REGLO : une nouvelle stratégie pour résumer un flux de séries temporelles

Alice Marascu*, Florent Masseglia* and Yves Lechevallier**

*INRIA Sophia-Antipolis, 2004 route des lucioles - BP 93 , 06902 Sophia-Antipolis
**INRIA Paris-Rocquencourt, Domaine de Voluceau , 78150 Rocquencourt
Prenom.Nom@inria.fr

Résumé. Les flux de séries temporelles sont aujourd'hui produits dans de nombreux domaines comme la finance (Zhu et Shasha (2002)), la surveillance de réseaux (Borgne et al. (2007); Airoldi et Faloutsos (2004)), la gestion de l'historique des usages fréquents (Giannella et al. (2003); Teng et al. (2003)), etc. Résumer de tels flux est devenu un domaine important qui permet de surveiller et d'enregistrer des informations fiables sur les séries observées. À ce jour, la majorité des algorithmes de ce domaine s'est concentrée sur des résumés séparés et indépendants (Giannella et al. (2003); Zhu et Shasha (2002); Chen et al. (2002)), en accordant à chaque série le même espace en mémoire. Toutefois, la gestion de cet espace mémoire est un sujet important pour les flux de données et une stratégie accordant la même quantité de mémoire à chaque série n'est pas forcément appropriée. Dans cet article, nous considérons que les séries doivent être en compétition vis à vis de l'espace mémoire, selon leur besoin de précision. Ainsi, nous proposons : (1) une stratégie de gestion de l'espace mémoire optimisée et (2) une nouvelle méthode de résumé des séries temporelles par approximation. Dans ce but, nous observons à la fois l'erreur globale et les erreurs locales. La répartition de la mémoire suit les étapes suivantes : (1) recherche de la séquence la mieux représentée et (2) recherche de la partie à compresser en minimisant l'erreur. Nos expérimentations sur des données réelles montrent l'efficacité et la pertinence de notre approche.

1 Introduction

Les résumés de séries temporelles sont l'objet de travaux de plus en plus importants en raison de leurs nombreuses applications mais aussi de leur complexité. Parmi les applications liées à ce domaine citons la recherche d'information (Ding et al. (2008); Palpanas et al. (2008)), la fouille (Chen et al. (2002); Papadimitriou et al. (2005); Lin et al. (2003)) ou encore la prévision (Cheng et Tan (2008); Borgne et al. (2007)) de données. La difficulté associée à la production de ces résumés est due aux très grands volumes de données (qui sont produites en continu et à grande vitesse). En effet, ces données arrivant sous forme de flux, le temps de traitement de chaque donnée doit être le plus faible possible. De plus, la taille d'un flux étant potentiellement illimitée, il est inévitable de traiter des problèmes liés à la mémoire nécessaire pour le résumer.

Un flux ne peut donc pas être stocké et doit être (au mieux) résumé. Il n'est alors pas toujours possible de répondre avec exactitude à des requêtes sur les données passées. Néanmoins, une réponse approximative très proche de la valeur demandée peut-être satisfaisante. Produire des résumés de séries temporelles, permettant de répondre à ce problème, est un sujet très étudié ces dernières années (Chang et Lee (2003); Chen et al. (2002); Cohen et Strauss (2003); Papadimitriou et al. (2005); Teng et al. (2003); Zhu et Shasha (2002)). Considérons M, la quantité de mémoire disponible. Imaginons que nous souhaitions résumer le comportement de n séries temporelles et considérons que nous disposons d'une mémoire organisée sous la forme d'une matrice à n lignes. Dans le cas d'indépendance entre les séries nous pouvons alors stocker en mémoire au maximum $t = M/n$ valeurs pour chaque série. Après t itérations du flux la mémoire est pleine et nous devons appliquer une méthode pour résumer les séries et quand t devient très grand ce résumé ne peut pas se faire sans introduire un degré d'approximation. Ce degré d'approximation se traduit alors par une erreur locale à chaque série et une erreur globale (cumul des erreurs locales) du modèle. Les questions qui se posent sont alors les suivantes : "quelle série doit être résumée pour minimiser l'erreur globale ?", "Comment résumer une série pour minimiser l'erreur locale ?", "quel algorithme utiliser pour calculer l'approximation aussi vite que possible ?".

Exemple 1 *De nombreux sites Web souhaitent mesurer la fréquence des requêtes sur leurs URLs et la variation de cette fréquence dans le temps. Ces données ne peuvent toutefois pas être stockées étant donnés leur volume et la durée d'observation. Ainsi, un résumé fiable de ces données doit être calculé. Ce résumé doit permettre de montrer, pour chaque URL observée, la fréquence des requêtes et son évolution dans le temps avec une qualité d'approximation aussi bonne que possible.*

Il existe de nombreuses techniques capables de répondre au problème de l'exemple 1. Une première classe de techniques consiste à minimiser l'erreur locale. Une autre classe de techniques utilise un facteur d'ancienneté des données. La majorité des solutions existantes dans les flux de données est basée sur la deuxième classe de méthodes (utilisant l'ancienneté).

Dans cet article, notre objectif et de proposer la méthode REGLO capable (1) de se restreindre à l'espace mémoire disponible et (2) de calculer une approximation efficace et fiable d'une série temporelle. REGLO est une méthode rapide et fiable de compression des séries ayant une stratégie d'optimisation basée sur la minimisation de l'erreur globale des résumés de plusieurs séries. De plus, REGLO fonctionne en temps réel.

Ce papier est organisé de la manière suivante. La section 2 présente les techniques existantes. En section 3 nous présentons notre méthode REGLO minimisant l'erreur globale. En section 4 nous exposons les formules de calcul de la fusion de deux segments et de l'erreur résiduelle pour notre nouvelle technique d'approximation *Approximation par les Millieux* (AMi)et pour la *Régression Linaire*(RL). Enfin, la section 5 donne nos résultats d'expérimentations.

2 Etat de l'art

Concernant l'approximation d'une série temporelle, de très nombreuses techniques ont été proposées dans la littérature. Ce sujet a été étudié dans de nombreux domaines comme la finance (Zhu et Shasha (2002)), l'extraction de motifs fréquents (Giannella et al. (2003); Teng

et al. (2003)), la surveillance de réseaux (Borgne et al. (2007); Airoldi et Faloutsos (2004)) ou encore dans des applications comme la gestion des capteurs d'un réseau de distribution d'eau potable (Papadimitriou et al. (2005)). Les méthodes d'approximation proposées sont basées sur les ondelettes (Popivanov (2002); Chan et Fu (1999)), la transformée de Fourier (Faloutsos et al. (1994); Rafiei et Mendelzon (1997)), la régression linaire (RL) (Shatkay et Zdonik (1996); Keogh et Pazzani (1998)), etc.

Dans Lin et al. (2003), les auteurs proposent une représentation symbolique des séries temporelles avec l'algorithme SAX. SAX permet de discrétiser les données d'origine à l'aide de chaînes de symboles. Les manipulations se faisant ensuite sur les symboles plutôt que sur les données. Dans Yi et Faloutsos (2000), les auteurs proposent *segmented-means*, une méthode d'extraction du vecteur de caractéristiques des série temporelles. *segmented-means* découpe chaque série en plusieurs segments de longueur égale. Ensuite, pour chaque segment, les caractéristiques sont obtenues en calculant la moyenne des valeurs de ce segment. Un avantage important de *segmented-means* est son adaptation à n'importe quelle norme $\mathcal{L}_p$. Cependant la majorité des approches se placent dans le cadre de la norme $\mathcal{L}_2$. Dans ce cas, l'erreur quadratique est utilisée comme mesure de qualité de l'approximation entre le modèle choisi et les valeurs des séries temporelles. Par exemple la régression linéaire permet de construire une droite qui approche au mieux, au sens des moindres carrés, les séries temporelles d'origine.

Bellman (1961) utilise la programmation dynamique pour trouver une solution optimale à l'approximation d'une série temporelle par M modèles. Les contraintes sont (1) la mise à jour des modèles doit être récursive, (2) l'erreur quadratique globale d'approximation doit se décomposer en une somme d'erreurs locales sur chaque modèle. Comme la complexité de cette approche est en $O(N^2)$ où N est le nombre de valeurs observées cette approche n'est pas utilisable dans le cadre des flux de données. De plus une version incrémentale générique de l'algorithme de programmation dynamique est impossible Cependant quand $M = 2$ la complexité devient linéaire et la formule de mise à jour de l'erreur quadratique globale est calculable à partir des erreurs locales associées à chaque modèle. Ce sont ces propriétés qui seront utilisées pour mettre en IJuvre une approche incrémentale. Une approche incrémentale basée sur la transformée de Fourier est proposée dans Ogras et Ferhatosmanoglu (2006). Les auteurs proposent de trouver les coefficients qui minimiseraient l'erreur quadratique entre la séquence d'origine et sa représentation, le tout dans un contexte dynamique.

Chen et al. (2002) et Palpanas et al. (2008) proposent des techniques pour un calcul incrémental de la RL sur une série temporelle. Dans Chen et al. (2002) les auteurs proposent un principe intéressant de fusion des segments en temps linéaire mais ils ne donnent pas d'information sur le calcul de l'erreur. Dans Palpanas et al. (2008) les auteurs reprennent le principe de fusion des segments donné par Chen et al. (2002) et proposent un théorème qui montre que l'erreur entre l'approximation obtenue sur deux segments et les points d'origine peut être calculée, de manière récursive, comme la somme des erreurs entre (a) les deux segments et les points d'origine et (b) les deux segments et l'approximation sur ces deux segments. Ensuite, Palpanas et al. (2008) propose une étude sur l'intégration du facteur d'ancienneté dans le calcul de la RL. Chen et al. (2002) propose également l'utilisation d'un facteur d'ancienneté en stockant des résumés de séries temporelles dans des cubes. Le modèle d'ancienneté utilisé est celui des fenêtres logarithmiques (le facteur d'oubli des données augmente de manière logarithmique dans le temps). Teng et al. (2003) et Giannella et al. (2003) proposent de gérer l'historique des fréquents extraits à partir d'un flux en appliquant également un facteur d'an-

cienneté. Dans Giannella et al. (2003), le principe des fenêtres logarithmiques est appliqué afin d'accorder une plus grande importance à l'historique récent. Dans Teng et al. (2003), l'historique est représenté sous la forme d'une série temporelle et l'approximation est appliquée en priorité sur les segments anciens.

Citons également Zhu et Shasha (2002) qui propose de gérer plusieurs séries temporelles simultanément. Les auteurs proposent StatStream, une méthode permettant de gérer un ensemble de séries et de détecter les paires de séries ayant un coefficient de corrélation supérieur à un seuil donné. Les comparaisons sont faites sur les représentations condensées (basées sur les ondelettes) des données d'origine.

3 REGLO : principe général et algorithme

La sous-section 3.1 présente le principe général de notre approche afin de résumer un flux de séries temporelles. L'algorithme REGLO est décrit en sous-section 3.2.

3.1 Principe général

Soit $T[1..n]$, un ensemble de n séries temporelles à observer. Soit $T[j]$ la j^{eme} série de T alors à l'étape s on a $T[j] = (T[j][1], ..., T[j][s]]$ comme ensemble des observations (valeurs) de $T[j]$. Soit M la quantité de mémoire disponible (en d'autres termes, on peut stocker M équations de ces segments en mémoire). Nous considérons le cas de temps discrétisé. A chaque étape s nous avons une valeur pour chaque série (la valeur par défaut étant zéro). La valeur maximum de s permettant de stocker les mesures sans compression est $s = M/n$. Quand $(s \times n) > M$, la représentation des séries doit être compressée et la perte d'information est liée à la différence entre ces deux nombres.

Soit $R[1..n]$, l'ensemble des représentations des n séries et $S[i]$ la taille de $R[i]$ (*i.e.* le nombre de segments utilisés par la représentation $R[i]$). La représentation $R[i]$ de la série i est une approximation de cette série en $S[i]$ segments avec comme contrainte globale $\sum_{i=1}^{n} S[i] = M$.

Soit $E[i]$, l'erreur de $R[i]$ par rapport aux données d'origine de la série i et $E(R)$, l'erreur globale de $R[1..n]$. Alors $E(R) = \sum_{i=1}^{n} E[i]$.

Le premier problème traité dans cet article porte sur la recherche d'une distribution optimale des M segments disponibles afin de minimiser l'erreur globale $E(R)$ dans un traitement en ligne. Ainsi, à chaque étape s nous avons le choix entre :

1. étendre le dernier segment des séries en incluant s.
2. ou bien créer un nouveau segment et réunir les deux segments ayant la plus faible erreur locale.

Ce choix se fait en fonction de l'erreur globale $E(R)$ de ces mises à jour.

3.2 L'algorithme REGLO

Le principe de REGLO repose sur une mesure fondamentale mais compliquée à calculer rapidement : l'erreur résiduelle de la fusion de deux segments. Dans un environnement dynamique tel qu'un flux de données, l'optimisation la plus importante est de rendre récursif le

calcul de l'erreur résiduelle c'est-à-dire que l'erreur résiduelle de la fusion de deux segments se calcule en fonction de l'erreur résiduelle de chaque segment et du coût de fusion de ces deux segments. LŠautre avantage de REGLO est dŠavoir une gestion globale de la mémoire que lŠon ne retrouve pas dans les algorithmes qui résument les séries temporelles.

Algorithme REGLO
Entrée : T, un flux composé de n séries temporelles et M le nombre de segments disponibles pour le modèle.
Sortie : à chaque étape s, $R[1..n]$, les représentations de T avec une erreur globale optimisée. $R[i]$ est la représentation de la série i. $R[i][j]$ est la j^{eme} valeur de la i^{eme} série. $R[i].der$ et $R[i].avantder$ sont le dernier et l'avant-dernier élément de $R[i]$.

1. Tant que $(s \times n) < M, \forall i \in [1..n]$
   (a) Affecter un segment à chaque nouvelle valeur de T.
   (b) Calculer l'erreur de fusion entre le dernier élément arrivé dans une série et le précédent élément et mettre a jour la pile C qui conserve les coûts de fusion de $R[1..n]$ triés par ordre croissant.[1]
2. Pour chaque nouvel ensemble de n valeurs de l'étape s ($s > M/n$) :
   (a) $\forall i \in [1..n]$
      i. Calculer l'erreur de fusion de la nouvelle valeur $T[i][s]$ avec le segment précédent ;
      ii. Mise à jour de l'erreur de $R[i]$ et, au cas échéant, de la pile C.
   (b) Exécuter n fois les pas suivants[2] :
      i. Fusionner les deux segments, t et t', correspondants au premier élément de la pile C et obtenir le segment t'' ;
      ii. Mettre à jour le(s) coût(s)[3] de fusion de t'' avec les segments adjacents et la pile C.

Dans la section 4 nous présentons les moyens de calculer la fusion de deux segments et l'erreur résiduelle pour (RL) et (AMi).

4 Fusion de deux segments

4.1 Fusion de deux segments dans la cadre de la régression linéaire

Comme mentionné en section 2, Chen et al. (2002) donne un ensemble de formules permettant d'obtenir la RL sur deux segments en $O(1)$ et Palpanas et al. (2008) donne la formule permettant de mettre à jour en O(1) l'erreur issue de cette fusion.

[1] La pile est utilisée afin de retrouver, à chaque étape, l'élément avec le moindre coût de fusion.
[2] Afin de libérer n segments de mémoire
[3] Pour un élément intérieur à la série on a deux mise à jour à faire (liens gauche et droite) et pour le dernier élément d'une séries on a une seule mise à jour (lien gauche).

Considérons deux ensembles $S_1 = \{i|i = 1, \ldots, k\}$ et $S_2 = \{i|i = k+1, \ldots, s\}$ et S la fusion. Palpanas et al. (2008) montre que les coefficients de l'équation $Y = \alpha_1.X + \beta_1$ de la régression linéaire de S sont dérivés des paramètres des régressions des ensembles S_1 et S_2.

Dans Palpanas et al. (2008), le théorème 2 est important car il permet de calculer la somme des erreurs carrées sur S avec les erreurs carrées sur S_1 et S_2 et l'erreur entre les régressions locales sur S_1 et S_2 et la régression globale sur S. Ce qui s'exprime comme :

$$E(S) = E(S_1) + E(S_2) + \hat{E}(S_1 \cup S_2) \quad (1)$$

Selon le lemme 1 de Palpanas et al. (2008), l'erreur $\hat{E}(S_1 \cup S_2)$ qui est le coût de cette fusion, peut être calculée en $O(1)$. La mise à jour de l'erreur $E(S)$ permet de calculer le coût de la fusion de S_1 avec S_2. Ce coût est égal à $C(S_1 \cup S_2) = E(S) - E(S_1) - E(S_2) = \hat{E}(S_1 \cup S_2)$ et est utilisé dans l'étape 2 de l'algorithme REGLO.

4.2 Une approche rapide pour la fusion des segments

Bien que les résultats précédents permettent de fusionner et d'évaluer l'erreur et le coût de fusion en O(1), nous voulons optimiser encore cette étape. En effet, ces calculs sont au cœur de notre algorithme d'approximation destiné à résumer un flux de données. Toute optimisation de ces calculs sera donc nécessaire. Si l'opération de fusion est en temps linéaire, de meilleurs temps d'exécution peuvent être obtenus en diminuant le nombre d'opérations, ce qui peut être crucial dans le contexte des flux de données. Dans cette section, nous proposons de calculer une approximation de la RL grâce à une idée innovante basée sur l'observation suivante : quand le coût de fusion de deux segments est faible alors les points de croisements entre la droite de régression et les segments sont proches des milieux de ces segments.

Fusion de deux segments

La RL a pour résultat une représentation d'une série temporelle telle que chaque segment minimise la somme des erreurs quadratiques entre la représentation et les données d'origine. Notre approche consiste à élaborer une approximation intuitive et efficace de ces valeurs d'origine. AMi, notre technique d'approximation, sera représentée par la ligne qui coupe les segments d'origine en leurs milieux. Cette technique présente deux avantages principaux :

1. Elle ne demande pas de nombreux calculs impliquant autant de variables que la RL, ce qui la rend plus rapide que cette dernière.
2. Son interprétation est naturelle. Cette technique ne minimise pas la somme du carré des erreurs, mais nous montrons que l'approximation obtenue est très fidèle aux données d'origine. En effet, la somme du carré des erreurs a pour but de pénaliser les représentations qui ne tiennent pas compte des valeurs aberrantes. L'erreur sur ces valeurs étant rendues très importante dans le calcul global en raison de la puissance de deux utilisée. Privilégier les valeurs aberrantes n'est pas forcément le meilleur moyen d'avoir une représentation fidèle des données d'origine. Comme nous le montrerons, les représentations obtenues par AMi et la RL sont comparables. Toutefois, nous voulons être moins sensibles aux valeurs aberrantes. Ainsi, la représentation obtenue par AMi est un bon compromis entre les normes $\mathcal{L}_1$ et $\mathcal{L}_2$. En effet, nos expérimentations montrent que AMi peut avoir de meilleurs résultats quand l'erreur est mesurée dans la norme $\mathcal{L}_1$.

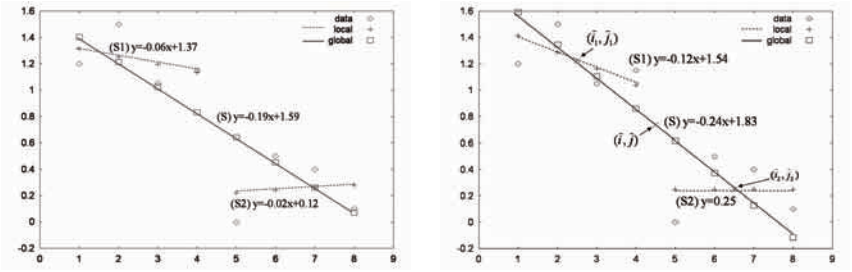

FIG. 1 – *Illustration de la RL (minimisant l'erreur quadratique) en comparaison d'AMi (ligne qui passe par les milieux des segments).*

Exemple 2 *La figure 1 illustre la différence entre la RL (gauche) et AMi (droite). Avec AMi, un premier ensemble de 4 segments est construit entre les points 1 et 2, les points 3 et 4, les points 5 et 6 et les points 7 et 8. Ensuite, le deuxième niveau d'approximation fusionne les deux premiers segments en passant par les milieux des segments et on obtient le segment $S1$ de la figure 1. Le même principe est appliqué aux deux segments restants (points 5 à 8) pour donner le segment $S2$. Le troisième niveau d'approximation fusionnera les deux segments $S1$ et $S2$ avec pour résultat S, le segment qui passe par leurs milieux (i.e. $(\bar{i}_1, \bar{j}_1)$ et $(\bar{i}_2, \bar{j}_2)$).*

Soit $(\bar{i}_1, \bar{j}_1)$ et $(\bar{i}_2, \bar{j}_2)$ les milieux des segments $S1$ et $S2$ avec $\bar{i}_1 = (k+1)/2$ et $\bar{i}_2 = (s+k+1)/2$. L'équation du segment S est obtenue par résolution des équations linéaires $\bar{j}_1 = \alpha.\bar{i}_1 + \beta$ et $\bar{j}_2 = \alpha.\bar{i}_2 + \beta$. La pente α et l'ordonnée β sont donnés par $\alpha = (\bar{j}_1 - \bar{j}_2)/(\bar{i}_1 - \bar{i}_2)$ et $\beta = (\bar{j}_2.\bar{i}_1 - \bar{j}_1.\bar{i}_2)/(\bar{i}_1 - \bar{i}_2)$ et le nouveau point milieu $(\bar{i}, \bar{j})$ de S est égal à $\bar{i} = (s+1)/2$ et $\bar{j} = \alpha.\bar{i} + \beta$. Le calcul de $\bar{j}$ est donné par $\bar{j} = k.\bar{j}_1 + (s-k)\bar{j}_2/s$

Mise à jour de l'erreur de AMi

Théorème 1 *L'erreur commise par AMi sur le segment $S = S_1 \cup S_2$ peut être calculée de manière récursive à partir des erreurs pondérées des segments S_1 et S_2 et de la pente α du segment S.*

Preuve : La somme des erreurs carrées de S peut être décomposée en deux parties :

$$E(S) = \sum_{i=1}^{k} (j_i - \alpha.i - \beta)^2 + \sum_{i=k+1}^{s} (j_i - \alpha.i - \beta)^2$$

La première partie de cette décomposition peut être réécrite de la manière suivante :

$$\sum_{i=1}^{k} (j_i - \alpha.i - \beta - (\alpha_1.i + \beta_1) + (\alpha_1.i + \beta_1))^2$$

$$= \sum_{i=1}^{k} (j_i - \alpha_1.i - \beta_1)^2 + (\alpha_1.i + \beta_1 - \alpha.i - \beta)^2$$

$$+2(j_i - \alpha_1.i - \beta_1)\big((\alpha_1 - \alpha).i + (\beta_1 - \beta)\big)$$

La première somme des erreurs quadratiques dans la première ligne de cette réécriture représente la somme des erreurs quadratiques de S_1 obtenue avec AMi. La seconde partie représente les erreurs entre le segment S_1 obtenu par AMi et le segment S obtenu par AMi calculé sur S_1. Comme le modèle obtenu par la RL, AMi vérifie que $\sum_{i=1}^{k}(j_i - \alpha_1.i - \beta_1) = 0$ et la dernière partie peut être simplifiée par $\Delta(S_1/\alpha) = 2(\alpha_1 - \alpha)\sum_{i=1}^{k}(j_i - \alpha_1.i - \beta_1)i$

Avec AMi, la somme des erreurs quadratiques de $E(S)$ est plus complexe que la somme des erreurs quadratiques de la RL définie par l'équation 1 mais la complexité par rapport à s est inchangée. $E(S)$ peut être exprimé de la manière suivante :

$$E(S) = E(S_1) + E(S_2) + \hat{E}(S_1 \cup S_2) + \Delta(S_1/\alpha) + \Delta(S_2/\alpha) \quad (2)$$

Le calcul de $\Delta(S_1/\alpha)$ ou $\Delta(S_2/\alpha)$ dépend de la pente du nouveau segment S et des valeurs $\delta(S_1) = \sum_{i=1}^{k} i.j_i$ et $\delta(S_2) = \sum_{i=k+1}^{s} i.j_i$. Ainsi, la relation linéaire $\delta(S) = \delta(S_1) + \delta(S_2)$ existe et δ est maintenu pour chaque segment.

De plus, $\Delta(S_1/\alpha)$ peut se calculer facilement : $\Delta(S_1/\alpha) = -\alpha.\delta(S_1) + \Lambda(S_1)$ où $\Lambda(S_1)$ dépend seulement du segment S_1 et on obtient :

$$E(S) = (E(S_1) + \Lambda(S_1)) + (E(S_2) + \Lambda(S_2)) + \hat{E}(S_1 \cup S_2) - \alpha(\delta(S_1) + \delta(S_2))$$

5 Expérimentations

Dans cette section, nous évaluons l'efficacité et la qualité de notre méthode d'approximation grâce à un ensemble d'expérimentations. Nous avons implémenté deux versions de REGLO. Dans la première version, l'approximation et le calcul de l'erreur sont basés sur la RL. Dans la deuxième version, l'approximation et le calcul de l'erreur sont basés sur AMi (C.f. section 4.2). Nous avons également implémenté une compression basée sur les ondelettes et une autre basée sur les fenêtres logarithmiques afin de comparer REGLO avec les techniques les plus importantes. Une description des ondelettes peut être trouvée en Daubechies (1992). Les fenêtres logarithmiques sont décrites en Chen et al. (2002); Giannella et al. (2003). Nous avons évalué nos algorithmes sur deux jeux de données réelles. Le premier jeu de données, "NYSE", est construit sur les données téléchargées depuis `http://icf.som.yale.edu/nyse`. Ce jeu contient 134 séries temporelles, chacune contenant 804 valeurs correspondant aux valeurs du marché de New-York de 1815 à 1925. Le deuxième jeu de données, "WEB", vient des logs d'accès Web anonymisés de l'Inria Sophia-Antipolis. Nous avons recueilli un an d'usage, pour un total de 14 Go. Un premier prétraitement a permis d'extraire les URLs les plus fréquentes de ce jeu. Avec un support minimum de 1%, nous en avons trouvé 223. Ensuite, nous avons reporté les variations du nombre de requêtes à ces URLs avec une fenêtre sautante de taille 1000. En d'autres termes, chaque série est mise à jour toutes les 1000 requêtes et la valeur affectée à la série correspond au nombre de requêtes sur l'URL pendant cet intervalle.

REGLO (versions RL et AMi) est implémenté en Java. Les expérimentations sont effectuées sur un PC équipé de 2 Gb de mémoire et un pentium à 4 processeurs cadencés à 2,2 Ghz.

5.1 Comparaison entre AMi et la RL

Dans ce premier ensemble d'expérimentations nous évaluons, sur les deux jeux de données :

- L'impact de AMi sur l'erreur moyenne à chaque étape.
- La différence des temps d'exécutions entre AMi et la RL.

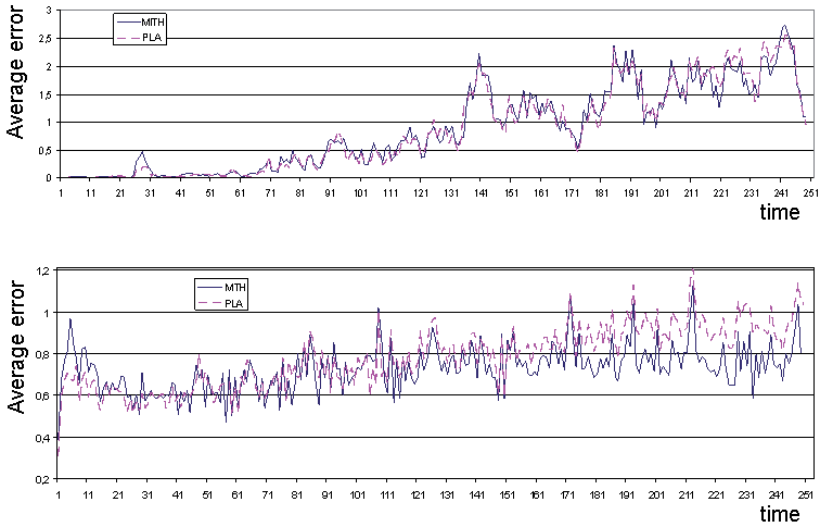

FIG. 2 – *Erreur moyenne étape par étape pour AMi et la RL sur NYSE et WEB.*

La figure 2 montre l'erreur moyenne des approximations obtenues par AMi et la RL pour les données NYSE (haut) et WEB (bas) sur les 251 premières valeurs. L'erreur est la moyenne des erreurs entre les représentations de l'enregistrement i dans les n séries et la valeur réelle de i. Ainsi, pour chaque unité de temps, nous connaissons l'erreur moyenne du modèle pour les n séries temporelles. Pour NYSE les erreurs moyennes commises par AMi et la RL sont très proches. La qualité de l'approximation obtenue par AMi est illustrée sur le jeu de données WEB. Pour ce jeu, AMi obtient une très bonne erreur moyenne en comparaison de la RL. La sous-section 5.2 donne les erreurs globales moyennes des deux méthodes d'approximation et AMi obtient de meilleurs résultats (0,87) que la RL (2,52). Cela est dû au fait que AMi ne tente pas d'optimiser la somme des erreurs carrées. En effet, notre but est trouver un compromis entre l'erreur minimale en $\mathcal{L}_1$ et celle en $\mathcal{L}_2$, ce pour quoi AMi obtient des résultats meilleurs que la RL (comme illustré par la figure 2).

La figure 3 montre les temps d'exécution de REGLO sur les deux jeux de données, sont systématiquement plus faible pour AMi. L'explication vient du nombre considérablement réduit d'opérations (Cf. les formules de la section 4.2) nécessaires à AMi pour calculer l'approximation et l'erreur résiduelle en comparaison des formules complexes (que nous avons pourtant simplifiées autant que nous le pouvons) de la RL (Cf. section 4.1).

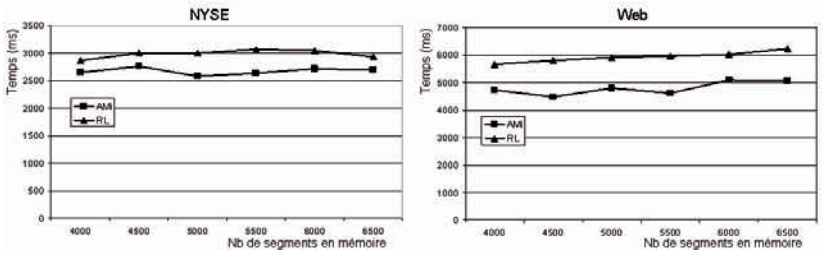

FIG. 3 – *Temps d'exécution de AMi et de la RL sur NYSE et WEB en fonction de la mémoire disponible.*

5.2 Ondelettes et fenêtres logarithmiques

L'objectif de cette section est de comparer l'erreur commise par REGLO avec celles des ondelettes et des fenêtres logarithmiques.

Le tableau 1 donne l'erreur globale moyenne de chaque modèle en comparaison aux données d'origine. Pour chaque modèle, l'erreur moyenne est calculée à chaque étape (comme décrit à la section 5.1). Ensuite, la valeur moyenne des erreurs de chaque modèle est reportée dans la table 1. Les modèles obtenus par AMi et la RL disposent chacun de 5000 segments. Les modèles basés sur les fenêtres logarithmiques demandent 11 segments par série (11 étant obtenu par le logarithme de s, le nombre maximum de valeurs dans les séries du jeu de données). Le modèle basé sur les ondelettes dispose d'un nombre de segments correspondant à celui de REGLO.

TAB. 1 – *Erreurs globales des ondelettes, des fenêtres logarithmiques, de la RL et de AMi*

	WEB	NYSE
Fenêtres	17.245627	2.451278
Ondelettes	14.0588	1.55812
RL	2.529895911	0.857884432
AMi	0.879335223	0.868300858

6 Conclusion

Dans cet article, nous avons présenté (1) REGLO, une stratégie de résumé d'un flux de séries temporelles et (2) AMi, un principe rapide d'approximation des valeurs d'une série. REGLO est capable de gérer de nombreuses séries temporelles sans limite de taille grâce à son principe d'équilibre et de distribution de la mémoire entre les représentations. Ce principe permet à chaque série d'obtenir ou de restituer de l'espace mémoire selon ses besoins de précision. Notre principe d'approximation rapide est comparé à la régression linéaire dans ses aspects formels mais aussi grâce à une batterie d'expérimentations. Notre étude sur les

relations entre AMi et la RL montre que AMi permet un calcul récursif de l'erreur entre la représentation et les données d'origine. Nous avons évalué nos algorithmes sur des données réelles afin de montrer les performances de notre technique d'approximation et de la distribution de l'espace mémoire. Ces expérimentations montrent que nos objectifs (diminuer l'erreur globale et accélérer les temps de calcul des résumés) sont atteints. Parmi les pistes ouvertes par ce travail citons l'étude des caractéristiques des données sur lesquelles l'approximation obtenue par AMi est meilleure que celle de la RL.

Références

Airoldi, E. et C. Faloutsos (2004). Recovering latent time-series from their observed sums : network tomography with particle filters. In *KDD '04 : Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, pp. 30–39. ACM.

Bellman, R. (1961). On the approximation of curves by line segments using dynamic programming. *Commun. ACM 4*(6), 284–292.

Borgne, Y.-A. L., S. Santini, et G. Bontempi (2007). Adaptive model selection for time series prediction in wireless sensor networks. *Signal Process. 87*(12), 3010–3020.

Chan, K. P. et A. W. C. Fu (1999). Efficient time series matching by wavelets. In *ICDE '99 : Proceedings of the 15th International Conference on Data Engineering*, Washington, DC, USA, pp. 126. IEEE Computer Society.

Chang, J. H. et W. S. Lee (2003). Finding recent frequent itemsets adaptively over online data streams. In *KDD'03 : Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 487–492.

Chen, Y., G. Dong, J. Han, B. Wah, et J. Wang (2002). Multidimensional regression analysis of time-series data streams. In *VLDB '02 : Proceedings of the 28th Int. Conf. on Very Large Data Bases*. VLDB Endowment.

Cheng, H. et P.-N. Tan (2008). Semi-supervised learning with data calibration for long-term time series forecasting. In *KDD '08 : Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, pp. 133–141. ACM.

Cohen, E. et M. Strauss (2003). Maintaining time-decaying stream aggregates. In *PODS '03 : Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 223–233.

Daubechies, I. (1992). *Ten lectures on wavelets*. Philadelphia, PA, USA : Society for Industrial and Applied Mathematics.

Ding, H., G. Trajcevski, P. Scheuermann, X. Wang, et E. Keogh (2008). Querying and mining of time series data : experimental comparison of representations and distance measures. *Proc. VLDB Endow. 1*(2), 1542–1552.

Faloutsos, C., M. Ranganathan, et Y. Manolopoulos (1994). Fast subsequence matching in time-series databases. In *SIGMOD '94 : Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp. 419–429. ACM.

Giannella, C., J. Han, J. Pei, X. Yan, et P. Yu (2003). *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining. AAAI/MIT.

Keogh, E. J. et M. J. Pazzani (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, pp. 239–243.

Lin, J., E. J. Keogh, S. Lonardi, et B. Y. chi Chiu (2003). A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, pp. 2–11.

Ogras, Y. et H. Ferhatosmanoglu (2006). Online summarization of dynamic time series data. *The VLDB Journal 15*(1), 84–98.

Palpanas, T., M. Vlachos, E. J. Keogh, et D. Gunopulos (2008). Streaming time series summarization using user-defined amnesic functions. *IEEE Trans. Knowl. Data Eng. 20*(7), 992–1006.

Papadimitriou, S., J. Sun, et C. Faloutsos (2005). Streaming pattern discovery in multiple time-series. In *VLDB '05 : Proceedings of the 31th Int. Conf. on Very Large Data Bases*, pp. 697–708. VLDB Endowment.

Popivanov, I. (2002). Similarity search over time-series data using wavelets. In *ICDE '02 : Proceedings of the 18th International Conference on Data Engineering*, Washington, DC, USA, pp. 212. IEEE Computer Society.

Rafiei, D. et A. Mendelzon (1997). Similarity-based queries for time series data. In *SIGMOD '97 : Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, New York, NY, USA, pp. 13–25. ACM.

Shatkay, H. et S. B. Zdonik (1996). Approximate queries and representations for large data sequences. In *ICDE '96 : Proceedings of the Twelfth International Conference on Data Engineering*, Washington, DC, USA, pp. 536–545. IEEE Computer Society.

Teng, W.-G., M.-S. Chen, et P. S. Yu (2003). A Regression-Based Temporal Pattern Mining Scheme for Data Streams. In *VLDB '03 : Proceedings of the 29th Int. Conf. on Very Large Data Bases*, pp. 93–104. VLDB Endowment.

Yi, B.-K. et C. Faloutsos (2000). Fast time sequence indexing for arbitrary lp norms. In *VLDB '00 : Proceedings of the 26th Int. Conf. on Very Large Data Bases*, pp. 385–394. VLDB Endowment.

Zhu, Y. et D. Shasha (2002). Statstream : statistical monitoring of thousands of data streams in real time. In *VLDB '02 : Proceedings of the 28th Int. Conf. on Very Large Data Bases*, pp. 358–369. VLDB Endowment.

Summary

Summarizing a set of streaming time series is an important issue that allows monitoring and storing reliable information. At this time, most efforts have been done for summarizing time series separately. However, in the case of data streams, algorithms have to be ressource-aware, which involves several different strategies for memory management. In this paper, our point of view is that memory should be partitionned in order to optimize the global error of the model.