

Rapport projet de 3<sup>ème</sup> année :  
Implémentation d'un algorithme multi-niveaux  
pour le problème des k-moyennes

17 janvier 2013

## Table des matières

<b>1</b>	<b>Définition du problème</b>	<b>3</b>
1.1	Le problème de partitionnement . . . . .	3
1.2	Problème des k-means . . . . .	3
1.3	Partitionnement de graphes . . . . .	5
1.4	Lien entre le problème des k-moyennes et le partitionnement de graphes . . . . .	6
<b>2</b>	<b>Les algorithmes</b>	<b>7</b>
2.1	Définitions . . . . .	7
2.1.1	Voisins . . . . .	7
2.1.2	“Supernoeud” . . . . .	7
2.2	L’algorithme de réduction de graphe . . . . .	7
2.3	L’algorithme spectral . . . . .	8
2.4	L’algorithme inverse de la réduction (par la méthode des K-moyennes) . . . . .	8
2.5	Algorithme de principe . . . . .	8
<b>3</b>	<b>Implémentation</b>	<b>9</b>
3.1	Classes essentielles . . . . .	9
3.1.1	KMConstraints . . . . .	9
3.1.2	KMInstance . . . . .	9
3.1.3	KMPartition . . . . .	9
3.2	Agrégation . . . . .	9
3.3	Raffinement . . . . .	10
<b>4</b>	<b>Présentation de quelques instances pour les tests</b>	<b>12</b>
<b>5</b>	<b>Tests et Analyses</b>	<b>13</b>
5.1	Essais sur le fichier wine.dat (instance 1) . . . . .	13
5.2	Essais sur le fichier segmentation.dat (instance 8) . . . . .	14
5.3	Essais sur le fichier letter.dat (instance 15) . . . . .	14
5.4	Analyse . . . . .	17
5.4.1	Nombre d’itérations . . . . .	17
5.4.2	Résultats . . . . .	17

## Table des figures

1	Graphe pour le calcul de $links(C_k, C_{k'})$ . . . . .	5
2	Extrait d’un fichier data . . . . .	12

# 1 Définition du problème

## 1.1 Le problème de partitionnement

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

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

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

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

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 :

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

Le problème des k-means se formule donc ainsi :

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

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

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

Autrement dit,

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

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

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

Nous pouvons aussi remarquer 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 éléments 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  $C_k$ .

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

**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 partitionnement du graphe  $G$  est un problème de partitionnement dont le critère d'évaluation est une fonction utilisant les arêtes.

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 :

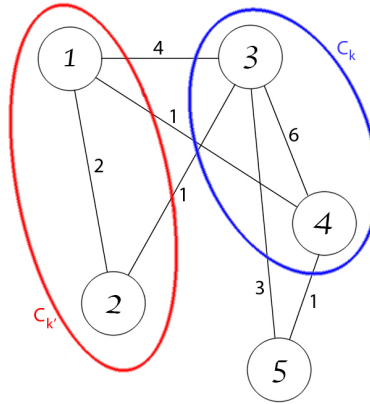


FIGURE 1 – Graphe pour le calcul de  $links(C_k, C_{k'})$

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 définissons maintenant la fonction

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

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

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

## 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'algorithmes permettant de travailler dans les graphes et d'optimiser les kmeans.

□

## 2 Les algorithmes

### 2.1 Définitions

#### 2.1.1 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  $G$ .

#### 2.1.2 “Supernoeud”

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

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

### 2.2 L'algorithme de réduction de 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  $G_{i+1}$  :

---

**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
```

---

## 2.3 L'algorithme spectral

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. Ceci est obtenu grâce à un algorithme spectral qui se base sur le calcul des  $m$  premiers vecteurs propres de la matrice des noyaux, ordonnés en fonction de leur valeur propre.

## 2.4 L'algorithme inverse de la réduction (par la méthode des K-moyennes)

Ce premier partitionnement ayant été obtenu, on peut le raffiner en parcourant les graphes  $G_{m-1}, \dots, G_0$  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'algorithme 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 partitionnement 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
     $d(i, \mathbf{m}_c) \leftarrow K_{ii} - \frac{2 \sum_{j \in \pi_c(t)} \text{poids}(j) \cdot K_{ij}}{\sum_{j \in \pi_c(t)} \text{poids}(j)} + \frac{\sum_{j,l \in \pi_c(t)} \text{poids}(j) \cdot \text{poids}(l) \cdot K_{jl}}{(\sum_{j \in \pi_c(t)} \text{poids}(j))^2}$ 
fin
Trouver  $c^*(i) = \text{argmin}_c d(i, \mathbf{m}_c)$  (on s'occupe des égalités de façon arbitraire)
tant que ( $\neg \text{convergence}$ ) 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 partitionnement initial obtenu via l'algorithme spectral sur  $G_{m-1}$  peut déjà servir de partitionnement pour  $G_0$ . Le raffinement sert donc à améliorer les résultats de ce partitionnement.

On pourrait également raffiner directement le partitionnement 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.

## 2.5 Algorithme de principe

Notre algorithme général peut se décomposer en trois phases :

- La première phase "d'agrégation" qui va utiliser l'algorithme (1) afin de réduire et d'obtenir un graphe avec  $K$  "supernoeuds".
- Une phase de "partitionnement" qui va nous façonner notre première 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**

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

$k \leftarrow k + 1;$

**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  $G_{k-1};$

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

**fin**

---

## 3 Implémentation

### 3.1 Classes essentielles

#### 3.1.1 KMConstraints

La classe KMConstraints est une liste de paires de points qui permet de d'associer les observations entre elles afin de les rassembler dans un même groupe. On utilisera notamment cette classe pour le partitionnement de l'ensemble mais aussi pour créer les points agrégés. Tout au long du programme, cette liste servira de connexion entre les différents niveaux d'agrégation et de raffinement.

#### 3.1.2 KMInstance

Cette classe contient toutes les informations qui nous permettent de définir l'ensemble des observations, à savoir la matrice des coordonnées des points, les poids de ces derniers et surtout un attribut KMConstraints qui nous permettra de définir les classes de la partition.

#### 3.1.3 KMPartition

KMPartition est une classe destinée à récupérer toutes les classes (au sens du partitionnement). Par le biais de l'attribut KMInstance qu'elle contient, on définit deux attributs qui nous permettent de trouver les différents centroïdes rattachés aux classes et d'avoir la matrice des distances entre eux.

### 3.2 Agrégation

```
void MultiLevelAlgo::buildMultiLevelData(size_t nbNodes) {
    KMPartition partition(_instance, _instance.nbObs());

    // on crée les singletons
```

```

for(size_t i(0); i<_instance.nbObs(); i++)
    partition.shift(i,i);

while(partition.nbLabels() > nbNodes ){
    IndexedList used(partition.usedLabels());
    // définit un nouveau niveau
    _multiLevelConstraints.push_back(new KMConstraints(_input.nbObs()));

    while(!used.empty()){
        size_t const m = used.pop_random();
        if( !used.empty()){
            // calculer la distance de ce centre avec les autres
            std::multimap<Double, size_t> neighbor;
            for(auto const & c : partition.usedLabels()){
                if ( m != c)

                    neighbor.insert(
                        std::make_pair(partition.getDistanceCenter(m,c) , c));
            }
            size_t const c(neighbor.begin()->second);

            _multiLevelConstraints.back()->newCtr(
                *partition.observations(m).begin(),
                *partition.observations(c).begin());

            partition.fusion(m,c);

            // si plusieurs plusieurs plus prêts : tirer au hasard (apres)
            used.erase(c);
        }
    };
};
}

```

### 3.3 Raffinement

```

void MultiLevelAlgo::refine() {
    // lancer le KMEANS sur chaque niveau en partant du plus élevé
    //(celui qui contient le moins de noeuds)
    // à chaque fois on initialise avec le niveau précédent (sauf le premier!)
    // Pour le premier faire un appel à random(0);
    KMInstance instance;
    Aggregations aggregations;
    Timer timer;
    // pour chaque level
    for (size_t level(0); level <= _multiLevelConstraints.size(); ++level) {

```

```

// ! on parcourt à l'envers
buildInstance(_multiLevelConstraints.size() - level, instance, aggregations);
KMInput input(instance, _input.maxNbLabels());
// initialiser cette input avec la solution courante
// attention il faut utiliser aggregation
//pour faire les neuds agrégés et la solution courante
if(level==0){
    input.random(0);
    for (size_t i(0); i < _input.nbObs(); ++i) {
        _input.shiftForced(i, input.label(aggregations.newIds[i]));
    }
} else {
    for (size_t i(0); i < _input.nbObs(); ++i) {
        input.shiftForced(aggregations.newIds[i], _input.label(i));
    }
}

// on lance l'algo (on pourra ici faire un lancement
// des K-moyennes tous les X niveaux pour diminuer
// le temps total du partitionnement

HMeans<false>()(input);
std::cout << std::setw(10)<<_multiLevelConstraints.size() - level;
std::cout << std::setw(10)<<input.ite();
std::cout << std::setw(10)<<timer.elapsed();
std::cout << std::setw(20)<<input.cost();
std::cout << std::endl;
// sauvegarde de la solution
input.computeCenters();
for (size_t i(0); i < _input.nbObs(); ++i) {
    _input.shiftForced(i, input.label(aggregations.newIds[i]));
}
for (size_t i(0); i < _input.nbObs(); ++i) {
    input.shiftForced(aggregations.newIds[i], _input.label(i));
}
}
}

```

## 4 Présentation de quelques instances pour les tests

Pour effectuer les différents tests pour le programme, nous avons une sélection de quatorze fichiers data qui sont des documents texte qui recensent les données sous forme de deux parties, la première ligne nous donne le nombre de points et le nombre de coordonnées qui les caractérisent, s'ensuit l'énumération des coordonnées pour chaque point.

Ce qui est intéressant dans ce set de données, c'est qu'elles sont liées à des phénomènes naturels, comme par exemple sur la biologie, et que de plus le nombre d'observations est très varié. Ainsi la robustesse et l'efficacité des algorithmes sont testées, ce qui nous permet d'être plus confiant dans les résultats que l'on obtient.

```
150 4
5.1 3.5 1.4 0.2
4.9 3.0 1.4 0.2
4.7 3.2 1.3 0.2
4.6 3.1 1.5 0.2
5.0 3.6 1.4 0.2
5.4 3.9 1.7 0.4
4.6 3.4 1.4 0.3
5.0 3.4 1.5 0.2
4.4 2.9 1.4 0.2
4.9 3.1 1.5 0.1
5.4 3.7 1.5 0.2
4.8 3.4 1.6 0.2
4.8 3.0 1.4 0.1
4.3 3.0 1.1 0.1
5.8 4.0 1.2 0.2
5.7 4.4 1.5 0.4
5.4 3.9 1.3 0.4
5.1 3.5 1.4 0.3
5.7 3.8 1.7 0.3
```

FIGURE 2 – Extrait d'un fichier data

## 5 Tests et Analyses

Ce qui suit n'est pas une liste exhaustive de tous les tests menés mais nous avons fait le choix de les classer en trois catégories, basées sur le nombre de points, car les résultats recoupent les conclusions que l'on a obtenues.

### 5.1 Essais sur le fichier wine.dat (instance 1)

On a tout d'abord testé notre programme sur un fichier avec peu de points(178). Les tableaux suivants rassemblent nos résultats :

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	6	7 - 3 - 5 - 5 - 7 - 4	31	524418
50	3	12 - 4 - 6	22	274247
500	1	13	13	538173

TABLE 1 – Résultats pour 10 classes (instance 1)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	6	4 - 2 - 3 - 3 - 5 - 3	20	58005
50	3	5 - 4 - 3	12	61135
500	1	7	7	157347

TABLE 2 – Résultats pour 25 classes (instance 1)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	6	2 - 2 - 3 - 2 - 2 - 2	13	19038
50	3	3 - 2 - 2	7	18172
500	1	7	7	55420

TABLE 3 – Résultats pour 50 classes (instance 1)

## 5.2 Essais sur le fichier segmentation.dat (instance 8)

Puis on a élaboré notre seconde batterie de test sur l'instance 8 car elle possédait un nombre de points moyens (2100). Les tableaux suivants rassemblent nos résultats :

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	8	7 - 6 - 9 - 3 - 4 - 5 - 6 - 4	43	9.737e+06
50	5	10 - 8 - 6 - 7 - 6	37	1.010e+07
400	3	14 - 14 - 13	41	9.609e+06
1000	2	13 - 13	26	1.007e+07
3000	1	42	42	9.179e+06

TABLE 4 – Résultats pour 10 classes (instance 8)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	8	14 - 8 - 12 - 8 - 8 - 7 - 6 - 8 - 8	79	4.111e+06
50	5	8 - 21 - 8 - 11 - 13	51	4.255e+06
400	3	25 - 37 - 13	75	3.941e+06
1000	2	18 - 6	24	4.119e+06
3000	1	21	21	5.108e+06

TABLE 5 – Résultats pour 25 classes (instance 8)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	8	16 - 7 - 4 - 7 - 3 - 6 - 12 - 7	62	2.567e+06
50	5	12 - 7 - 7 - 6 - 22	54	2.573e+06
400	3	16 - 10 - 25	51	2.279e+06
1000	2	28 - 14	42	2.528e+06
3000	1	29	29	2.603e+06

TABLE 6 – Résultats pour 50 classes (instance 8)

## 5.3 Essais sur le fichier letter.dat (instance 15)

Et enfin on a testé sur ce fichier letter.dat car il possède 10 fois plus de points (20 000). Les tableaux suivants rassemblent nos résultats :

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	18 - 16 - 14 - 16 - 15 - 18 - 30 - 20 - 25 - 11	183	868493
50	8	31 - 13 - 17 - 15 - 15 - 22 - 16 - 11	140	866949
400	5	36 - 12 - 15 - 29 - 16	108	861369
1000	4	58 - 17 - 15 - 30	120	861988
20000	1	38	38	887123

TABLE 7 – Résultats pour 10 classes (instance 15)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	36 - 19 - 17 - 18 - 25 - 21 - 15 - 34 - 25 - 24	234	622828
50	8	38 - 16 - 26 - 29 - 30 - 43 - 36 - 24	242	628076
400	5	66 - 37 - 33 - 14 - 41	191	627338
1000	4	40 - 68 - 19 - 26	153	626962
20000	1	66	66	626243

TABLE 8 – Résultats pour 25 classes (instance 15)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	47 - 21 - 15 - 11 - 7 - 9 - 13 - 12 - 60 - 21	216	489627
50	8	35 - 56 - 14 - 27 - 28 - 12 - 77	249	479613
400	5	102 - 24 - 21 - 13 - 35	195	502102
1000	4	68 - 28 - 39 - 68	203	486241
20000	1	151	151	482586

TABLE 9 – Résultats pour 50 classes (instance 15)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	44 - 19 - 18 - 55 - 13 - 12 - 12 - 17 - 15 - 61	266	333205
50	8	32 - 25 - 10 - 15 - 11 - 20 - 21 - 38	172	324756
400	5	34 - 23 - 31 - 58 - 64	210	316548
1000	4	32 - 22 - 21 - 49	124	327787
20000	1	59	59	322654

TABLE 10 – Résultats pour 150 classes (instance 15)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	22 - 10 - 8 - 9 - 9 - 12 - 10 - 20 - 16 - 64	180	188375
50	8	20 - 16 - 8 - 11 - 13 - 10 - 24 - 23	125	190568
400	5	33 - 19 - 15 - 16 - 17	100	190720
1000	4	22 - 19 - 16 - 42	99	189539
20000	1	54	54	218040

TABLE 11 – Résultats pour 500 classes (instance 15)

Nb de points maximum	Nb de niveaux	Nb d'itérations des K-moyennes à chaque niveau	Nb d'itérations total	Résultat
5	10	9 - 7 - 7 - 11 - 6 - 8 - 6 - 7 - 8 - 11	80	88733
50	8	11 - 7 - 6 - 5 - 5 - 9 - 7 - 11	61	88654
400	5	11 - 8 - 5 - 8 - 9	41	89048
1000	4	11 - 10 - 7 - 10	38	89642
20000	1	26	26	109690

TABLE 12 – Résultats pour 2000 classes (instance 15)



## 5.4 Analyse

### 5.4.1 Nombre d'itérations

De manière générale, on remarque que le nombre d'itérations à chaque niveau est inférieur au nombre d'itérations si on applique directement les K-moyennes. C'est un résultat que l'on cherchait à obtenir pour valider la méthode du multi-level.

Cependant si on se réfère au nombre total d'itérations on a une nette augmentation dès lors qu'on utilise l'algorithme multi-level ; il serait donc peut-être intéressant de chercher une meilleure méthode qui diminuerait cet écart, même s'il n'a une influence que sur le temps de calcul et que pour le moment ce n'est pas notre objectif premier.

### 5.4.2 Résultats

Si maintenant on regarde les résultats que l'on obtient grâce au multi-niveau et que l'on compare ceux-ci aux résultats directement donnés par l'application des K-moyennes, on remarque très simplement que les premiers résultats sont inférieurs aux seconds. C'est un résultat intéressant que l'on ne cherchait pas forcément à avoir mais dans certains cas on améliore grandement la fonction objectif (notamment pour des problèmes avec un nombre faible de points.).

On remarque aussi que plus on a de classes, plus le résultat est faible mais cela semble logique du fait que la fonction objectif est la mesure de la distance des points par rapport aux centroïdes auxquels ils sont affectés, donc plus il y a de classes plus le point est proche de son centroïde donc plus les distances entre les points et leurs centroïdes sont faibles.