K-Spectral Centroïd pour des données massives

Brieuc Conan-Guez, Alain Gély, Lydia Boudjeloud-Assala, Alexandre Blansché

Université de Lorraine - Site de Metz - île du Saulcy 57045 METZ CEDEX 1
brieuc.conan-guez@univ-lorraine.fr, alain.gely@univ-lorraine.fr,
lydia.boudjeloud-assala@univ-lorraine.fr, alexandre.blansche@univ-lorraine.fr
http://www.lita.univ-lorraine.fr/

Résumé. Nous nous intéressons à la classification non supervisée de séries chronologiques. Pour ce faire, nous utilisons l'algorithme K-Spectral Centroïd (K-SC), une variante des K-Means. K-Spectral Centroïd utilise une mesure de dissimilarité entre séries chronologiques, invariante par translation et par changement d'échelle. Cet algorithme est coûteux en temps de calcul : lors de la phase d'affectation, il nécessite de tester toutes les translations possibles pour identifier la meilleure ; lors de la phase de représentation, le calcul du nouveau barycentre nécessite l'extraction de la plus petite valeur propre d'une matrice. Nous proposons dans ce travail trois optimisations de K-SC. L'identification de la meilleure translation peut être réalisée efficacement en utilisant la transformée de Fourier discrète. Chaque matrice peut être calculée incrémentalement. Le calcul du nouveau barycentre peut s'effectuer à moindre coût grâce à la méthode de la puissance itérée. Ces trois optimisations fournissent exactement la même classification que K-SC.

1 Introduction

Dans ce travail, nous nous intéressons à la classification non supervisée de séries chronologiques issues de l'analyse de media-sociaux, pour faire émerger différents types de partages de l'information. Ce travail s'inscrit dans le cadre de l'ANR INFO-RSN dont l'objet principal est l'étude de la diffusion de l'information sur les réseaux socionumériques (Twitter).

Pendant 6 mois, les tweets citant l'URL d'un article de presse issu d'une liste prédéfinie de 32 médias ont été collectés. Plusieurs types de séries temporelles peuvent être définis. On peut par exemple associer une série temporelle à chacune des URL collectées, ou encore à chacun des hashtags apparaissant dans la collecte. La série temporelle est alors l'ensemble des informations d'horodatage des tweets citant un article, ou comportant un hashtag.

L'étude de séries chronologiques présente deux problématiques. L'une est sémantique et concerne le choix de la dissimilarité pour comparer les séries. L'autre, plus technique, concerne le temps de calcul pour la classification de ces séries.

D'un point de vue sémantique, la dissimilarité choisie doit permettre de mettre en évidence des comportements similaires, bien qu'à des échelles différentes. En effet, le volume de tweets engendré par un quotidien national est bien évidemment très différent de celui d'un journal de presse locale. D'autre part il faut permettre une certaine lattitude de façon à distinguer les

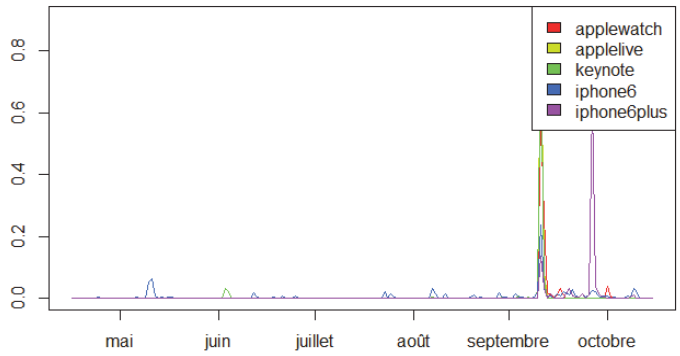

FIG. 1 – *Hashtags de comportements similaires tout au long de l'étude*

formes de partages indépendamment du moment de ces derniers. On voudrait par exemple que les différentes séries chronologiques de la figure 1 se retrouvent dans la même classe. En effet, elles ont le même comportement mais à des dates différentes. Ainsi, le hashtag « iphone6plus » (fin septembre) a un comportement similaire à « applewatch », « applelive », « keynote », « iphone6 » mais n'est pas utilisé au même moment (début septembre).

Dans un premier temps, nous avions étudié l'impact d'une échelle logarithmique sur la classification des séries (Charpentier et Flachaire (2014); Conan-Guez et al. (2016)) mais celle-ci a une trop grande tendance à minimiser le poid de la fin de série. Nous avons donc besoin d'une dissimilarité qui soit invariante au changement d'échelle et à la translation. Il existe de nombreux travaux sur la comparaison de séries temporelles, et on pourra par exemple se référer à Aghabozorgi et al. (2015) pour une étude comparative. Une approche classique se base sur la distance DTW (Dynamic Time Warping) et l'adapte à des données de grande dimension (voir par exemple Keogh et Pazzani (2000); Keogh et Ratanamahatana (2005)). Cependant, cette distance ne préserve pas forcément la forme des pics d'activités, ces derniers pouvant être plus ou moins étirés ou compressés sur l'axe des temps par DTW.

D'autres travaux proposent une dissimilarité qui, bien qu'indépendante de la translation dans le temps et des effets d'échelles, conserve l'aspect général des périodes d'activités. L'algorithme K-Spectral Centroïd proposé dans Yang et Leskovec (2011) est l'une de ces méthodes. Basé sur l'algorithme K-means, il souffre de deux écueils. Lors de la phase d'affectation, il nécessite de tester toutes les translations possibles entre séries temporelles pour identifier la meilleure. Lors de la phase de représentation, le calcul d'un nouveau barycentre nécessite tout d'abord la construction d'une matrice, puis l'extraction de la plus petite valeur propre de cette même matrice. Ces deux étapes se révèlent coûteuses en temps de calcul. Pour pallier ce problème, les auteurs proposent une approche multi-niveaux : l'algorithme est appliqué sur une succession de séries représentant la même information, mais à des résolutions différentes. Ici, nous présentons trois améliorations de l'algorithme K-SC de base. Ce dernier

étant utilisé à l'identique dans l'approche multi-niveaux, cette dernière bénéficie automatiquement des améliorations proposées dans ce travail. Aussi nous nous comparons à l'algorithme de base pour présenter nos trois optimisations de K-Spectral Centroïd :

La première amélioration porte sur la phase d'affectation, l'identification de la meilleure translation peut être réalisée efficacement dans l'espace fréquentiel en utilisant les transformées de Fourier. Cette approche permet de réduire la complexité algorithmique de la recherche. Les deux améliorations suivantes portent sur la phase de représentation : la première permet d'accélérer le calcul des matrices nécessaires à l'obtention des barycentres grâce à une approche incrémentale, la seconde ramène l'extraction de la plus petite valeur propre à un problème équivalent d'extraction de la plus grande valeur propre. Ce dernier problème est classique en algèbre linéaire et possède des algorithmes de résolution aux performances éprouvées (par exemple par la méthode de la puissance itérée). Ces trois optimisations fournissent exactement le même résultat (classification) que celui de K-SC mais en réduisant fortement les temps de calcul.

Dans la suite de l'article, nous présentons l'algorithme K-Spectral Centroïd (section 2), puis les trois optimisations que nous proposons pour réduire les temps de calculs (section 3). Les résultats expérimentaux sont exposés dans la section 4 et nous concluons dans la section 5.

2 L'algorithme K-Spectral Centroïd

2.1 Dissimilarité invariante par décalage et changement d'échelle

On considère ici des séries chronologiques à valeurs positives indicées par un intervalle de $\mathcal{Z}$. On note $\dot{x}$ l'indice le plus petit de la série x et $\ddot{x}$ l'indice le plus grand. On a donc $x = (x_{\dot{x}}, x_{\dot{x}+1}, x_{\dot{x}+2}, \ldots, x_{\ddot{x}})$. Le nombre de valeurs de x est noté $L(x)$ et vaut $\ddot{x} - \dot{x} + 1$.

La norme de x se définit classiquement par $\|x\| = \sqrt{x \cdot x}$ grâce au produit scalaire :

$$x \cdot y = \sum_{i=\max(\dot{x},\dot{y})}^{\min(\ddot{x},\ddot{y})} x_i \, y_i$$

On considère l'opérateur de décalage temporel τ_o de paramètre o. Ce paramètre appartient à un sous-intervalle de $\mathcal{Z}$, noté $\mathcal{O}$. La série décalée $z = \tau_o(y)$ a pour indice $\dot{z} = \dot{y}+o$, $\ddot{z} = \ddot{y}+o$ et pour valeurs $z_i = y_{i-o}$. Le produit scalaire avec un décalage s'écrit :

$$x \cdot \tau_o(y) = \sum_{i=\max(\dot{x},\dot{y}+o)}^{\min(\ddot{x},\ddot{y}+o)} x_i \, y_{i-o} = y \cdot \tau_{-o}(x)$$

Comme indiqué dans l'introduction, la dissimilarité utilisée dans l'algorithme K-SC est invariante par décalage temporel et par changement d'échelle. Elle se concentre donc sur la forme des séries chronologiques. La minimisation d'un paramètre α permet de neutraliser les effets d'échelle entre deux séries qui auraient le même comportement mais sur des ordres de grandeur différents. Voici l'expression de la dissimilarité :

$$d(x,y) = \min_{o \in \mathcal{O}, \alpha \in \mathbb{R}^+} \frac{\|x - \alpha \, \tau_o(y)\|}{\|x\|}$$

On pose $g(x,y) = \min_{\alpha \in \mathbb{R}^+} \frac{\|x-\alpha y\|}{\|x\|}$. Ce qui permet d'obtenir l'expression suivante : $d(x,y) = \min_{o \in \mathcal{O}} g(x, \tau_o(y))$. La valeur α permettant le calcul de g est obtenue par annulation de la dérivée : $\alpha = \frac{x \cdot y}{\|y\|^2}$. En remplaçant α par cette expression dans g, on obtient :

$$g(x,y) = \sqrt{1 - \frac{(x \cdot y)^2}{\|x\|^2 \|y\|^2}}$$

Comme dans la pratique, les normes de x et y peuvent être précalculées, le calcul de g ne nécessite plus que l'évaluation d'un unique produit scalaire. Ce résultat apparait dans la version longue du papier Yang et Leskovec (2011), lors de la démonstration de la propriété de symétrie de d, qui est évidente de par l'expression de g obtenue.

2.2 K-Spectral Centroïd

L'algorithme K-Spectral Centroïd (K-SC) est une variante des K-Means, qui utilise la dissimilarité d définie dans la section précédente. Cet algorithme optimise le critère d'inertie suivant :

$$F = \sum_{k=1}^{K} \sum_{x_i \in C_k} d^2(\mu_k, x_i)$$

où K est le nombre total de classes et C_k est la classe numéro k, μ_k le barycentre de C_k et x_i les séries chronologiques à traiter.

De manière identique à K-Means, K-SC exécute itérativement deux étapes : affectation (production d'une nouvelle partition), et représentation (calcul des nouveaux barycentres).

La phase d'affectation utilise la dissimilarité d. La complexité de l'étape d'affectation est $O(nKt^2)$, où n est le nombre de séries chronologiques, K est le nombre de classes, t est la longueur moyenne des séries chronologiques. En effet l'évaluation de d a pour complexité $O(t^2)$ (on considère ici que $|\mathcal{O}| = O(t)$).

La phase de représentation calcule le barycentre μ_k de la classe C_k en minimisant l'inertie suivante :

$$\mu_k = \arg\min_{\mu} \sum_{x_i \in C_k} d^2(\mu, x_i) = \arg\min_{\mu} \frac{1}{\|\mu\|^2} \mu^t \sum_{x_i \in C_k} \left(Id - \frac{x_i \, x_i^t}{\|x_i\|^2} \right) \mu$$

Dans Yang et Leskovec (2011), les auteurs indiquent que ce problème de minimisation est équivalent à l'extraction de la plus petite valeur propre de la matrice "somme" apparaissant dans l'expression de droite. Si l'on effectue une diagonalisation complète de la matrice pour extraire cette valeur propre, la complexité algorithmique est $O(nt^2 + Kt^3)$, avec le terme nt^2 pour le calcul des matrices "somme" et le terme Kt^3 pour l'extraction de toutes les valeurs propres de toutes les matrices. Si on considère que $n \gg t$, la complexité est donnée par $O(nt^2)$. On voit bien ici que le coût de calcul des matrices est supérieur à celui de leur diagonalisation.

3 Améliorations proposées

3.1 Affectation : calcul efficace grâce à la transformée de Fourier

Le calcul de la dissimilarité d nécessite de calculer $g(x, \tau_o(y))$ pour toute les valeurs de $o \in \mathcal{O}$ afin de trouver o minimisant $g(x, \tau_o(y))$. Nous proposons ici d'accélérer le calcul de d en menant le calcul dans l'espace fréquentiel grâce aux transformées de Fourier.

On remarque tout d'abord que minimiser $g(x, \tau_o(y))$ en o revient à maximiser le produit scalaire $S(o) = x \cdot \tau_o(y)$ en o. On note x^0 le vecteur contenant les mêmes valeurs que x mais complété par des zéros à droite. $x^0 = (x_{\dot{x}}, \ldots, x_{\ddot{x}}, 0, \ldots, 0)$. La longueur de x^0 est égale à $L(x^0) = L(x) + L(y) - 1$. On définit de même ${}^0y$, le vecteur complété par des zéros à gauche et de même longueur $L({}^0y) = L(x^0)$. On a ${}^0y = (0, \ldots, 0, y_{\dot{y}}, \ldots, y_{\ddot{y}})$.

On note $\mathcal{F}$ l'opérateur qui calcule la transformée de Fourier discrète d'un vecteur. La transformation inverse est notée $\mathcal{F}^{-1}$. En remarquant que la fonction S est une fonction de corrélation croisée, et que la transformée de Fourier d'une corrélation croisée s'exprime sous la forme d'un simple produit entre spectres dans l'espace fréquentiel (Kunt (1986)), les valeurs de $S(o)$ pour $\dot{x} - \ddot{y} \leq o \leq \ddot{x} - \dot{y}$ sont données par la transformation suivante :

$$S \equiv \mathcal{F}^{-1}\left(\mathcal{F}(x^0)\ \mathcal{F}^*({}^0y)\right)$$

où z^* est l'application de conjugaison complexe, qui a $z = a + ib$ associe $z^* = a - ib$.

L'évaluation de toutes les valeurs de $S(o)$ nécessite donc trois transformées de Fourier, dont une transformée inverse et un produit de deux vecteurs complexes dans l'espace fréquentiel (produit du spectre de x^0 et du conjugué du spectre de ${}^0y$). La complexité algorithmique du calcul de la transformée de Fourier rapide (Kunt (1986)) est de $O(t \ln(t))$, où t est la longueur moyenne des séries. La complexité algorithmique de la phase d'affectation est donc à présent de $O(nKt \ln(t))$ au lieu de $O(nKt^2)$ pour un calcul complet de tous les produits scalaires.

Dans notre implémentation, les transformées de Fourier des séries du jeu de données ainsi que celles des barycentres sont pré-calculées. Grâce à ces pré-calculs, l'évaluation de la dissimilarité d ne nécessite seulement que le calcul de la transformée inverse $\mathcal{F}^{-1}$, les deux autres transformées ne sont plus nécessaires. Cette optimisation ne change pas la complexité théorique, mais permet de gagner au moins un facteur 2 sur le temps de calcul de la phase d'affectation.

3.2 Représentation : Calcul des barycentres

3.2.1 Calcul des valeurs propres par la méthode de la puissance itérée

Comme indiqué dans la section précédente, le calcul d'un barycentre nécessite l'extraction de la plus petite valeur propre λ_{min} d'une matrice. Ce calcul peut se réaliser soit par une diagonalisation complète de la matrice, soit en utilisant un algorithme qui extrait juste λ_{min} : la méthode de la puissance inverse. Bien que ne nécessitant pas le calcul de l'inverse de la matrice, cette approche est coûteuse car elle impose la résolution d'un système à chaque itération.

On peut cependant remarquer que le problème de l'extraction de λ_{min} peut se ramener simplement à l'extraction de la plus grande valeur propre λ_{max} de la matrice $M_k = \sum_{x_i \in C_k} \frac{x_i\ x_i^t}{\|x_i\|^2}$. Plus précisément :

$$\begin{aligned}\mu_k &= \arg\min_{\mu} \frac{\mu^t \sum_{x_i \in C_k} \left(Id - \frac{x_i\ x_i^t}{\|x_i\|^2}\right) \mu}{\|\mu\|^2} \\ &= \arg\max_{\mu} \frac{\mu^t \left(\sum_{x_i \in C_k} \frac{x_i\ x_i^t}{\|x_i\|^2}\right) \mu}{\|\mu\|^2} \\ &= \arg\max_{\mu} \frac{\mu^t M_k \mu}{\|\mu\|^2}\end{aligned}$$

$R_{M_k}(\mu) = \frac{\mu^t M_k \mu}{\|\mu\|^2}$ est le quotient de Rayleigh associé à la matrice M_k. Cette fonction a la propriété suivante : $\max_\mu R_{M_k}(\mu) = \lambda_{max}$ et $\min_\mu R_{M_k}(\mu) = \lambda_{min}$. Maximiser $R_{M_k}(\mu)$ revient à extraire le vecteur propre associé à la plus grande valeur propre $\lambda_{\max}$ de M_k. Ceci peut être réalisé efficacement par la méthode de la puissance itérée (Björck (2015)).

On peut remarquer que pour des x_i à valeurs positives, le barycentre peut être choisi à valeurs positives. En effet, si μ contient des valeurs négatives, le vecteur des valeurs absolues, noté μ^+ mènera nécessairement à une valeur $R_{M_k}(\mu^+) \geq R_{M_k}(\mu)$. Dans la pratique, le vecteur propre μ_k a donc toutes ses valeurs de même signe. Dans le cas négatif, il suffit de prendre l'opposé du vecteur.

Le terme $\frac{x_i\ x_i^t}{\|x_i\|^2}\mu$ peut se réécrire de la manière suivante $(\mu \cdot \frac{x_i}{\|x_i\|})\frac{x_i}{\|x_i\|}$. Ce terme correspond à la projection du vecteur μ sur le vecteur unitaire $\frac{x_i}{\|x_i\|}$. Cet opérateur de projection $P_i = \frac{x_i\ x_i^t}{\|x_i\|^2}$ a donc sa plus grande valeur propre égale à 1 et les suivantes sont nulles. On a donc $\lambda_{max} = \max_\mu R_{M_k}(\mu) \leq \sum_{x_i \in C_k} \max_\mu R_{P_i}(\mu) = |C_k|$, où $|C_k|$ est le cardinal de la classe C_k. Avec le même argument pour λ_{min}, on constate que les valeurs propres de M_k sont comprises dans l'intervalle $[0, |C_k|]$. Toutes les valeurs propres sont donc positives, et surtout les deux plus grandes valeurs propres de M_k sont bien distinctes, ce qui favorise la convergence de la méthode de la puissance itérée. Cette méthode ne nécessite à chaque itération que la multiplication de la matrice M_k par un vecteur, et le nombre d'itérations est relativement réduit. Le calcul exact des barycentres peut donc être mené efficacement.

3.2.2 Calcul des matrices par une approche incrémentale

On peut remarquer que le calcul des K matrices $M_k = \sum_{x_i \in C_k} \frac{x_i\ x_i^t}{\|x_i\|^2}$ peut être mené de manière incrémentale. En effet, une matrice M_k est la somme des contributions de chaque individu de la classe : une contribution est la matrice $P_i = \frac{x_i\ x_i^t}{\|x_i\|^2}$ de rang 1. On peut remarquer que les matrices P_i n'ont pas les mêmes dimensions, car les séries ne sont pas de mêmes longueurs. La sommation s'effectue donc en ajoutant à la bonne position la sous-matrice P_i à la matrice M_k. Le calcul complet de toutes les matrices nécessite donc n sommes, une par individu.

Si au cours d'une itération, un individu change de classes, on peut mettre à jour la matrice de la classe de départ et la matrice de la classe d'arrivée de la manière suivante : on retranche la contribution de l'individu à la matrice de départ, et on ajoute cette contribution à la matrice d'arrivée. Dans le cas spécifique de K-SC, un autre cas nécessite d'effectuer cette double opération : si un individu reste dans la même classe, mais que la meilleure translation n'est plus

la même. On doit alors dans la même matrice, retrancher l'ancienne contribution et ajouter la contribution translatée à la matrice.

Le calcul incrémental ne se révèle efficace que dans le cas où le nombre de modifications (changement de classes, ou translation différente) reste faible. Si l'on note p le nombre de modifications, le calcul incrémental nécessite $2p$ sommes de matrices (soustraction puis addition p fois), alors que le calcul complet nécessite n sommes. Si $2p < n$, on peut espèrer que le calcul incrémental se révèle plus rapide que le calcul complet. On peut remarquer que ce critère qui détermine si l'approche incrémentale doit être utilisée est simple à mettre en oeuvre. Une faiblesse de ce critère tient au fait qu'il ne prend pas en compte la longueur des séries. C'est le critère utilisé dans ce travail.

Enfin, le calcul incrémental impose un coût mémoire beaucoup plus important. On peut proposer l'estimation suivante : les matrices sont symétriques, et donc seuls les triangulaires inférieures des matrice sont à mémoriser. La longueur d'une matrice peut dans le pire cas être égale au double de la longueur de la plus grande série (à cause des translations). Pour des séries de longueur maximum 5000 (taille maximum traitée dans ce travail), chaque matrice peut occuper jusqu'à 400Mo. Pour une dizaine de classes, le coût mémoire du calcul incrémental peut atteindre 4Go. Ce qui est important.

4 Résultats expérimentaux

4.1 Jeu de test

Notre jeu de test se base sur un millier (976) de séries temporelles correspondant aux hashtags cités plus de 1000 fois lors de l'étude. Une discrétisation à l'échelle de la journée (24h) est mise en place. Chaque valeur d'une série contient donc le nombre de tweets ayant fait référence à un hashtag pendant une journée entière. Après cette discrétisation, les séries ont en moyenne 155 points (la série la plus courte comptant 2 points - les tweets sont répartis sur 2 jours - et la série la plus longue 183).

De façon à faciliter les comparatifs en limitant les biais, nous utiliserons ce jeu d'essai comme données à partir desquelles nous en construirons d'autres, comptant plus de séries, ou bien des séries plus longues. Nous nous assurons ainsi avoir des séries similaires en forme pour l'ensemble de nos tests. Pour ce faire :

- Nous augmentons le nombre de séries en répliquant les séries du jeu de données initial. Nous obtenons ainsi un jeu de données 1, 3, 5, 7, 9 fois plus grand que le jeu de données initial.
- Nous augmentons la longueur des séries en concaténant 1, 2, 3, 4, 5, 10, 20, 30 fois chacune des séries précédentes.
- Nous bruitons chacune des séries obtenues : chaque point de la série est modifié d'une amplitude d'au plus 5% (tirage uniforme) de sa valeur initiale.

Nous obtenons ainsi un nouveau jeu de données dont nous contrôlons la taille et la longueur des séries. Puis nous appliquons comme dans Yang et Leskovec (2011) une régularisation à chaque série grâce à un noyau gaussien d'écart-type σ. En effet, les données Twitter sont naturellement bruitées, et nécessitent donc un lissage avant d'exécuter l'algorithme K-SC.

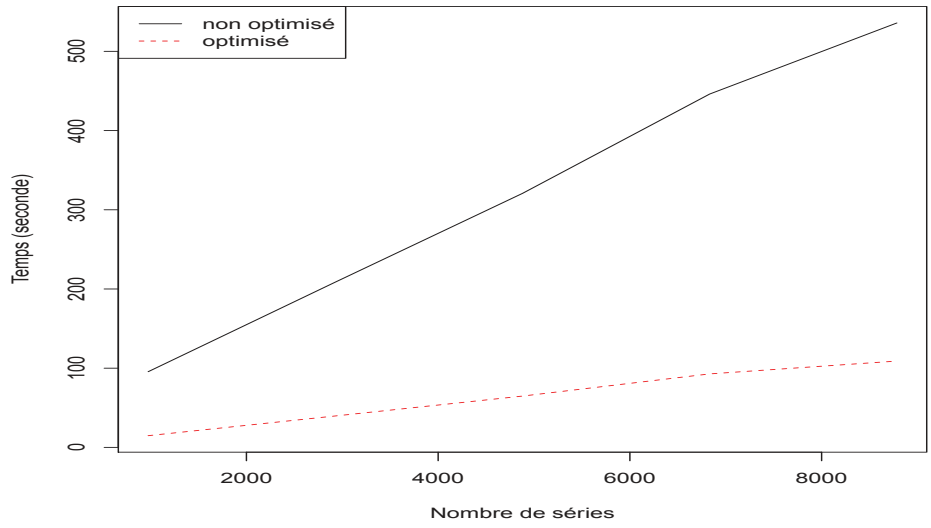

FIG. 2 – *Temps d'exécution (seconde) normalisé en fonction du nombre de séries*

4.2 Analyse des résultats

Nous cherchons à analyser l'impact des trois optimisations sur le temps de calcul.

Le paramètre de lissage gaussien est $\sigma = 1$. On rappelle que chaque point des séries correspond à une journée. Le choix $\sigma = 1$ a été validé visuellement. Le nombre de classes de la partition est fixe pour toutes les expériences et est fixé à 8. Le nombre maximum d'itérations de K-SC est fixé à 40. K-SC peut s'arrêter avant si le gain d'inertie normalisé prend une valeur inférieure à 1e-4. L'algorithme non optimisé et l'algorithme avec les trois optimisations sont exécutés 10 fois sur un même jeu de données avec 10 initialisations aléatoires. Le temps "machine" de l'ensemble de ces 10 exécutions est reporté en secondes dans la suite de ce travail.

Les expériences ont été menées sur un serveur contenant 40 coeurs d'exécution à 2.3GHz et 32 Go de mémoire vive. Tous les algorithmes ont été réimplémentés en Java. La diagonalisation des matrices utilise la bibliothèque EJML (Abeles (2016)), et le calcul des transformées de Fourier utilise la bibliothèque JTransforms (Wendykier (2015)).

4.2.1 Etude de l'effet de la taille du jeu de données

Dans cette première expérience, nous étudions l'effet de la taille du jeu de données sur le temps d'exécution de l'algorithme et de ses différentes optimisations. Les cinq jeux de données utilisés ont respectivement 976, 2 928, 4 880, 6 832 et 8 784 séries chronologiques. La longueur des séries est comprise entre 2 et 183, avec une moyenne à 155.

Comme indiqué précédemment, le critère d'arrêt de K-SC est double : le nombre maximum d'itérations est fixé à 40 et l'algorithme est stoppé dès lors que le gain d'inertie normalisé est

inférieur à 1e-4. De ce fait, le nombre d'itérations de l'algorithme varie d'une exécution à l'autre, et donc les temps de calcul sont dépendants de ce nombre d'itérations. Or dans ce travail, nous souhaitons analyser l'impact sur la durée de calcul des trois optimisations en supprimant l'effet lié au nombre d'itérations réalisé par l'algorithme. Pour résoudre ce problème, nous avons appliqué aux temps de calcul reportés dans ce travail une normalisation en considérant que chaque exécution réalise 20 itérations. La valeur de 20 correspondant à la moyenne du nombre d'itérations de toutes les expérimentations.

L'algorithme K-SC avec ou sans optimisations a une complexité linéaire en le nombre de séries chronologiques à traiter. Les expérimentations représentées sur la figure 2 sont cohérentes avec cette complexité théorique. Les complexités étant linéaires, le facteur d'accélération estimé est de 4.6.

nb séries	K-SC		affectation		calcul des M_k		barycentres	
	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$
976	14.8	6.5	13.1	4.0	0.6	2.8	1.0	41.7
2 928	39.7	5.3	37.4	4.2	1.4	3.4	0.9	51.8
4 880	64.8	4.9	61.7	4.2	2.1	4.0	0.9	57.0
6 832	92.8	4.8	88.8	4.3	2.8	4.1	0.9	60.7
8 784	109.0	4.9	104.5	4.5	3.4	4.4	0.9	56.9

TAB. 1 – *Temps d'exécution (seconde) normalisé de la version optimisée et facteur d'accélération par rapport à la version non optimisée*

La table 1 fournit les durées (normalisées) totales et intermédaires en secondes de l'algorithme optimisé et les facteurs d'accélération de ce dernier par rapport à la version non optimisée. Il est donc aisé de recalculer les durées associées à la version non optimisée. La phase de précalcul (transformées de Fourier des séries du jeu de données) n'est pas reportée, car elle est négligeable devant les autres quantités. On l'obtient facilement en retranchant au temps total la somme des temps intermédiaires (affectation, calcul des M_k et des barycentres). On constate que la phase d'affectation est la partie la plus coûteuse pour l'algorithme optimisé. Pour le jeu de données le plus grand, elle représente 96% du temps total. L'utilisation de la transformée de Fourier fournit un facteur d'accélération de 4.5 pour cette phase d'affectation.

Les temps de calcul des matrices M_k et l'extraction des plus grandes valeurs propres pour l'algorithme optimisé sont marginaux comparés à ceux de la phase d'affectation. Pour le plus grand jeu de données, le calcul des M_k représente 3.1% du temps total, et le calcul des barycentres 0.9%. Le facteur d'accélération pour la phase de calcul des matrices M_k croît avec le nombre de séries, passant de 2.8 à 4.4. Ceci s'explique par une plus grande efficacité du calcul incrémental quand le nombre de séries augmente. Enfin, on peut noter les facteurs d'accélération très importants obtenus grâce à l'utilisation de la méthode de la puissance itérée. Pour les plus grands jeux de données, cette accélération ne se retrouve pas dans le facteur d'accélération global. En effet, le calcul global est dominé par le nombre de séries n.

4.2.2 Etude de l'effet de la longueur des séries

Dans cette expérience, on souhaite étudier l'effet de la longueur des séries sur le temps d'exécution. La taille des jeux de données est fixée à 976. On considère ici huit jeux de données

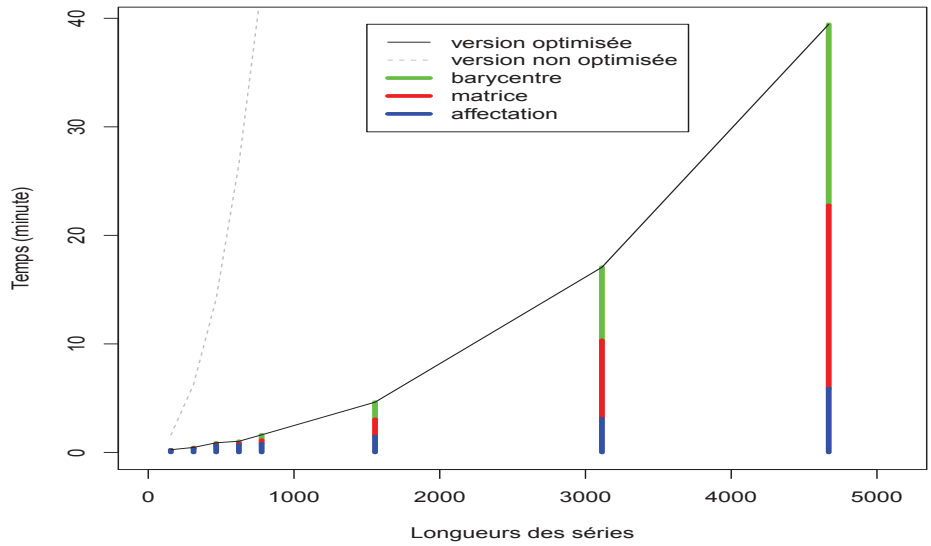

FIG. 3 – *Temps d'exécution (minute) normalisé en fonction de la longueur des séries. Les durées intermédiaires (affectation, calcul des matrices puis des barycentres) sont empilées.*

obtenus en appliquant les facteurs multiplicatifs suivants : 1, 2, 3, 4, 5, 10, 20 et 30. Le premier jeu de données (facteur multiplicatif unité) a des séries de longueur moyenne 155 (min=2 et max=183). Pour les jeux de données suivants, on a pour longueurs moyennes : 311, 466, 622, 778, 1 556, 3 113 et 4 669. Les longueurs maximales sont 366, 549, 732, 915, 1 830, 3 660 et 5 490.

La phase d'affectation a une complexité en $O(t\ln(t))$ (le nombre de séries et de barycentres étant ici intégré dans la notation O), où t est la longueur des séries. La phase de calcul des matrices M_k a une complexité quadratique en t. La phase de calcul des barycentres grâce à la méthode de la puissance itérée semble avoir elle aussi une complexité quadratique : le nombre d'itérations de la méthode est relativement constant et indépendant de la taille des matrices pour les jeux de données considérés. On rappelle que les algorithmes d'extraction de l'ensemble des valeurs propres d'une matrice ont une complexité cubique, ce qui explique les facteurs d'accélération importants liés à la méthode de la puissance itérée. En modélisant la relation entre le temps de calcul total et la longueur des séries t par un modèle non linéaire à deux paramètres $temps \sim \alpha t^{\beta}$. La valeur estimée pour le paramètre β vaut 1.99, soit une complexité quadratique en la longueur des séries.

La figure 3 et le tableau 2 donnent quelques éléments chiffrés sur le comportement pratique de l'algorithme. Du fait des temps observés plus importants pour ces expériences portant sur les longueurs des séries, les durées sont toutes exprimées en minutes. La figure 3 permet non seulement une comparaison de l'algorithme optimisé par rapport à la version non optimisée,

longueurs	K-SC		affectation		calcul des M_k		barycentres	
	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$	durée	$\frac{nonopti}{opti}$
155	0.25	6.5	0.22	4.0	0.01	2.8	0.02	41.7
311	0.46	13.7	0.36	8.4	0.04	2.6	0.05	61.9
466	0.89	15.9	0.67	9.5	0.10	2.7	0.12	60.2
622	1.00	25.8	0.70	15.6	0.16	2.8	0.18	84.9
778	1.62	26.6	0.81	20.2	0.32	2.0	0.49	52.9
1 556	4.63	51.2	1.48	37.0	1.57	1.6	1.57	114.4
3 113	17.06	154.4	3.11	98.3	7.22	3.3	6.72	342.6
4 669	39.45	211.4	5.89	112.9	16.87	2.9	16.69	457.0

TAB. 2 – *Temps d'exécution (minute) normalisé de la version optimisée et facteur d'accélération par rapport à la version non optimisée.*

mais permet aussi de visualiser la part de chacune des phases (affectation, calcul des matrices M_k, calcul des barycentres) dans le temps total. Les temps d'exécution de la version non optimisée pour des longueurs au delà de 1000 ne sont pas reportés sur la figure 3, car ils sont très grands devant ceux de la version optimisée. Ils peuvent être recalculés grâce aux facteurs d'accélération fournis dans le tableau 2.

Contrairement aux expériences portant sur le nombre de séries, la part de la phase de représentation (calcul des M_k et des barycentres) n'est pas négligeable par rapport à la phase d'affectation : elle domine maintenant cette dernière. On constate que le calcul des matrices M_k et le calcul des barycentres ont des temps proportionnels : pour ce jeu de données de 976 séries groupées en 8 classes, le coefficient est proche de 1. Enfin nos optimisations sont d'autant plus efficaces que la longueur des séries augmente. Pour des séries de 150 éléments, le facteur d'accélération est déjà de 6 ; pour des séries de presque $5\,000$ éléments, il est supérieur à 200.

5 Conclusions

Nous avons proposé un ensemble d'optimisations qui permet d'améliorer la complexité de l'algorithme tout en fournissant des résultats identiques. Les expériences suggèrent que d'une complexité cubique en la taille des séries, on obtient à présent une complexité quadratique. De plus, pour une dizaine de classes et à longueurs de séries égales, la version optimisée est presque 5 fois plus rapide.

Afin de conforter ces premiers résultats, il semble souhaitable de mener d'autres expériences sur de nouveaux jeux de données, notamment issus d'autres domaines que les réseaux socionumériques.

6 Remerciement

Ce travail est réalisé dans le cadre de l'ANR INFO-RSN.

Références

Abeles, P. (2016). Efficient java matrix library (ejml) - http ://ejml.org.

Aghabozorgi, S., A. S. Shirkhorshidi, et Y. W. Teh (2015). Time-series clustering - a decade review. *Inf. Syst. 53*(C), 16–38.

Björck, A. (2015). *Numerical Methods in Matrix Computations*. Springer International Publishing.

Charpentier, A. et E. Flachaire (2014). Log-transform kernel density estimation of income distribution. Work paper, Université du Québec à Montréal - CTAN (http://www.ctan.org/). <halshs-01115988>.

Conan-Guez, B., A. Gély, L. Boudjeloud, A. Blansche, D. Compagno, et A. Mercier (2016). Représentation de données temporelles par un modèle à échelle logarithmique. In *Proceedings 13ème atelier sur la Fouille de Données Complexes (FDC) Extraction et Gestion des Connaissances (EGC) Reims, Janvier*.

Keogh, E. J. et M. J. Pazzani (2000). Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, New York, NY, USA, pp. 285–289. ACM.

Keogh, E. J. et C. A. Ratanamahatana (2005). Exact indexing of dynamic time warping. *Knowledge and Information Systems 7*(3), 358–386.

Kunt, M. (1986). *Digital signal processing*. Artech House communication and electronic defense library. Artech House.

Wendykier, P. (2015). Jtransforms - https://sites.google.com/site/piotrwendykier/software.

Yang, J. et J. Leskovec (2011). Patterns of temporal variation in online media. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pp. 177–186. ACM.

Summary

We introduce a K-Spectral Centroïd algorithm, a variant of K-Means to cluster large time series. K-Spectral Centroïd uses a dissimilarity between time series, which is invariant regarding translation and scaling. This algorithm is relatively expensive in computation time. Indeed, during the assignment phase, it requires testing all possible translations to identify the best solution. During the representation phase, the calculation of the new barycenter requires the extraction of the smallest eigenvalue of a matrix. We propose in this work to improve these two points. We measure subsequently the impact of these improvements on various experiments.