

Contrôle écrit INF442

— Traitement des données massives —

École Polytechnique, Département informatique

Frank Nielsen

Promotion X2014
Mercredi 8 juin 2016
14h-17h

Informations générales

- Les exercices sont tous *indépendants* les uns des autres, et peuvent donc être traités dans n'importe quel ordre sur la copie.
- On appréciera la *clarté* (en premier lieu) et la *concision* des réponses (en second lieu). Tout document du cours INF442 autorisé (polycopié, memento C++ et planches des cours en amphithéâtre).
- Tous les problèmes sont structurés en plusieurs sous-problèmes. Indiquez précisément la numérotation des sous-problèmes que vous traitez sur vos feuilles d'examen.
- Prenez 10 minutes de votre temps pour bien lire les énoncés avant de commencer, et gardez au moins 5 à 10 minutes à la fin pour la relecture finale.

Barème

Les problèmes ci-dessous comptent pour la note finale de la façon suivante :

Problème	%
1 — Sur la loi d'Amdahl	10%
2 — Sur l'hypercube	20%
3 — Sur l'arbre recouvrant de poids minimal	40%
4 — Sur quelques extensions des k -moyennes	30%

Pour chaque exercice, le pourcentage du barème de chaque question est indiqué *relativement* à l'exercice (la somme des pourcentages des questions d'un même exercice est donc 100%).

Problème 1 : (10%) Accélération et loi d'Amdahl

- I-1. (10%) Quelle est l'accélération théorique obtenue d'un programme séquentiel P qui peut-être parallélisé sur une fraction $\alpha_{\text{par}} \in (0, 1)$ du code sur un hypercube de dimension d ?
- I-2. (20%) Soit un programme séquentiel P qui peut-être parallélisé à 60% ($\alpha_{\text{par}} = 0.6$). Quelle est
- I-2-a. (10%) l'accélération théorique obtenue pour $M = 3$ machines ?
 - I-2-b. (5%) l'accélération asymptotique quand $M \rightarrow \infty$?
 - I-2-c. (5%) l'efficacité asymptotique de sa parallélisation quand $M \rightarrow \infty$?
- I-3. (35%) Soit un programme séquentiel P qui finit son exécution en un temps t . Quel est le pourcentage α_{par} de parallélisme à réaliser dans P pour que ce programme finisse son exécution en temps βt sur $M > 1$ machines (avec $0 < \beta < 1$) ? Vérifier votre formule pour $\beta = \frac{1}{M}$, et expliquer pourquoi on doit nécessairement avoir $\beta \geq \frac{1}{M}$.
- I-4. (35%) Soit maintenant un programme séquentiel $P = P_1; \dots; P_n$ qui est la succession de n programmes séquentiels P_1, \dots, P_n . Notons l_i les bornes inférieures et u_i les bornes supérieures des pourcentages de parallélisme possible du programme P_i (c'est-à-dire, $l_i \leq \alpha_{\text{par},i} \leq u_i, \forall i \in \{1, \dots, n\}$ où $\alpha_{\text{par},i}$ est le pourcentage de parallélisation du programme P_i).
- I-4-a. (20%) Sachant que le programme P_i demande s_i pourcentage du temps total d'exécution de P , donner une formule pour calculer le *pourcentage moyen* de parallélisation $\bar{\alpha}_{\text{par}}$ de P , et en déduire une borne inférieure et une borne supérieure sur l'accélération globale du programme P parallélisé sur M machines.
 - I-4-b. (15%) Donner votre encadrement pour la configuration suivante : $P = P_1; P_2$ ($n = 2$) avec $s_1 = \frac{1}{3}, l_1 = u_1 = \alpha_{\text{par},1} = \frac{1}{2}$ et $s_2 = \frac{2}{3}, l_2 = u_2 = \alpha_{\text{par},2} = \frac{3}{4}$.

Problème 2 : (20%) Topologie de l'hypercube et routage

Soit un hypercube H_d de dimension d . Pour un nœud $v \in H_d$ de cet hypercube, on note $\text{dec}(v)$ son code *décimal*, $\text{bin}(v)$ son code *binaire*, et $\text{Gray}(v)$ son code de *Gray*. On note $D_H(v, v')$ la *distance de Hamming* entre deux nœuds v et v' de l'hypercube.

- II-1. (5%) À quelle distance sur l'hypercube H_4 se trouve le nœud v du nœud v' lorsque $\text{Gray}(v) = 0110$ et $\text{Gray}(v') = 1101$?
- II-2. (10%) Combien de nœuds n_i sont à une distance de Hamming i d'un nœud $v \in H_d$? En déduire le nombre de nœuds sur l'hypercube en sommant les nombres de nœuds à distance $0, \dots, d$ d'un nœud donné.
- II-3. (20%)
- II-3-a. (5%) Quel est le code binaire $\text{bin}(v)$ du nœud $v \in H_5$ ayant le code décimal 25 ?
 - II-3-b. (5%) Donnez ensuite son code de Gray correspondant $\text{Gray}(v)$ (pour $\text{dec}(v) = 25$).
 - II-3-c. (5%) Donnez le code binaire correspondant au code de Gray pour $\text{Gray}(v') = 01101$.
 - II-3-d. (5%) Quel est son code décimal $\text{dec}(v')$ correspondant ?
- II-4. (10%) Démontrer qu'il n'y a pas de cycles de longueur impaire dans un hypercube.
- II-5. (40%) Décrire en *pseudo-code* MPI une procédure de routage **Ecube**(**b1**, **b2**, **T**) d'un message T sur l'hypercube H_d d'un nœud source v_1 à un nœud destination v_2 (avec v_1 et v_2 donnés en codes binaires $b_1 = \text{bin}(v_1)$ et $b_2 = \text{bin}(v_2)$, respectivement). On supposera que la variable **rank** retourne le code Gray du nœud correspondant au processus. On utilisera la fonction $\text{Gray}(v)$, la fonction booléenne **XOR** (OU exclusif) et les opérations *bloquantes* **send**(**T**, **k**) et **receive**(**T**, **k**) où k indique la dimension de l'axe sur l'hypercube où envoyer/réceptionner le message à partir du nœud appelant. On utilisera la fonction **LS1bit**(**g**) qui renvoie la position du premier bit à 1 en partant des poids faibles pour $g \neq 0$, et -1 si $g = 0$ (par exemple, **LS1bit**(0010)=1 et **LS1bit**(1001)=0), et la fonction **Barrier**() qui est une barrière de synchronisation pour tous les processus.
- Sur le modèle de coût de communication (α, τ) où α est le temps de latence pour initier une communication, et τ le temps requis par bit transféré sur une arête de communication, quelle est la complexité de votre algorithme de routage **Ecube** pour envoyer un message de longueur L de v_1 à v_2 avec $D_H(v_1, v_2) = l$.
- Vérifier manuellement votre algorithme en l'appelant pour $b_1 = 11001$ et $b_2 = 01001$. Comment pourrait-on améliorer cette complexité lorsque le temps de latence $\alpha > 0$ est important ?
- II-6. (15%)
- II-6-a. (7%) Dessinez trois arbres recouvrants du cube H_3 enraciné à un nœud donné v , chacun de ces arbres devant passer par une arête incidente à v distincte. Est-ce que vos arbres recouvrants partagent des arêtes communes ?
 - II-6-b. (8%) Sur un hypercube H_d , avec d que l'on supposera pair, on sait qu'il existe $\frac{d}{2}$ arbres recouvrants deux à deux *arêtes-disjoints* (c'est-à-dire, ne partageant *aucune* arête commune). Sur le modèle d -port de H_d où l'on peut *simultanément* envoyer des messages sur les d arêtes incidentes de n'importe quel nœud, quelle est la complexité d'une opération de communication d'un nœud à un autre pour envoyer un message de longueur L en supposant le temps de latence nul ($\alpha = 0$) sur le modèle (α, τ) de communication ?

Problème 3 : (40%) *L'arête la plus vitale pour l'arbre couvrant de poids minimal*

Soit $G = (V, E, w)$ un graphe non orienté connexe pondéré avec V ($n = |V|$) l'ensemble des n sommets, E ($m = |E|$) l'ensemble des m arêtes et $\forall e \in E, w(e) \geq 0$ les poids associés aux arêtes.

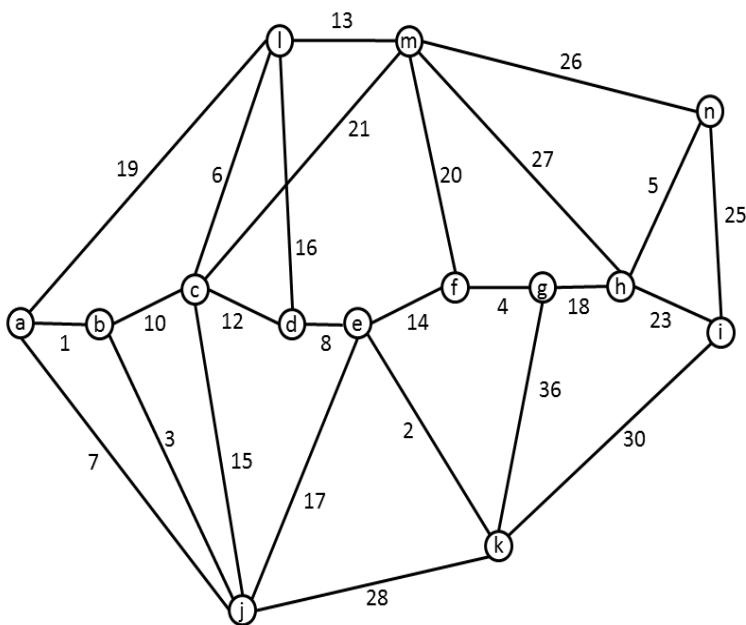


FIGURE 1 – Le graphe exemple G .

Un *arbre couvrant* T de G est un sous-ensemble d'arêtes qui connectent tous les sommets sans créer de cycle. Un *Arbre Couvrant de Poids Minimal* (ACPM, ou en anglais, *Minimum Spanning Tree* — MST) T^* est un arbre couvrant dont la somme des poids des arêtes qui le constituent $w(T^*) = \sum_{e \in T^*} w(e)$ est minimale. Nous supposons que tous les poids sont *différents* afin d'avoir l'unicité de l'ACPM. On note $w_{\text{ACPM}}(G)$ le poids de l'ACPM de G .

La définition d'un ACPM peut se généraliser aux graphes non connexes : dans le cas d'un graphe non connexe, nous parlons alors de *Forêt Couvrante de Poids Minimal* (FCPM, ou en anglais, *Minimum Spanning Forest* — MSF). Une FCPM est donc l'union des ACPMs de ses composantes connexes.

- III-1. (10%) Calculer à la main l'ACPM T^* du graphe exemple G représenté dans la Figure ci-dessus (Figure 1), et donner son poids $w(T^*) = w_{\text{ACPM}}(G)$.

On pourra utiliser l'algorithme de Kruskal qui commence par trier les arêtes par poids croissant, initialise $T = \emptyset$, puis itérativement parcourt ces arêtes triées en ajoutant l'arête courante e dans l'arbre ($T \leftarrow T \cup \{e\}$) si et seulement si $T \cup \{e\}$ ne forme pas de cycle.

III-2. (25%) Donner deux pseudo-codes correspondant (1) à l'algorithme *séquentiel* et (2) à l'algorithme *parallèle* pour déterminer l'ACPM T^* de G . Vous pouvez utiliser par exemple l'algorithme de Kruskal, et on notera p le nombre de processus à mémoire distribuée du cluster (on supposera que n est un multiple de p). Analyser le temps parallèle, l'accélération et l'efficacité de votre parallélisation.

On pourra utiliser une structure de données de type **union-find** pour gérer les *ensembles disjoints* avec les opérations suivantes :

- **Find** : Détermine dans quel sous-ensemble un élément x se trouve. **Find** retourne un élément représentatif du sous-ensemble si bien qu'on peut déterminer si deux éléments appartiennent au même sous-ensemble en comparant leur élément représentatif.
- **Union** : Fusionne deux sous-ensembles en un sous-ensemble. (L'élément représentatif est choisi comme un des deux éléments représentatifs.)

On ne demande pas d'implémenter cette structure de données mais juste de faire appel aux primitives **Find** et **Union**.

On rappelle qu'un *regroupement hiérarchique* est représenté par un *dendrogramme* qui est un arbre binaire (chaque nœud de l'arbre a au plus deux fils). Le *regroupement hiérarchique à saut minimum* (RHSM, ou *single linkage hierarchical clustering* — SLHC) est une méthode de regroupement hiérarchique qui consiste à fusionner à chaque étape les deux clusters qui ont la distance la plus petite (le saut minimum). Considérons n éléments à regrouper hiérarchiquement, et soit D la matrice des distances entre ces éléments. L'algorithme du regroupement hiérarchique à saut minimum est donné ci-dessous.

```

1: Entrée :  $n$  éléments,  $D$  matrice des distances entre des éléments
2: Sortie : Dendrogramme  $T$  du regroupement hiérarchiquement  $C$  des  $n$  éléments,
    $N$  l'ensemble des sommets de  $T$ 
3: Début
4: Chaque élément  $i$  constitue un cluster  $C_i = \{i\}$ , pour  $i = 1, \dots, n$ ;
5:  $N = \{C_i : i = 1, \dots, n\}$ ;
6:  $T = \emptyset$ ;
7:  $C = N$ ;
8: while  $|C| > 1$  do
9:    $(C_i^*, C_j^*) = \arg \min_{C_i \in C, C_j \in C} ( \min_{x \in C_i, y \in C_j} d(x, y) )$ ;
10:   $C' = C_i^* \cup C_j^*$ ;
11:   $N = N \cup C'$ ;
12:   $T = T \cup \{(C', C_i^*), (C', C_j^*)\}$ ;
13:   $C = C \cup C' \setminus \{C_i^*, C_j^*\}$ 
14: end while
15: return  $T$ ;

```

Algorithm 1: Regroupement Hiérarchique à Saut Minimum (RHSM).

III-3. (15%) Montrer comment l'ACPM peut être utilisé pour calculer le Regroupement Hiérarchique à Saut Minimum (RHSM, avec sa représentation en dendrogramme).

$\forall e \notin T^*, C(e)$ est le chemin dans T^* qui connecte les deux extrémités de l'arête e . Ainsi, $C(e) \cup \{e\}$ forme un unique cycle dans $T^* \cup \{e\}$. En supprimant l'arête $e = (u, v) \in T^*$ du graphe $(T^* \setminus \{e\})$, on déconnecte T^* en deux sous-arbres T_u et T_v , où $u \in T_u$ et $v \in T_v$. $\forall e \in T^*, C^{-1}(e)$ est défini comme le sous-ensemble des arêtes qui reconnectent T_u et T_v pour former un arbre couvrant.

III-4. (15%) Implémenter en parallèle une méthode qui permet de déterminer $C(e)$ pour chaque $e \notin T^*$ et $C^{-1}(e)$ pour chaque $e \in T^*$.

Une *arête de remplacement* $r(e)$ d'une arête $e \in T^*$ est une arête par laquelle e doit être remplacée pour créer un nouvel ACPM si e est supprimée de G .

III-5. (15%) Écrire une méthode qui permet de retourner les arêtes de remplacement, c'est à dire $r(e)$ pour tout $e \in E$.

L'*arête la plus vitale* de G par rapport à l'ACPM est l'arête $e \in E$ telle que $w_{\text{ACPM}}(G \setminus \{e\}) \geq w_{\text{ACPM}}(G \setminus \{e'\})$ pour toute $e' \in E$. C'est-à-dire, l'arête la plus vitale est l'arête de G dont sa suppression induit la plus forte augmentation du poids de l'ACPM de $G \setminus \{e\}$. Ainsi l'arête la plus vitale doit nécessairement appartenir à l'arbre T^* (sinon le poids de l'ACPM ne changerait pas).

En général, l'arête la plus vitale permet de déterminer la composante la plus critique d'un système donné dont la fonctionnalité peut se modéliser par un problème de graphe. Si cette composante venait à défaillir ou à être attaquée, cela engendrerait la plus grande perturbation de la performance du système. En terme de graphe, la suppression de l'arête correspondante impliquerait une perturbation maximale de la propriété du graphe. Les arêtes les plus vitales ont diverses applications notamment dans l'étude de la fiabilité des systèmes, leur résilience ou encore dans le domaine de la sécurité.

III-6. (20%) Quelle est l'arête la plus vitale au regard de l'ACPM du graphe G représenté dans la Figure de notre exemple de graphe (Figure 1) et l'augmentation de poids qu'elle engendre? Écrire une méthode qui permet de retourner l'arête la plus vitale. Vous pouvez faire appel aux méthodes précédentes.

(Cet exercice est proposé par Sonia Toubaline et Frank Nielsen.)

Problème 4 : (30%) Sur quelques généralisations des k -moyennes

Soit un ensemble de n données $X = \{x_1, \dots, x_n\}$, chaque donnée x_i ayant d attributs : $x_i = (x_i^{(1)}, \dots, x_i^{(d)})$. On note $[k] = \{1, \dots, k\}$ pour tout entier k positif. Dans ce problème, nous nous proposons de généraliser le regroupement par les k -moyennes.

Cet exercice est composé de **trois parties indépendantes**.

(35%) **IV-A. Des k -moyennes aux k -modes.**

Considérons les données de X comme toutes *catégoriques*. C'est-à-dire, que $x_i^{(j)} \in E_j$ où E_j est un ensemble de $m_j = |E_j|$ étiquettes (tags) $t_{j,l}$ avec $l \in \{1, \dots, m_j\}$. Par exemple, prenons $d = 3$ pour une illustration, avec $E_1 = \{\text{Femme}, \text{Homme}\}$, $E_2 = \{X13, X14, X15\}$ et $E_3 = \{\text{athlétisme}, \text{aviron}, \text{basket-ball}, \text{course d'orientation et raid}, \text{équitation}, \text{escalade}, \text{escrime}, \text{football}, \text{golf}, \text{handball}, \text{judo}, \text{natation}, \text{rugby}, \text{tennis}, \text{volley-ball}\}$. Alors, chaque donnée $x_i \in X$ est un triplet (sexe, promotion, discipline).

On rappelle que la distance de Hamming entre deux données catégoriques est :

$$D_H(x_i, x_j) = \sum_{l=1}^d 1_{[x_i^{(l)} \neq x_j^{(l)}]}$$

C'est-à-dire que la distance de Hamming entre deux données catégoriques compte le nombre d'étiquettes différentes.

- IV-A-1. (10%) Prouver que la distance de Hamming D_H satisfait l'*inégalité triangulaire* $D_H(p, q) + D_H(q, r) \geq D_H(p, r) \forall p, q, r$, et est donc bien une distance métrique. Pour cela, on commencera par prouver la propriété élémentaire suivante : la somme de deux distances métriques est une distance métrique.
- IV-A-2. (15%) On veut définir un "centre" c pour l'ensemble X comme le minimiseur de la fonction de coût :

$$l(x; X) = \frac{1}{n} \sum_{i=1}^n D_H(x, x_i)$$

On appellera *mode*¹ un tel minimiseur de $l(x; X)$.

Donner une caractérisation du mode (centre de Hamming) pour un cluster X et un algorithme pour en calculer un en temps linéaire. Pour cela, on notera $n_{i,j} = |\{x \in X : x^{(i)} = t_{i,j}\}|$ ($\forall i \in [d], j \in [m_i]$, avec $|\cdot|$ le cardinal d'un ensemble), $n_i^* = \max_{j \in [m_i]} n_{i,j}$, et $T_i = \{t_{i,j} : n_{i,j} = n_i^*\}$.

Montrer que le minimiseur de $l(x; X)$ n'est pas forcément unique en donnant un exemple 2D ($d = 2$) et n'appartient pas forcément à X en donnant un autre exemple.

- IV-A-3. (10%) En déduire que l'on peut généraliser l'algorithme des k -moyennes de Lloyd dans le contexte des données catégoriques en utilisant la distance de Hamming, et donner la fonction de coût à minimiser. Préciser une méthode d'initialisation des k centres. Par analogie aux k -moyennes, cet algorithme s'appelle les k -modes. On note $M = \sum_{j=1}^d m_j$ le nombre total d'étiquettes toute dimension confondue. Prouver la convergence monotone des k -modes en un nombre fini d'itérations que l'on bornera.

1. En probabilité, le ou les modes d'une probabilité discrète sont le ou les maxima de sa fonction de masse.

(45%) **IV-B. Des k -moyennes aux k -moyennes floues**

Le clustering par les k -moyennes n'est pas efficace lorsque les clusters ne sont pas bien séparables. Aussi, les *k -moyennes floues* considèrent un *poids d'appartenance* $w_{i,j} > 0$ du point x_i au cluster C_j pour $i \in [n], j \in [k]$, avec $W_i = \sum_{j=1}^k w_{i,j} = 1$ et $0 < \sum_{i=1}^n w_{i,j} < n$ (quand $k > 1$, avec $j \in [k]$). Soit $C = \{c_1, \dots, c_k\}$ l'ensemble des k centres des clusters. Les k -moyennes floues (appelé en anglais *fuzzy c-means*, FCM) cherchent à minimiser la fonction de coût suivante :

$$J_p(X; C; w) = \sum_{j=1}^k \sum_{i=1}^n w_{i,j}^p \|x_i - c_j\|^2 = \sum_{j=1}^k \sum_{i=1}^n w_{i,j}^p d_{i,j}^2, \quad (1)$$

pour un paramètre p de puissance fixé au préalable, avec $p \in [1, \infty)$ (pas nécessairement entier). On note $d_{i,j}^2 = \|x_i - c_j\|^2$. L'algorithme de clustering minimisant $J_p(X; C; w)$ procède comme suit :

- Initialiser les poids $w_{i,j}$. Par exemple, $w_{i,j}^{(-1)} = \infty, w_{i,j}^{(0)} = \frac{1}{k}, \forall i \in [n], \forall j \in [k]$. $t \leftarrow 0$
- Tant que $\max_{i \in [n], j \in [k]} |w_{i,j}^{(t)} - w_{i,j}^{(t-1)}| > \epsilon$, faire :
 - Calculer les centres $c_j^{(t)}$ des clusters comme les centroïdes en fonction des poids $w_{i,j}^{(t)}$ et de p (cf. Eq. 1).
 - Mettre à jour les poids d'appartenance dans $w_{i,j}^{(t+1)}$
 - $t \leftarrow t + 1$

IV-B-1. (10%) Donner la formule du centroïde Euclidien c (l'unique minimiseur de $l(X; c) = \sum_{i=1}^n w_i \|x_i - c\|^2 = \sum_{i=1}^n w_i d_{i,j}^2$) pour un ensemble de n points x_i avec des poids positifs $w_i > 0$ pas forcément normalisés.

En déduire la formule du centroïde $c_i^{(t)}$ à partir des poids $w_{i,j}^{(t)}$ et du paramètre p .

IV-B-2. (15%) Montrez que la formule de mise à jour des poids minimisant $J_p(X; C; w)$ avec les centres fixés sous les n contraintes $W_i = 1, \forall i \in [n]$ est :

$$w_{i,j}^{(t+1)} = \left(\sum_{l=1}^k \left(\frac{\|x_i - c_j^{(t)}\|}{\|x_i - c_l^{(t)}\|} \right)^{\frac{2}{p-1}} \right)^{-1} = \frac{1}{\sum_{l=1}^k \left(\frac{d_{i,j}^{(t)}}{d_{i,l}^{(t)}} \right)^{\frac{2}{p-1}}} = \frac{\left(d_{i,j}^{(t)} \right)^{-\frac{2}{p-1}}}{\sum_{l=1}^k \left(d_{i,l}^{(t)} \right)^{-\frac{2}{p-1}}},$$

avec $d_{i,l}^{(t)} = \|x_i - c_l^{(t)}\|$.

Pour cela, on utilisera la *méthode des Lagrangiens pour l'optimisation sous contraintes d'égalité*.

Méthode des Lagrangiens :

On donne la recette appliquée à un petit exemple illustratif pour la minimisation de $f(x, y) = x^2 + y^2$ sous la contrainte d'égalité $x + y = 1$:

- Re-écrire les contraintes d'égalité $l = r$ en transférant tous les termes à gauche de l'égalité et en mettant un zéro sur la partie droite de l'égalité : $l = r$ devient $l - r = 0$. Dans notre exemple, $x + y = 1 \Rightarrow x + y - 1 = 0$.
- Construire la fonction Lagrangienne. Ici, $L(x, y, \lambda) = x^2 + y^2 + \lambda(x + y - 1)$. Pour une fonction f de \mathbb{R}^d avec l contraintes, on construit donc le Lagrangien $L(x_1, \dots, x_d, \lambda_1, \dots, \lambda_l)$.

— Annuler les dérivées partielles pour obtenir un système à résoudre. Ici, on a :

$$\begin{aligned}\frac{\partial L}{\partial x} &= 2x + \lambda = 0, \\ \frac{\partial L}{\partial y} &= 2y + \lambda = 0, \\ \frac{\partial L}{\partial \lambda} &= x + y - 1 = 0.\end{aligned}$$

— Résoudre le système. Ici, on obtient $\lambda = -1$, et $x = y = \frac{1}{2}$.

IV-B-3. (10%) Montrez que

IV-B-3-a. (3%) l'algorithme des k -moyennes floues converge monotonement,

IV-B-3-b. (3%) $w_{i,j} \rightarrow \frac{1}{k}$ quand $p \rightarrow \infty$,

IV-B-3-c. (4%) les k -moyennes floues tendent vers les k -moyennes quand $p \rightarrow 1$ (avec $w_{i,j} \in \{0, 1\}$).

IV-B-4. (10%) Écrire en *pseudo-code* un algorithme qui parallélise les k -moyennes floues sur un cluster de M machines en utilisant le formalisme de MPI. On supposera que le nombre n de données x_i est divisible par M , et que les données sont déjà présentes par paquets de $\frac{n}{M}$ sur les processus locaux. Quelle est la complexité de votre algorithme parallèle ? Indiquer la grande différence au niveau des opérations de communication globale avec l'algorithme parallèle pour les k -moyennes.

(20%) **IV-C. Des k -moyennes au clustering probabiliste avec les modèles de mélanges.**

La densité d'une loi de probabilité Gaussienne (normale) isotropique de \mathbb{R}^d d'espérance $\mu \in \mathbb{R}^d$ et de matrice de covariance $\Sigma = \sigma^2 I$ pour $\sigma > 0$ fixé (avec I la matrice identité) est :

$$p(x; \mu) = (2\pi)^{-\frac{d}{2}} \sigma^{-d} \exp\left(-\frac{1}{2\sigma^2} \|x - \mu\|^2\right).$$

IV-C-1. (10%) Montrer que le maximum de vraisemblance pour estimer μ à partir de n échantillons x_1, \dots, x_n identiquement et indépendamment distribués (avec σ fixé) est équivalent à minimiser la fonction de coût pour calculer un centroïde. En déduire que :

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i.$$

On pourra noter $l(x; \mu) = \log p(x; \mu)$ la log-vraisemblance.

Soit un modèle de mélange en statistique à $k \in \mathbb{N}$ composantes de densité

$$m(x; w, \theta) = \sum_{i=1}^k w_i \times p(x; \mu_i)$$

avec w un vecteur de poids positif normalisé ($\forall i \in [k], w_i > 0$ et $\sum_{i=1}^k w_i = 1$) et $\theta = \{\mu_1, \dots, \mu_k\}$ les paramètres des composantes gaussiennes isotropiques de la mixture. On note

$$l(X; w, \theta) = \sum_{i=1}^n \log m(x_i; w, \theta)$$

la log-vraisemblance du mélange pour un échantillon *identiquement et indépendamment* (iid) distribué X .

On considère l'algorithme itératif de *maximisation de vraisemblance* $\text{EM}(X, k, \sigma, \epsilon)$ pour apprendre le mélange. A l'itération t , on note $w^{(t)} = \{w_1^{(t)}, \dots, w_k^{(t)}\}$ et $\theta^{(t)} = \{\mu_1^{(t)}, \dots, \mu_k^{(t)}\}$.

$\text{EM}(X, k, \sigma, \epsilon)$

- $t \leftarrow 0$. Initialiser les poids $w_i^{(0)}$ et les centres $\mu_i^{(0)}$ pour $i \in [k]$. Par exemple, en appelant un algorithme des k -moyennes($X; C = \{c_1, \dots, c_k\}$), puis en faisant $\forall i \in [k], \mu_i^{(0)} = c_i$ et $\forall i \in [k], w_i^{(0)} = \frac{|C_i|}{n}$ avec $|C_i|$ désignant le nombre de points du i -ème cluster. Par convention, $l(X; w^{(-1)}, \theta^{(-1)}) = -\infty$.
- Tant que $|l(X; w^{(t)}, \theta^{(t)}) - l(X; w^{(t-1)}, \theta^{(t-1)})| > \epsilon$ pour un seuil ϵ fixé
 - **étape E**. Calculer les probabilités d'appartenance des x_i aux composantes :

$$\forall i \in [n], \forall j \in [k], \quad l_{j,i}^{(t)} = \frac{w_j^{(t)} p(x_i; \mu_j^{(t)})}{\sum_{l=1}^k w_l^{(t)} p(x_i; \mu_l^{(t)})}$$

- **étape M**. Mettre à jour les paramètres :

$$w_j^{(t+1)} = \frac{1}{n} \sum_{i=1}^n l_{j,i}^{(t)}, \quad \mu_j^{(t+1)} = \frac{\sum_{i=1}^n l_{j,i}^{(t)} x_i}{\sum_{i=1}^n l_{j,i}^{(t)}}$$

- $t \leftarrow t + 1$

IV-C-2. (10%) Montrer que lorsque $\sigma \rightarrow 0$, minimiser la log-vraisemblance tend à minimiser la fonction de coût des k -moyennes. Quel est le nombre d'itérations maximal pour l'algorithme des k -moyennes de Lloyd ? Et quel est le nombre d'itérations maximal pour cet algorithme k -EM(X, k, σ, ϵ) si l'on fixe $\epsilon = 0$ dans le seuil de convergence ?

Contrôle écrit avec **corrigé** INF442
— Traitement des données massives —

École Polytechnique, Département informatique

Frank Nielsen

Promotion X2014
Mercredi 8 juin 2016
14h-17h

Informations générales

- Les exercices sont tous *indépendants* les uns des autres, et peuvent donc être traités dans n'importe quel ordre sur la copie.
- On appréciera la *clarté* (en premier lieu) et la *concision* des réponses (en second lieu). Tout document du cours INF442 autorisé (polycopié, memento C++ et planches des cours en amphithéâtre).
- Tous les problèmes sont structurés en plusieurs sous-problèmes. Indiquez précisément la numérotation des sous-problèmes que vous traitez sur vos feuilles d'examen.
- Prenez 10 minutes de votre temps pour bien lire les énoncés avant de commencer, et gardez au moins 5 à 10 minutes à la fin pour la relecture finale.

Barème

Les problèmes ci-dessous comptent pour la note finale de la façon suivante :

Problème	%
1 — Sur la loi d'Amdahl	10%
2 — Sur l'hypercube	20%
3 — Sur l'arbre recouvrant de poids minimal	40%
4 — Sur quelques extensions des k -moyennes	30%

Solution

- I-1. Sur l'hypercube H_d de dimension d , on a $M = 2^d$ nœuds, et donc $\text{speedup}(H_d) = \frac{1}{\alpha_{\text{seq}} + \frac{\alpha_{\text{par}}}{2^d}}$.
- I-2. I-2-a. Nous avons $\alpha_{\text{par}} = \frac{6}{10} = \frac{3}{5}$ et $\alpha_{\text{seq}} = 1 - \alpha_{\text{par}} = \frac{2}{5}$. On applique la loi d'Amdahl : $\text{speedup}(M) = \frac{1}{\alpha_{\text{seq}} + \frac{\alpha_{\text{par}}}{M}}$, et on trouve $\text{speedup}(3) = \frac{5}{3} \simeq 1,67 \leq 3$.
- I-2-b. L'accélération asymptotique pour $M \rightarrow \infty$ est $\frac{1}{\alpha_{\text{seq}}} = 5/2 = 2,5$.
- I-2-c. L'efficacité asymptotique est $\frac{\text{speedup}(M)}{M} \sim \frac{1}{\alpha_{\text{seq}} M} \rightarrow 0$ quand $M \rightarrow +\infty$.
- I-3. On a $\text{speedup}(M) = \frac{1}{\alpha_{\text{seq}} + \frac{\alpha_{\text{par}}}{M}} = \frac{t}{\beta t} = \frac{1}{\beta}$ et $\alpha_{\text{par}} = 1 - \alpha_{\text{seq}}$. On trouve donc $\alpha_{\text{par}} = \frac{M(1-\beta)}{M-1}$.
 Pour $\beta = \frac{1}{M}$, on a $\alpha_{\text{par}} = 1$, c'est-à-dire que le programme est 100% parallélisable. Puisque $\text{speedup}(M) = \frac{1}{\beta} \leq M$ (M est l'accélération optimale), on en déduit que $\beta \geq \frac{1}{M}$.
- I-4. I-4-a. On a par définition $\sum_{i=1}^B s_i = 1$, et

$$\sum_{i=1}^B s_i \times l_i \leq \bar{\alpha}_{\text{par}} = \sum_{i=1}^B s_i \times \alpha_{\text{par},i} \leq \sum_{i=1}^B s_i \times u_i$$

- I-4-b. Soit $P = P_1; P_2$ avec $s_1 = \frac{1}{3}, l_1 = u_1 = \alpha_{\text{par},1} = \frac{1}{2}$ et $s_2 = \frac{2}{3}, l_2 = u_2 = \alpha_{\text{par},2} = \frac{3}{4}$. On a $\bar{\alpha}_{\text{par}} = \frac{1}{2} \times \frac{1}{3} + \frac{2}{3} \times \frac{3}{4} = \frac{2}{3}$. Ainsi, on a $\text{speedup}(M) = \frac{3}{1 + \frac{2}{M}}$ et pour $M = 2$, on trouve $\text{speedup}(2) = \frac{3}{2}$.

Solution

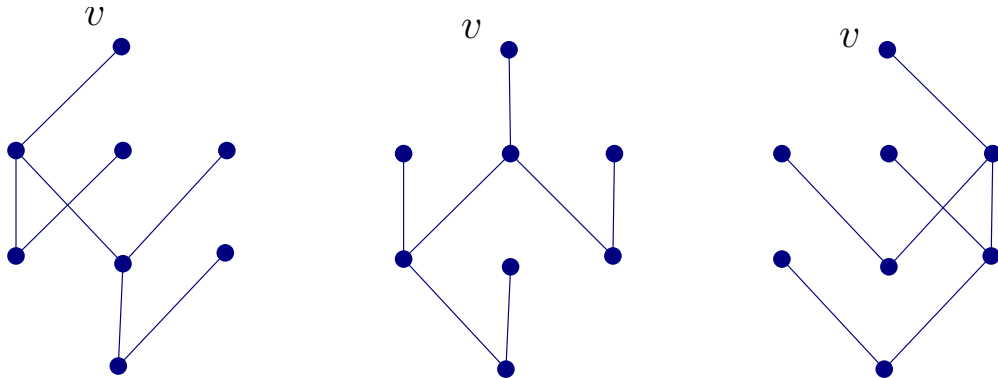
- II-1. La distance de Hamming entre v et v' avec $\text{Gray}(v) = 0110$ et $\text{Gray}(v') = 1101$ est $\text{XOR}(0110, 1101) = 3$ (nombre de bits différents).
- II-2. Puisque un nœud à distance i de v à i bits différents dans le code de Gray, on en déduit qu'il y a $n_i = \binom{d}{i} = \frac{d!}{i!(d-i)!}$ nœuds à distance i de v .
On vérifie ainsi que le nombre de nœuds de l'hypercube est $\sum_{i=0}^d n_i = \sum_{i=0}^d \binom{d}{i} = 2^d$ en additionnant tous les nombres de nœuds à distance i pour $i \in \{0, \dots, d\}$.
- II-3. II-3-a. Soit v tel que $\text{dec}(v) = 25$. En représentation binaire, on utilise 5 bits pour les 32 nœuds de H_5 , et on a $\text{bin}(v) = 11001$.
II-3-b. On convertit en code de Gray pour obtenir $\text{Gray}(v) = 10101$.
II-3-c. Soit v' tel que $\text{Gray}(v') = 01101$, on a $\text{bin}(v') = 01001$.
II-3-d. On trouve $\text{dec}(v') = 9$.
- II-4. Il n'y a pas de cycles de longueur impaire dans un hypercube puisque pour chaque arête traversée on flippe exactement un bit de la représentation du code de Gray entre les nœuds de cette arête. La longueur d'un cycle est donc le nombre de bits flippés dans la représentation du code de Gray des nœuds du cycle. Pour n'importe quel nœud du cycle, en parcourant tous les nœuds du cycle et en revenant à ce nœud initial, on doit donc nécessairement flipper un nombre pair de bits dans la représentation du code de Gray de v . Bien entendu, notons que tous les chemins de longueur paire ne sont pas forcément des cycles !
- II-5. On décrit l'algorithme de routage **E-cube** comme suit : le nœud v contenant le message l'envoie sur le lien $k = \text{LS1bit}(\text{XOR}(v, g2))$, et son voisin qui a le code $\text{XOR}(v, 2^{**k})$ le réceptionne (on flippe le k -ième bit). On répète la procédure jusqu'à temps que $\text{XOR}(v, g2) = 0$. Il y a l étapes et pour chaque étape, on doit synchroniser tous les processus avec une barrière de synchronisation.

Pour deux nœuds v et v' avec $D_H(v, v') = l$, la complexité du temps de communication est

$$O(l(\alpha + \tau L))$$

On pourrait améliorer cette complexité en faisant des communications pipelinées en partitionnant le message en r morceaux.

- II-6. Une figure illustrant les trois arbres recouvrants de H_3 partant d'un nœud donné :



Pour deux nœuds v et v' avec $D_H(v, v') = l$, la complexité du temps de communication en utilisant les $\frac{d}{2}$ arbres recouvrants (en fractionnant le message en $\frac{d}{2}$ messages de longueur $\frac{2L}{d}$) est

$$O\left(\left(\tau \frac{2L}{d}\right)\right)$$

L'algorithme Ecube en pseudo-code MPI :

Ecube(b1,b2,T)

```

g1=gray(b1);
g2=gray(b2);

v=g1;
r=XOR(v,g2);

while (r != 0 )
{
k=LS1bit(r);

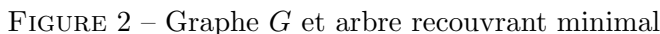
if (rank == v)
    send(T,k);
else
if (rank == XOR(v,2**k))
receive(T,k);

Barrier();

v=XOR(v,2**k); // 2**k veut dire deux puissance k
r=XOR(v,g2);
}

```

III-1. T^* est représenté sur la Figure 2 avec son poids total $w(T^*) = 119$.



Algorithme 3 est un pseudo-code de l'algorithme de Kruskal en parallèle. L'idée est de partitionner l'ensemble V sur les p processeurs ($V = \cup_{i=1}^p V_i$) de sorte que chaque processeur i traite $\frac{n}{p}$ sommets consécutifs de V_i et leur arêtes (ajuster le dernier processeur). Ensuite, chaque processeur détermine son MSF local en utilisant par exemple Algorithme 4 (Kruskal en séquentiel). Enfin, on fait des fusions deux à deux pour unir des MSFs locaux de deux processeurs jusqu'à ce qu'il ne reste qu'un seul processeur qui en 'mergeant' les deux dernier MSFs donne le MST global. Pour merger deux MSFs, on peut également utiliser Algorithme 4 (Kruskal en séquentiel).

- Calculer le MSF en local prend $O(\frac{n^2}{p})$. On a au total $\log p$ opérations ‘merge’ chacune prenant $O(n^2 \log p)$. Pendant chaque ‘merge’, au plus $O(n)$ arêtes sont envoyées. Ainsi, le temps total en parallèle est $t_p = O(n^2/p) + O(n^2 \log p)$.
- Calcul de l’accélération : $speedup(p) = \frac{t_{seq}}{t_p}$, avec $t_{seq} = O(m + n \log n)$
- Calcul de l’efficacité : $e(p) = \frac{speedup(p)}{p} = O(\frac{m+n \log n}{p})$

III-3. L'Algorithme 1 est très similaire à l'algorithme de Kruskal. L'utilisation du MST dans le single linkage hierarchical clustering peut se faire de la manière suivante : Nous trions les arêtes du MST par ordre croissant de leur poids, et initialisons chaque cluster à un sommet du MST. À chaque étape k du single linkage hierarchical clustering, la k ème arête (x^*, y^*) du MST prise selon l'ordre considéré est celle correspondante à (C_i^*, C_j^*) , c'est-à-dire $w((x^*, y^*)) =$

$$\min_{C_i \in C, C_j \in C} \left(\min_{x \in C_i, y \in C_j} d(x, y) \right).$$

Ainsi à partir du MST nous pouvons donc reconstruire le dendrogramme T du single linkage hierarchical clustering.

III-4. — Faire une Depth First Search en parallèle pour $C(e), e \notin T^*$.

— Pour $C^{-1}(e), e = (u, v) \in T^*$, faire T_u et T_v de deux couleurs. Tester les arêtes de $e' \in E \setminus T^*$ (peut se faire en parallèle). Si les deux extrémité de e' ont des couleurs différentes alors e' est une arête de remplacement.

III-5. L'arête de remplacement $r(e)$ est la plus petite arête de $C^{-1}(e)$. Il suffit d'appeler l'algorithme précédent pour déterminer $C^{-1}(e)$ et de prendre l'arête $e' = \arg \min_{f \in C^{-1}(e)} w(f)$.

III-6. Les arêtes de remplacement des arêtes de T^* sont données dans le Tableau 1.

e	(a,b)	(b,c)	(c,d)	(d,e)	(e,f)	(f,g)	(g,h)	(h,i)	(b,j)	(e,k)	(c,l)	(i,m)	(h,n)
r(e)	(a,j)	(c,j)	(l,d)	(e,j)	(m,f)	(m,n)	(m,n)	(i,n)	(a,j)	(j,k)	(d,l)	(f,m)	(i,n)
Δw	6	5	4	9	6	22	8	2	4	26	10	7	20

Tableau 1 – Arêtes de remplacement des arêtes de T^*

L'arête la plus vitale au regard du MST est alors (e, k) dont la suppression engendre une augmentation du poids du MST de 26.

C'est l'arête e^* telle que $w(r(e^*)) - w(e^*) = \max_{e \in T^*} w(r(e)) - w(e)$. Calculer donc $r(e)$ pour tout $e \in T^*$ et ensuite déterminer e^* parmi ces arêtes là.

Plus de détails dans le papier [1] (1991).

```

1: Entrée :  $G = (V, E, w)$  un graphe non orienté connexe pondéré
2: Sortie : le MST  $T$ 
3: Début
4:  $T = \emptyset$ ;
5: Trier les arêtes de  $E$  par ordre croissant des poids;
6: for each  $e = (u, v) \in E$  pris dans cet ordre do
7:   if l'ajout de  $e$  à  $T$  ne cre pas de cycle then
8:      $T = T \cup \{e\}$ ;
9:   end if
10: end for
11: return  $T$ ;

```

Algorithm 2: ALGORITHME SÉQUENTIEL DE KRUSKAL - APPROCHE 1


```

1: Entrée :  $G = (V, E, w)$  un graphe non orienté connexe pondéré
2: Sortie : le MST  $T$ 
3: Début
4: MPIinit()
5: MPI_Status status;
6: MPI_Comm_size(MPI_COMM_WORLD, &totalnodes);
7: MPI_Comm_rank(MPI_COMM_WORLD, &mynode);
8: Partitionner  $V$  en  $V_i$  pour  $i = 1, \dots, \text{totalnodes}$ ;
9: // Revient à diviser les indice de 1 à  $n$  sur totalnodes processeurs
10: // Chaque processeur détermine son MSF en local
11:  $T\_local = \emptyset$ ;
12: for each  $v \in V_i$  do
13:   // Créer un ensemble disjoint pour chaque sommet
14:   Make_Set( $v$ );
15: end for
16: Trier les arêtes de  $E_i$  par ordre croissant des poids;
17: for each  $e = (u, v) \in E_i$  pris dans cet ordre do
18:   // Find_Set( $v$ ) retourne l'ensemble disjoint auquel appartient  $v$ 
19:   if Find_Set( $u$ )  $\neq$  (v) then
20:      $T\_local = T\_local \cup \{e\}$ ;
21:     // Unir les deux MSFs
22:     Union( $u, v$ );
23:   end if
24: end for
25: Fusionner les MSFs deux à deux jusqu'à ce qu'il ne reste qu'un processus
26: IDÉE du merge deux à deux : un proc  $a$  envoie son MSF local au proc  $b$ 
27: MPI_Send( $T\_local$ , count, MPI_Datatype,  $b$ , tag, MPI_Comm comm);
28: MPI_Recv( $T\_recv$ , count, MPI_Datatype,  $a$ , tag, MPI_Comm comm, MPI_Status);
29: Le proc  $b$  calcul son nouveau MSF local à partir de  $T\_local \cup T\_recv$ .
30: À la dernière itération, le dernier proc calcule le MST  $T\_global$  à partir de son MSF  $T\_local$  et
    du MSFs reçu  $T\_recv$ 
31: MPI_Finalize();
32: return  $T\_global$ ;

```

Algorithm 3: ALGORITHME DE KRUSKAL EN PARALLÈLE

```

1: Entrée :  $G = (V, E, w)$  un graphe non orienté connexe pondéré
2: Sortie : le MST  $T$ 
3: Début
4:  $T = \emptyset$ ;
5: for each  $v \in V$  do
6:   // Créer un ensemble disjoint pour chaque sommet
7:   Make_Set( $v$ );
8: end for
9: Trier les arêtes de  $E$  par ordre croissant des poids;
10: for each  $e = (u, v) \in E$  pris dans cet ordre do
11:   // Find_Set( $v$ ) retourne l'ensemble disjoint auquel appartient  $v$ 
12:   if Find_Set( $u$ )  $\neq$  ( $v$ ) then
13:      $T = T \cup \{e\}$ ;
14:     // Unir les deux MSFs
15:     Union( $u, v$ );
16:   end if
17: end for
18: return  $T$ ;

```

Algorithm 4: ALGORITHME SÉQUENTIEL DE KRUSKAL - APPROCHE 2

Solution

IV-A-1. Montrons que la distance de Hamming satisfait l'inégalité triangulaire :

Soit $D_1(p, q)$ et $D_2(p, q)$ deux distances métriques, montrons que $D(p, q) = D_1(p, q) + D_2(p, q)$ est une distance métrique. Pour cela, on doit vérifier les trois axiomes :

Identité des indiscernables. $D(p, q) \geq 0$ avec égalité si et seulement si $p = q$.

Preuve : $D(p, q) = D_1(p, q) + D_2(p, q) \geq 0$ et égal à 0 ssi $p = q$ puisque D_1 et D_2 satisfont l'identité des indiscernables.

Symétrie. $D(p, q) = D(q, p)$.

Preuve : $D(p, q) = D_1(p, q) + D_2(p, q) = D_1(q, p) + D_2(q, p) = D(q, p)$ puisque D_1 et D_2 satisfont l'axiome de la symétrie.

Inégalité triangulaire. $D(p, q) \leq D(p, r) + D(r, q)$.

Preuve : $D(p, q) = D_1(p, q) + D_2(p, q) \leq D_1(p, r) + D_1(r, q) + D_2(p, r) + D_2(r, q) = D(p, r) + D(r, q)$ puisque D_1 et D_2 satisfont l'axiome de l'inégalité triangulaire.

Puisque la distance de Hamming est une distance séparable, c'est-à-dire une somme de d distances élémentaires de Hamming 1D, il suffit simplement de vérifier que la distance de Hamming 1D est une métrique satisfaisant l'inégalité triangulaire (puisque la somme de deux distances métriques est une distance métrique). Soient 3 bits $(p, q, r) \in \{0, 1\}^2$, montrons que $D_H(p, q) + D_H(q, r) \geq D_H(p, r) \forall p, q, r$. Puisque $D_H(p, q) = D_H(q, p)$, on peut considérer $p \leq q$ et on vérifie manuellement les six cas suivants :

p	q	r	$D_H(p, q) + D_H(q, r)$	$D_H(p, r)$
0	0	0	0	0
0	1	0	2	0
1	1	0	1	1
0	0	1	1	1
0	1	1	1	1
1	1	1	0	0

IV-A-2. Soit c un minimiseur de

$$l(x; X) = \frac{1}{n} \sum_{i=1}^n D_H(x, x_i).$$

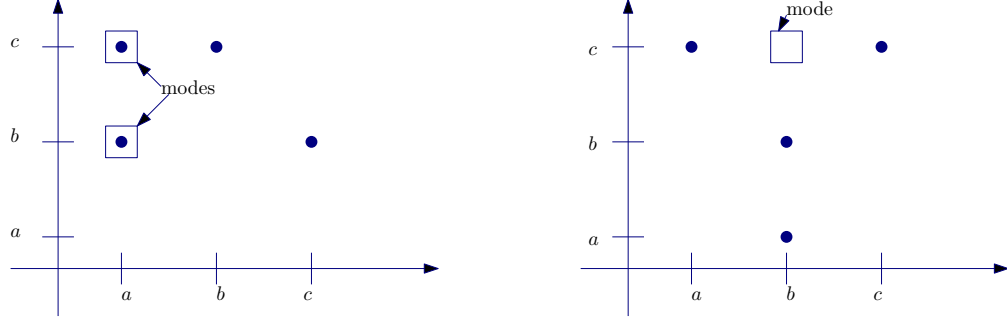
On cherche à caractériser $c = \arg \min_{x \in E_1 \times \dots \times E_d} l(x; X)$.

On a :

$$\begin{aligned} \sum_{i=1}^n D_H(x, x_i) &= \sum_{i=1}^n \sum_{j=1}^d D_H(x^{(j)}, x_i^{(j)}), \\ &= \sum_{j=1}^d \underbrace{\sum_{i=1}^n D_H(x^{(j)}, x_i^{(j)})}_{=n_j(x)}. \end{aligned}$$

Puisque $n_j(x) \leq n_j^*$ avec égalité quand $x^{(j)} \in T_j$, en notant $f_j^* = \frac{n_j^*}{n}$, on en déduit qu'un mode de X doit avoir *pour chaque attribut j* une étiquette de *fréquence maximale* f_j^* de X . L'algorithme consiste donc à calculer pour chaque dimension j , les étiquettes de fréquence maximale T_j , et à choisir $c^{(j)}$ dans T_j , en temps linéaire. On peut aussi calculer combien il y a de modes maximaux : $\prod_{j=1}^d |T_j|$.

Le mode de $X = \{(a, b), (a, c), (c, b), (b, c)\}$ n'est pas forcément unique puisque nous avons deux minimiseurs (a, b) et (a, c) (voir la figure ci-dessous, à gauche). Pour $X = \{(a, c), (b, a), (b, b), (c, c)\}$, le mode (b, c) est unique, mais n'appartient pas à X (voir la figure ci-dessous, à droite).



IV-A-3. On peut ainsi généraliser l'algorithme de Lloyd des k -moyennes à la distance de Hamming. La fonction de coût à minimiser pour k clusters est :

$$l(X; C) = \frac{1}{n} \sum_{i=1}^n \min_{j \in [k]} D_H(c_j, x_i) \geq 0$$

où c_j sont les k centres de Hamming, les k modes. On initialise les k modes en tirant des étiquettes aléatoires pour chaque attribut. A chaque cycle, on associe les données à leur plus proche mode, puis pour chaque groupe de données ayant le même mode, on calcule le nouveau mode, et on termine ces itérations lorsque la fonction de coût ne décroît plus strictement.

Puisque $\forall x, x', D_H(x, x') \leq \sum_{j=1}^d m_j = M$, on a initialement $nl(X; C) \leq nM$, et on assure à chaque itération de décroître d'au moins une unité $nl(X; C)$ avec $l(X; C) \geq 0$. On a donc au plus nM itérations avant la convergence.

Voir [2] (1998) pour le papier introduisant les k -modes.

IV-B-1. Le centre de masse pour des poids positifs non-normalisés est :

$$c = \sum_{i=1}^n \frac{w_i}{W} x_i, \quad W = \sum_{i=1}^n w_i$$

Preuve : Soit on normalize les poids avec $w'_i = \frac{w_i}{W}$, et on en déduit ainsi immédiatement la formule, ou soit on minimise $\sum_{i=1}^n w_i \|x_i - c\|^2$, calcule le gradient et on l'annule en vérifiant que le Hessien est bien positif-défini (ce qui assure que la solution est unique).

Il en résulte que :

$$c_j = \frac{\sum_{i=1}^n w_{i,j}^p x_i}{\sum_{i=1}^n w_{i,j}^p}$$

IV-B-2. Pour les données X et les centres C fixés, notons $L(w, \lambda) = J_p(X; C; w) + \sum_{i=1}^n \lambda_i (1 - \sum_{j=1}^k w_{i,j})$ la fonction Lagrangienne qui prend en compte les n contraintes de poids à satisfaire : $\sum_{j=1}^k w_{i,j} = 1$.

Les dérivées partielles du Lagrangien pour les nk poids $w_{i,j}$ sont :

$$\frac{\partial L(w, \lambda)}{\partial w_{i,j}} = pw_{i,j}^{p-1} d_{i,j}^2 - \lambda_i, \quad \forall i \in [n] \forall j \in [k],$$

et elles s'annulent pour les valeurs des poids :

$$w_{i,j} = \left(\frac{\lambda_i}{pd_{i,j}^2} \right)^{\frac{1}{p-1}},$$

Puisque nous avons $\sum_{j=1}^k w_{i,j} = 1$, on en déduit que :

$$\sum_{j=1}^k \left(\frac{\lambda_i}{pd_{i,j}^2} \right)^{\frac{1}{p-1}} = 1.$$

On trouve donc :

$$\lambda_i = \left(\sum_{j=1}^k (pd_{i,j}^2)^{-\frac{1}{1-p}} \right)^{1-p},$$

et finalement on trouve :

$$w_{i,j} = \frac{d_{i,j}^{-\frac{2}{1-p}}}{\sum_{l=1}^k d_{i,l}^{-\frac{2}{1-p}}} = \frac{\|x_i - c_j\|^{-\frac{2}{1-p}}}{\sum_{l=1}^k \|x_i - c_l\|^{-\frac{2}{1-p}}}$$

IV-B-3. IV-B-3-a. A chaque itération, calculer les k centroïdes font nécessairement diminuer la fonction de coût : $J_p(X; C^{(t)}; w^{(t)}) \leq J_p(X; C^{(t+1)}; w^{(t)})$. puis la mise à jour des poids que l'on résout comme un problème de minimisation font également diminuer la fonction de coût : $J_p(X; C^{(t+1)}; w^{(t)}) \leq J_p(X; C^{(t+1)}; w^{(t+1)})$ et donc par transitivité, on a : $J_p(X; C^{(t)}; w^{(t)}) \leq J_p(X; C^{(t+1)}; w^{(t+1)})$. Puisque $J_p(X; C^{(t)}; w^{(t)}) \geq 0$, l'algorithme des k -moyennes floues converge monotonement vers un minimum local.

IV-B-3-b. Quand $p \rightarrow \infty$, puisque $\frac{1}{d_{i,j}^{-\frac{2}{p-1}}} \rightarrow 1$ (puisque $x^{\frac{1}{p}} \rightarrow 1$), on a $w_{i,j} \rightarrow \frac{1}{k}$.

IV-B-3-c. Quand $p \rightarrow 1^+$, $w_{i,j} = \frac{(d'_{i,j})^q}{\sum_{l=1}^k (d'_{i,l})^q}$ avec $d'_{i,j} = \frac{1}{d_{i,j}}$ et $q \rightarrow 0$. Donc $w_{i,j} \rightarrow \frac{(d'_{i,j})^q}{\max_{l \in [k]} (d'_{i,l})^q}$. Ainsi $w_{i,j^*} \rightarrow 1$ pour $j^* = \arg \max_{j \in [k]} d'_{i,j}$ et $w_{i,j'} \rightarrow 0$ pour $j' \neq j^*$ puisque $\sum_{l=1}^k kw_{i,l} = 1$. Puisque $j^* = \arg \max_{j \in [k]} d'_{i,j} = \arg \min_{j \in [k]} d_{i,j}$, on en déduit que le point x_i est associé au centre le plus proche. Ainsi on a prouvé que les k -moyennes floues tendent vers les k -moyennes quand $p \rightarrow 1$ (avec $w_{i,j} \in \{0, 1\}$).

IV-B-4. La figure 2 donne le pseudo-code pour les k -moyennes floues en MPI.

La grande différence avec l'algorithme des k -moyennes parallèle en MPI est qu'on ne diffuse pas (en MPI, une opération de *broadcasting*, **Bcast**) à chaque itération les centroïdes à tous les processus.

Voir le papier [4] (2002) pour une implémentation détaillée en MPI.

Algorithm 1: Parallel Fuzzy c-Means (PFCM)

```
1:  P = MPI_Comm_size();
2:  myid = MPI_Comm_rank();
3:  randomise my_uOld[j][i] for each x[i] in fuzzy cluster j;
4:  do {
5:      myLargestErr = 0;
6:      for j = 1 to c
7:          myUsum[j] = 0;
8:          reset vectors my_v[j] to 0;
9:          reset my_u[j][i] to 0;
10:     endfor;
11:     for i = myid * (n / P) + 1 to (myid + 1) * (n / P)
12:         for j = 1 to c
13:             update myUsum[j];
14:             update vectors my_v[j];
15:         endfor;
16:     endfor;
17:     for j = 1 to c
18:         MPI_Allreduce(myUsum[j], Usum[j], MPI_SUM);
19:         MPI_Allreduce(my_v[j], v[j], MPI_SUM);
20:         update centroid vectors:
           v[j] = v[j] / Usum[j];
21:     endfor;
22:     for i = myid * (n / P) + 1 to (myid + 1) * (n / P)
23:         for j = 1 to c
24:             update my_u[j][i];
25:             myLargestErr = max{|my_u[j][i] - my_uOld[j][i]|};
26:             my_uOld[j][i] = my_u[j][i];
27:         endfor;
28:     endfor;
29:     MPI_Allreduce(myLargestErr, Err, MPI_MAX);
30: } while (Err >= epsilon)
```

FIGURE 2 – Les k -moyennes floues en MPI.

IV-C-1. Le maximum de vraisemblance est $\hat{\mu} = \arg \max_{\mu \in \mathbb{R}^d} \prod_{i=1}^n p(x_i; \mu)$. Puisque le logarithme est une fonction monotone croissante, la maximisation de la vraisemblance est équivalente à la maximisation de sa log-vraisemblance :

$$\hat{\mu} = \arg \max_{\mu \in \mathbb{R}^d} \sum_{i=1}^n l(x_i; \mu)$$

On a

$$l(x; \mu) = -\frac{1}{2\sigma^2} \|x - \mu\|^2 - d \log \sigma - \frac{d}{2} \log(2\pi)$$

Puisque $\sigma > 0$ et d sont constants, $l(x; \mu) \propto \|x - \mu\|^2$, et ce problème de maximisation de vraisemblance est équivalent à :

$$\hat{\mu} = \arg \max_{\mu \in \mathbb{R}^d} - \sum_{i=1}^n \|x_i - \mu\|^2.$$

C'est donc équivalent à $\arg \min_{\mu \in \mathbb{R}^d} \sum_{i=1}^n \|x_i - \mu\|^2$. On reconnaît la fonction à minimiser pour le centroïde, et on en déduit que :

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i.$$

IV-C-2. Quand $\sigma \rightarrow 0$, on a :

$$l_{j,i} = \frac{\exp(-\frac{1}{2\sigma^2} \|x_i - \mu_j\|^2)}{\sum_{l=1}^k \exp(-\frac{1}{2\sigma^2} \|x_i - \mu_l\|^2)}$$

Soit $j^* = \arg \max_l -\frac{1}{2\sigma^2} \|x_i - \mu_l\|^2$, alors quand $\sigma \rightarrow 0$, on a

$$l_{j,i} \rightarrow \frac{\exp(-\frac{1}{2\sigma^2} \|x_i - \mu_j\|^2)}{\exp(-\frac{1}{2\sigma^2} \|x_i - \mu_{j^*}\|^2)} \rightarrow \begin{cases} 1 & j = j^* \\ 0 & j \neq j^* \end{cases}$$

L'algorithme de Lloyd pour les k -moyennes est garanti de converger vers un minimum local en un temps fini. Puisqu'on ne répète jamais deux fois les mêmes partitions, le nombre d'itérations est borné par le deuxième nombre de Stirling $S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$. L'algorithme EM bien que convergeant monotonement ne finit pas en temps fini. Le nombre d'itérations est donc infini quand $\epsilon = 0$, et c'est pour cela qu'on a **besoin** d'implémenter un critère $\epsilon > 0$ pour stopper les itérations.

IV-C-3. Voici le pseudo-code de EM en MPI :

Plus de détails dans le papier [3].

Références

- [1] Lih-Hsing Hsu, Rong-Hong Jan, Yu-Che Lee, Chun-Nan Hung, and Maw-Sheng Chern. Finding the most vital edge with respect to minimum spanning tree in weighted graphs. *Information Processing Letters*, 39(5) :277–281, 1991.
- [2] Zhexue Huang. Extensions to the k -means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery*, 2(3) :283–304, 1998.
- [3] Ayush Kapoor, Harsh Hemani, N. Sakthivel, and S. Chaturvedi. *Emerging ICT for Bridging the Future - Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*, chapter MPI Implementation of Expectation Maximization Algorithm for Gaussian Mixture Models, pages 517–523. Springer International Publishing, Cham, 2015.
- [4] Terence Kwok, Kate Smith, Sebastián Lozano, and David Taniar. Parallel fuzzy c -means clustering for large data sets. In *Euro-Par Parallel Processing*, pages 365–374. Springer, 2002.