, m)$

$$RASSOC\,(W, m)^* = \min_{\Pi = (C_1, \ldots, C_m)} \sum_{k=1}^{m} f^{RAssoc}\,(C)$$
$$\Pi \in \mathcal{P}_V^m$$

1.4 Lien entre le problème des k-moyennes et le partitionnement de graphes

Théorème 3. *Résoudre $KMEANS(K, m)$ est équivalent à résoudre $RASSOC\,(K, m)$.*

Démonstration. Toute solution optimale de l'un est une solution optimale de l'autre. ***Attention on ne dit pas que les objectifs coïncident***.

La littérature est remplie d'algorithme permettant de travailler dans les graphes et d'optimiser les kmeans. □

2 Les algorithmes

2.1 L'algorithme des kmeans

Réaffecter chaque élément à la classe dont le centroid est le plus proche. Nécessite que la matrice des noyaux soit définie positive.

2.2 L'algorithme spectral

Calculer les m premiers vecteurs propres de la matrice des noyaux, ordonnés en fonction de leur valeur propre.

2.3 L'algorithme de réduction de graphe

Expliquer ce que l'on fait : on choisit un critère local pour agrégé le graphe

2.4 L'algorithme de inverse de la réduction

Pas grand chose à dire ici.

3 L'algorithme GRACLUS

3.1 Déclarations/Classes/Typedef :

3.1.1 GraphType :

Fichier de définition : struct.h
Fonction : structure rassemblant les information sur le graphe donné initialement (ex : vecteurs de sommets , arêtes , poids ...) .
Appelle les structures : idxtype = int, RInfoType, VRInfoType, NRInfoType.

3.1.2 RInfoType :

Fichier de définition : struct.h
Fonction : Cette structure enregistre les informations sur les degrées pour la partition en K-classes.
Appelle les structures : /

3.1.3 VRInfoType :

Fichier de définition : struct.h
Fonction : Cette structure enregistre les informations sur les degrées pour la partition en K-classes basé sur les volumes.
Appelle les structures : /

3.1.4 NRInfoType :

Fichier de définition : struct.h
Fonction : Cette structure enregistre les informations sur les degrées pour la partition en K-classes.
Appelle les structures : /

3.1.5 CtrlType ;

Fichier de définition : struct.h
Fonction : La structure suivante sert de passerelle d'information entre Graclus et Metis
Appelle les structures : WorkSpaceType, timer = double

3.1.6 WorkSpaceType :

Fichier de définition : struct.h
Fonction : Cette structure garde differentes données de l'espace de travail.
Appelle les structures : EDegreeType, VDegreeType

3.1.7 EDegreeType :

Fichier de définition : struct.h
Fonction : Cette structure stocke un sommet
Appelle les structures : /

3.1.8 VDegreeType :

Fichier de définition : struct.h
Fonction : Cette structure stocke un sommet pour la methode volume.
Appelle les structures : /

3.2 Fonctions Graclus

3.2.1 ReadGraph

ReadGraph(GraphType * graph, char * filename , int * wgtflag) ;
Fichier de définition : io.c
Fonction : lit le fichier de données *filename* (*wgtflag* permet d'identifier le format des données, à savoir : est ce qu'il ya des poids sur les sommets, les arêtes, les deux ou aucun) initialise une instance de la structure de *graph* à partir de ces données.
Appelle les fonctions : InitGraph(graph)

3.2.2 MLKKM_PartGraphKway

MLKKM_PartGraphKway(int * nvtxs, idxtype *xadj, idxtype * adjncy, idxtype * vwgt, idxtype *adjwgt, int *wgtflag, int *numflag, int *nparts, int *chainlength, int *options, int *edgecut, idxtype *part, int levels) ;
Fichier de définition : mlkkm.c
Fonction : initialise un tableau de float *tpwgts* appelle MLKKM_WPartGraphKway
Appelle les fonctions : MLKKM_WPartGraphKway(nvtxs, xadj, adjncy, vwgt, adjwgt, wgtflag, numflag, nparts, chainlength, tpwgts, options, edgecut, part, levels)

3.2.3 ComputePartitionBalance

ComputePartitionBalance(GraphType *graph, int nparts, idxtype *where, float *ubvec) ;
Fichier de définition : stat.c
Fonction : calcule l'équilibre d'une partition en fonction des poids des sommets calcule une quantité (le poids moyen ?) d'une partition et place le résultat dans *ubvec*
Appelle les fonctions : N/A

3.2.4 ComputeNCut

ComputeNCut(GraphType *graph, idxtype *where, int npart) ;
Fichier de définition : debug.c
Fonction : calcule l'objectif de la coupe normalisée sur le graphe *graph* étant donnée une partition *where*
Appelle les fonctions : N/A

3.2.5 ComputeRAsso

ComputeRAsso(GraphType *graph, idxtype *where, int npart) ;
Fichier de définition : debug.c
Fonction : calcule l'objectif de l'association par moyenne sur le graphe *graph* étant donnée une partition *where*
Appelle les fonctions : N/A

4 LE RESTE EST A VOTRE DISPOSITION

4.1 Définitions

4.1.1 Voisins

Soit X un point d'un graphe $G = (\mathcal{V}, \mathcal{E}, A)$, on considère un point $Y \in G$ voisin de X s'il existe un arc $(X, Y) \in \mathcal{E}$. On définit ainsi l'ensemble $N_G(X)$ qui regroupe tous les voisins de X dans G.

4.1.2 "Supernoeud"

Lors de la réduction d'un graphe $G = (\mathcal{V}, \mathcal{E}, A)$ en un graphe $H = (\mathcal{V}', \mathcal{E}', A')$, on appele "supernoeud" un sommet de H défini comme un ensemble de sommets de G, soit X un tel supernoeud, on definit $N_H(X)$ comme l'ensemble des supernoeud de H qui contiennent un voisin d'un élément de X :

$$N_H(X) \quad = \quad \{Y \in \mathcal{V}' / \exists (x, y) \in X \times Y,\ x \in N_G(y)\}$$

4.2 Réduction du graphe

La première étape de l'algorithme consiste à réduire la taille du graphe de départ en fusionnant des sommets en "supernoeuds" via différentes méthodes de combinaison.

L'intérêt de cette agrégation réside dans l'obtention d'un graphe dont la taille réduite permettra d'appliquer un algorithme spectral avec une vitesse d'exécution satisfaisante.

On utilise ici l'algorithme suivant pour construire G_i à partir de G_{i+1} :

4.3 Phase de partitionnement initial

La réduction du graphe s'arrête lorsque celui-ci compte moins que $20K$ sommets, où K est le nombre de classes désiré. On peut alors obtenir un partitionnement initial de G_0 en partitionnant G_m, le plus petit graphe obtenu après la réduction.

Pour ce faire l'algorithme spectral de Yu et Shi généralisé avec des pondérations arbitraires est bien efficace sur des problèmes de tailles raisonnables car les résultats sont relativement précis et rapides. Cepandant, il faudra repenser cette partie dans le cas de problème plus important car l'algorithme spectral de Yu et Shi est trop couteux et donc, il sera moins efficace que d'autres algorithmes.

Algorithme 1: Phase de réduction

pour chaque $x \in G_i$ **faire**
  $marque\,(x) \leftarrow faux$
fin
pour chaque $x \in G_i$ **faire**
  si $\forall y \in voisins\,(x)\,, marque\,(y) = vrai$ **alors**
    $marque(x) \leftarrow vrai$;
    $G_i \leftarrow G_i \setminus x$;
  sinon
    trouver y qui maximise $\frac{e(x,y)}{w(x)} + \frac{e(x,y)}{w(y)}$, avec $w(i)$ le poids associé au sommet$i$ et $e(x, y)$ le poids de l'arc entre x et y.;
    Placer x et y au sein d'un même supernoeud;
    $marque(x) \leftarrow vrai$;
    $marque(y) \leftarrow vrai$;
    $G_i \leftarrow G_i \setminus \{x\}$;
    $G_i \leftarrow G_i \setminus \{y\}$;
  fin
fin

4.4 Phase de raffinement

Ce premier partitionnement ayant été obtenu, on peut le raffiner en parcourant les graphes G_{m-1} ,..., G_0 de la manière suivante : si un "supernoeud" de G_i est dans une partition c, alors tous les sommets de G_{i-1} qui le composent sont également dans c, ce qui mène à un partitionnement initial pour le graphe G_{i-1}.

Il faut ensuite raffiner ce partitionnement pour G_{i-1} via l'agorithme suivant, lequel se poursuit tant qu'il permet d'améliorer la fonction objectif sur le graphe G_{i-1} ou jusqu'à une quantité d'itérations maximale. Notons, pour ce graphe, $\pi_c^{(0)}$ le partionnement initial et K une matrice noyau selon la fonction objectif du partitionnement. L'algorithme est alors le suivant :

Algorithme 2: Phase de raffinement

pour chaque *chaque ligne i de K* **faire**
  $d(i, \boldsymbol{m}_c) \leftarrow K_{ii} - \frac{2\sum_{j\in\pi_c(t)} \cdot poids(j)\cdot K_{ij}}{\sum_{j\in\pi_c(t)} \cdot poids(j)} + \frac{\sum_{j,l\in\pi_c(t)} \cdot poids(j)\cdot poids(l)\cdot K_{jl}}{(\sum_{j\in\pi_c(t)} \cdot poids(j))^2}$
fin
Trouver $c^*(i) = argmin_c d(i, \boldsymbol{m}_c)$ (on s'occupe des égalités de façon arbitraire)
tant que ((¬ *convergence*) *et* (*seuil d'itérations pas dépassé*)) **faire**
  Mettre à jour les partitions :
  $\pi_c^{(t+1)} \leftarrow \{i : c^*(i) = c\}$;
  $t \leftarrow t + 1$;
fin

Notons qu'au cours du raffinement, les centres ne sont pas modifiés. La mise à jour se produit en fin d'algorithme en fonction des réaffectations effectuées par ce dernier.

Ce raffinement n'est pas en soi nécessaire pour obtenir une partition de G_0 : le partionnement initial obtenu via l'algorithme spectral sur G_{m-1} peut déjà servir de partionnement pour G_0. La raffinnement sert donc à améliorer les résultats de ce partitionnement.

On pourrait également raffiner directement le partionnement initial sur G_0, mais cette solution augmenterait le nombre d'itération de l'algorithme des k-moyennes et réduirait donc l'efficacité de l'algorithme.

4.5 Algorithme de principe

Notre algorithme général peut se décomposer en trois phases :

- Le première phase "d'agrégagtion" qui va utiliser l'algorithme (1) afin de reduire et d'obtenir un graphe avec K "supernoeuds".
- Une phase de "partitionnement" qui va nous façonner notre premiere et grossière partition décrite au point 2.2.
- Puis vient la phase de "raffinement" qui, par le biais de l'algorithme (2), permet d'améliorer la grossière partition afin de tirer le meilleur regroupement possible.

Algorithme 3: Algorithme général de partitionnement$(K, G_0 = (\mathcal{V}_0, \mathcal{E}_0, A_0), \{\pi_c\}_{c=1}^k, t_{max})$

Input : K : nombre de partitions que l'on désire , G_0 : Graphe initial , t_{max} : nombre maximum d'itération

Output : $\{\pi_c\}_{c=1}^k$: partitionnement final de notre graphe

$k = 0$

tant que $|\mathcal{V}_k| < 20K$ **faire**

  $G_{k+1} \leftarrow algorithm1(G_k)$;

  $k \leftarrow k + 1$;

fin

$\{\pi_c\}_{c=1}^k \leftarrow$Partitionnement primaire par l'algorithme de Yu et Shi.

pour *i de k jusqu'à 1 avec pas* $= -1$ **faire**

  Appliquer $\{\pi_c\}_{c=1}^k$ *à* G_{k-1};

  $\{\pi_c\}_{c=1}^k \leftarrow algorithm2(K, k, \omega, t_{max}, \{\pi_c\}_{c=1}^k)$;

fin

5 To-do list (40%)

1. ~~Comprendre l'équivalence graphes / k-moyennes (rapport).~~ (100%)
2. Rédaction de la première partie (rapport). (66%)
3. Lire / "dérouler" le code et isoler les appels metis.
4. Prevoir un créneau pour ce voir.
5. Ecrire l'algo 3 avec les fonctions de metis.
6. ~~Completer le rapport avec les nouvelles info de l'entretien.~~ (100%)
7. Apprendre a ce servir de GIT.
8. ~~reprendre les algos en bon français.~~ (100%)
9. Homogéinisé les notations dans les algos (prendre soit les notation de 1.1 soit celle de 1.2)