Recommandations Personnalisées de Requêtes MDX

Elsa Negre

Laboratoire d'Informatique, Université François Rabelais Tours, France
elsa.negre@univ-tours.fr

Résumé. Une session d'analyse OLAP peut être définie comme une session interactive durant laquelle un utilisateur lance des requêtes pour naviguer dans un cube. Très souvent, choisir quelle partie du cube va être naviguée par la suite, et, de ce fait, concevoir la prochaine requête, est une tâche difficile. Dans cet article, nous proposons d'utiliser ce que tous les utilisateurs du système OLAP ont fait pendant leurs précédentes explorations du cube afin de recommander des requêtes MDX à l'utilisateur.

1 Introduction

Les utilisateurs de systèmes OLAP naviguent interactivement dans un cube en lançant une séquence de requêtes sur un entrepôt de données, ce que nous appelons une session d'analyse (ou session) dans la suite de l'article. Ce processus est souvent pénible puisque l'utilisateur peut ne pas avoir d'idée sur ce que pourrait être la prochaine requête [Sarawagi (2000)].

Le problème abordé dans Giacometti et al. (2008) et revu dans cet article est donc le suivant : Comment aider l'utilisateur à concevoir sa prochaine requête ?

Comme réponse, nous proposons d'exploiter ce que les utilisateurs (l'utilisateur courant et les autres utilisateurs du système OLAP) ont fait lors de leurs précédentes navigations dans le cube, et d'utiliser cette information comme base pour recommander ce que pourrait être la prochaine requête. À cet effet, nous présentons une instanciation particulière avec un lien vers la personnalisation d'un cadre générique existant. Celui-ci permet de recommander des requêtes MDX, en utilisant le log du serveur, c'est-à-dire, l'ensemble des sessions précédentes sur le cube, et la séquence de requêtes de la session courante.

Le cadre est fondé sur le processus suivant :

1. Partitionner le log pour regrouper des requêtes qui sont similaires, afin de faire face au problème de faible densité du log.
2. Générer des recommandations candidates en commençant par trouver quelles sessions du log coïncident avec la session courante et ensuite, prédire ce que peut être la prochaine requête.
3. Ordonner les requêtes candidates en présentant à l'utilisateur la requête la plus pertinente en premier.

Ce cadre est générique dans le sens où il n'impose pas une méthode particulière de partitionnement du log, de génération des requêtes candidates ou d'ordonnancement de celles-ci. Au lieu de cela, ces actions sont laissées comme paramètres du cadre qui peut être instancié de diverses manières afin de changer la méthode de calcul des recommandations. Le raisonnement sousjacent à cette proposition d'un cadre générique est que les recommandations dépendent fortement des utilisateurs et des données sur lesquelles les recommandations sont calculées. Adomavicius et Tuzhilin (2005) explique le besoin de systèmes de recommandations flexibles et adaptés à l'utilisateur.

Cet article est organisé comme suit : la section 2 présente les travaux existants, la section 3 motive notre approche grâce à un exemple simple, la section 4 présente les définitions formelles des notions utilisées dans l'article et la section 5 propose une instanciation particulière de notre cadre. La section 6 présente nos résultats expérimentaux. Enfin, les conclusions et les travaux futurs sont abordés dans la section 7.

2 Travaux existants

D'après nos connaissances dans le domaine, nous avons été les premiers à traiter le problème de la recommandation de requêtes OLAP (et particulièrement de requêtes MDX [1]) dans Giacometti et al. (2008).

L'idée d'employer ce que les autres utilisateurs ont fait pour produire des recommandations est très populaire dans le domaine de la Recherche d'Informations [Adomavicius et Tuzhilin (2005)], et dans l'exploitation des usages du Web (Web Usage Mining) [Srivastava et al. (2000)]. Par exemple, Baeza-Yates et al. (2004) emploie l'algorithme des k-means pour grouper des requêtes soumises à un moteur de recherche, génère des recommandations candidates et ordonne les candidats.

Notre contribution est d'adapter ces techniques existantes à OLAP. Dans le domaine OLAP, le travail de Sapia (1999, 2000) a un point commun avec notre travail : prévoir la prochaine requête OLAP. Néanmoins, il existe beaucoup de différences entre leurs travaux et le nôtre :
Premièrement, la principale préoccupation de Sapia (1999, 2000) est de prétraiter les données mais pas de recommander une requête.
Deuxièmement, Sapia (1999, 2000) ne traite pas des requêtes MDX. Or, la manière de regrouper les requêtes en classes que nous proposons, repose uniquement sur le schéma de la requête (c'est-à-dire, les dimensions et les niveaux). De plus, la distance que nous utilisons prend en compte les membres [2].
Troisièmement, un modèle de Markov est employé pour prévoir la prochaine requête, tandis que nous avons choisi de ne pas utiliser un modèle probabiliste.

[1] Notre choix s'est porté sur des requêtes MDX car il n'existe pas de langage de requêtes standard pour la manipulation de cubes de données, mais MDX est le plus employé.

[2] Il est facile de trouver les ensembles de membres d'une requête MDX sans évaluer cette requête (si les dimensions tiennent en mémoire), ce qui n'est pas le cas pour les requêtes relationnelles en général.

Ainsi, le cadre générique pour la recommandation de requêtes exposé dans Giacometti et al. (2008) utilise la séquence de requêtes de la session courante ainsi que le log de requêtes du serveur OLAP, c'est-à-dire, les séquences de requêtes précédemment posées sur le cube.

Cela consiste en trois étapes, détaillées Figure 1 :

1. La première étape consiste à utiliser une technique de partitionnement d'un ensemble de requêtes pour partitionner le log de requêtes afin d'obtenir toutes les sessions généralisées (celles issues du log et la session généralisée courante) (Figure 1 : Partie 1 : Partitionnement).
2. La seconde étape consiste à utiliser la session généralisée courante et l'ensemble des sessions généralisées du log afin de prédire les recommandations candidates (Figure 1 : Partie 2 : Prédiction). Il s'agit d'abord de trouver les sessions généralisées qui coïncident avec la session généralisée courante, puis de retenir les classes candidates à la recommandation et enfin de retourner les requêtes candidates qui correspondent aux classes candidates.
3. La dernière étape consiste à ordonner les recommandations (requêtes candidates) en fonction d'un critère de satisfaction (Figure 1 : Partie 3 : Ordonnancement).

Chacune de ces étapes est paramétrée avec une ou plusieurs fonctions. En changeant ces paramètres, la manière de générer les recommandations change.

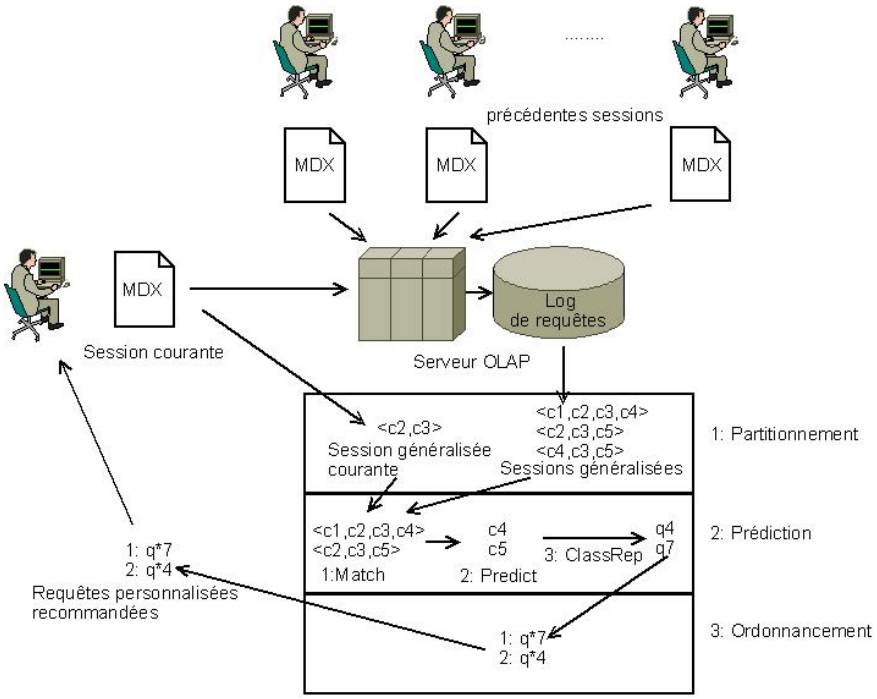

FIG. 1 – *Vue d'ensemble du cadre générique*

3 Exemple

Dans cette section, nous illustrons avec un exemple simple l'idée générale de notre cadre présenté Figure 1.

Considérons un serveur OLAP utilisé par plusieurs utilisateurs. Chaque utilisateur peut ouvrir une session sur le serveur pour naviguer dans le cube en lançant une séquence de requêtes exprimées en langage MDX. Le serveur enregistre ces sessions, c'est-à-dire, les séquences de requêtes lancées pendant chaque session. Par exemple, supposons que le log soit composé des trois sessions suivantes : $s_1 = \langle q_1, q_2, q_3, q_4 \rangle, s_2 = \langle q_5, q_6, q_7 \rangle, s_3 = \langle q_8, q_9, q_{10} \rangle$ où les q_i sont des requêtes MDX.

Supposons maintenant qu'une nouvelle session, appelée *session courante*, soit ouverte par un utilisateur. Par exemple, cette session est : $S_c = \langle q_{11}, q_{12} \rangle$. Cet utilisateur pourrait être intéressé par la façon dont les autres utilisateurs ont exploré le cube et utiliser cette information pour concevoir sa prochaine requête.

Au vu du nombre d'utilisateurs et le nombre de sessions possibles, le log peut être très volumineux, mais pas autant que le cube lui-même. En outre, les divers utilisateurs peuvent avoir différents intérêts, ainsi, leurs requêtes peuvent naviguer dans des parties très différentes du cube. Cela signifie qu'il est peu probable de trouver une requête donnée plus d'une fois dans le log. Ainsi, le log peut être volumineux et épars.
Afin de faire face à la taille et à la dispersion, les requêtes du log peuvent être groupées dans des classes, pour *partitionner* le log, une classe de requêtes étant un ensemble de requêtes proches les unes des autres. Et, dans les sessions du log, les requêtes sont remplacées par la classe à laquelle elles appartiennent, ce que nous appelons une *session généralisée*. Dans notre exemple, supposons que les sessions généralisées sont : $g_1 = \langle c_1, c_2, c_3, c_4 \rangle, g_2 = \langle c_2, c_3, c_5 \rangle, g_3 = \langle c_4, c_3, c_5 \rangle$ avec, q_1 appartient à la classe c_1, q_2, q_5 appartiennent à la classe c_2, q_3, q_6, q_9 appartiennent à la classe c_3, ... (Figure 1 : Partie 1 : Partitionnement, Droite : Sessions généralisées)

Naturellement, dans la session courante elle-même, les requêtes peuvent être aussi remplacées par la classe à laquelle elles appartiennent. Dans notre exemple, supposons que la *session généralisée courante* est $g_c = \langle c_2, c_3 \rangle$, puisque q_{11} appartient à c_2 et q_{12} appartient à c_3. (Figure 1 : Partie 1 : Partitionnement, Gauche : Session généralisée courante)

Maintenant, il est possible de constater que la session courante S_c coïncide ("*match*") avec les sessions s_1 et s_2 du log (puisque, dans notre cas, g_c est une sous-séquence de g_1 et de g_2) (Figure 1 : Partie 2 : Prédiction, Droite : Match). L'utilisateur pourrait être intéressé par le fait de connaître les requêtes qui suivent la séquence qui coïncide avec ces sessions. En observant les sessions généralisées g_1 et g_2, nous pouvons *prédire* que de telles requêtes appartiennent aux classes $c_4 = \{q_4, q_8\}$ et $c_5 = \{q_7, q_{10}\}$, appelées les *classes candidates* (Figure 1 : Partie 2 : Prédiction, Centre : Predict). Nous souhaitons proposer des requêtes à l'utilisateur, et non des classes, une unique requête doit être extraite de chaque classe candidate. Il peut s'agir, par exemple, de la requête qui *représente* le mieux la classe. Dans notre exemple, supposons que ces requêtes sont q_4 et q_7, appelées les *requêtes candidates* (Figure 1 : Partie 2 : Prédiction, Droite : ClassRep). Enfin, l'utilisateur peut être intéressé par seulement une requête. Par conséquent, les requêtes candidates devraient être présentées dans un ordre donné. Ainsi, les requêtes candidates doivent être *ordonnées*, par exemple, en fonction de leur adéquation avec

le profil de l'utilisateur (Figure 1 : Partie 3 : Ordonnancement). Dans notre exemple, supposons que l'ordre est $\langle q_7, q_4 \rangle$, et donc, cet ensemble ordonné est recommandé à l'utilisateur. La première requête proposée est la requête q_7. Si l'utilisateur ne la considère pas pertinente, il se verra proposer la requête q_4.

Le cadre générique que nous proposons dans Giacometti et al. (2008) permet d'aller au delà de cet exemple simple. En effet, la technique de partitionnement du log, d'appariement des sessions, de prédiction des classes candidates, le représentant de chaque classe et l'ordonnancement sont laissés comme paramètres. Diverses instanciations du cadre peuvent être imaginées, comme vu dans Giacometti et al. (2008). L'une d'entre elles est présentée dans la section 5.

4 Définitions formelles

Dans cette section, nous donnons les définitions formelles des notions utilisées dans l'article.

Soit R une instance de relation de schéma $sch(R) = \{A_1, \ldots, A_N\}$. Nous notons $adom(A_i)$ le domaine actif de chaque attribut $A_i \in sch(R)$.

Cubes et Dimensions Un cube C de dimension N est un tuple $C = \langle D_1, \ldots, D_N, F \rangle$ où :
- Pour $i \in [1, N]$, D_i est une table de dimension de schéma $sch(D_i) = \{L_i^0, \ldots, L_i^{d_i}\}$. Pour chaque dimension $i \in [1, N]$, chaque attribut L_i^j décrit un niveau d'une hiérarchie, j étant la profondeur de ce niveau. L_i^0 est le niveau le plus bas ainsi que la clé primaire de D_i.
- F est une table de fait de schéma $sch(F) = \{L_1^0, \ldots, L_N^0, m\}$ où m est un attrinut numérique.

Dans la suite, notons que le nom d'une dimension $D_i, i \in [1, N]$ est aussi utilisé pour parler d'un attribut de domaine actif $adom(D_i) = \bigcup_{j=0}^{d_i} adom(L_i^j)$. Pour tout $i \in [1, N]$, $adom(D_i)$ est l'ensemble de tous les membres de la dimension D_i.
Notons également que, dans cet article, la définition du cube est faite sans perte de généricité puisque dans le problème que nous considérons, nous ne nous intéressons pas à la manière dont les requêtes sont évaluées mais simplement à la manière dont elles sont exprimées.
Notons enfin que, une des dimensions D_i pourra être la dimension *Mesure*[3], constituée d'une seule hiérarchie où tous les membres ont comme ancêtre commun, la racine de la hiérarchie.

Membre Soit C un cube de dimension N, un membre m est un élément de $\bigcup_{i \in [1,N]} adom(D_i)$.

Référence de cellule Soit C un cube de dimension N, une référence de cellule (ou référence, par abus) est un N-uplet $\langle r_1, \ldots, r_N \rangle$ où $r_i \in adom(D_i)$ pour tout $i \in [1, N]$.

Soit un cube C, nous notons par $ref(C)$, l'ensemble de toutes les références de C.

[3] Dimension plate

Distance entre références Soit un cube C, une distance entre références de $ref(C)$, est une fonction de $ref(C) \times ref(C)$ vers un ensemble de nombres réels.

Requête Dans cet article, nous considérons uniquement des requêtes MDX simples, vues comme un ensemble de références, comme défini dans Bellatreche et al. (2005). Plus particulièrement, considérant que les clauses SELECT et WHERE d'une requête MDX définissent l'ensemble de références que l'utilisateur veut extraire du cube, nous proposons de voir les requêtes MDX comme des ensembles de références, pour une instance donnée du cube.

Soit $C = \langle D_1, \ldots, D_N, F \rangle$ un cube de dimension N et $R_i \subseteq adom(D_i)$ un ensemble de membres de la dimension D_i pour tout $i \in [1, N]$. Une requête sur un cube C de dimension N est un ensemble de références $R_1 \times \ldots \times R_N$.

Soit un cube C, nous notons $req(C)$ l'ensemble des requêtes possibles sur C.

Distance entre requêtes Soit un cube C, une distance entre requêtes de $req(C)$ est une fonction de $req(C) \times req(C)$ vers un ensemble de nombres réels.

Session d'analyse Soit un cube C, une session d'analyse (ou session, par abus) $s = \langle q_1, \ldots, q_p \rangle$ sur C est une séquence finie de requêtes de $req(C)$. Nous notons $req(s)$, l'ensemble des requêtes de la session s, $session(C)$ l'ensemble de toutes les sessions sur le cube C et $s[i]$ la $i^{ème}$ requête de la session s.

Log d'une base de données Soit un cube C, un log de base de données (ou log, par abus) est un ensemble fini de sessions. Nous notons $req(\mathscr{L})$, l'ensemble des requêtes contenues dans le log $\mathscr{L}$.

Classe de requêtes Soit un cube C, une classe de requêtes est un ensemble $Q \subseteq req(C)$.

Représentant de classe Soit un cube C, un représentant de classe est une fonction de $2^{req(C)}$ dans $req(C)$.

Partitionnement d'un ensemble de requêtes Soit un cube C et une distance entre requêtes, un partitionnement d'un ensemble de requêtes est une fonction p de $2^{req(C)}$ dans $2^{2^{req(C)}}$ telle que, pour tout $Q \subset req(C)$, p calcule une partition de Q sous la forme d'un ensemble P de paires disjointes de classes de requêtes.

Classificateur de requête Soit un cube C, un classificateur de requête cl est une fonction de $req(C) \times 2^{2^{req(C)}}$ dans $2^{req(C)}$ telle que, si $q \in req(C)$ est une requête, $P \subseteq 2^{req(C)}$ est un ensemble de classes alors $cl(q, P) \in P$. Nous avons $cl(q, P)$ est la classe de q.

Session généralisée Soit une session s et un ensemble de classes de requêtes, la session généralisée de s est la séquence des classes de chaque requête de s.
Formellement, soit un cube C, un ensemble de classes de requêtes P, un classificateur de requête cl et $s = \langle q_1, \ldots, q_p \rangle$ une session sur C, la session généralisée gs de s est la séquence $\langle c_1, \ldots, c_p \rangle$ où :

- $c_i = cl(q_i, P)$ est la classe de q_i pour tout $i \in [1, p]$.
- $\forall i, c_i \in P$.
- $\forall i, q_i \in req(C)$.

Nous notons $gs[i]$ la $i^{ème}$ classe de la session généralisée gs.

Ordonnancement de requêtes Soit un cube C et un ensemble de requêtes $S \in 2^{req(C)}$, un ordonnancement de requêtes $ordre$ est une fonction de $2^{req(C)}$ vers un tuple de requêtes, tel que $ordre(S)$ ordonne les requêtes de S.

Profil utilisateur Soit un cube C de dimension N, un profil utilisateur Γ sur C est un tuple $\Gamma = \langle <_d, \{<_1, \ldots, <_N\}\rangle$ où :

- $<_d$ est un ordre total sur l'ensemble des dimensions de C. Soit deux dimensions D_i et D_j, nous considérons que D_i est moins intéressante que D_j si $D_i <_d D_j$.
- Pour tout $i \in [1, N]$, $<_i$ est un ordre total sur $adom(D_i)$. Soit deux membres m et m' de $adom(D_i)$, nous considérons que m est moins intéressant que m' si $m <_i m'$.

Ordre sur les références Soit un profil utilisateur Γ sur un cube C et soit r et r' deux références telles que $r, r' \in ref(C)$, nous notons $\Delta(r, r')$ l'ensemble des dimensions où r et r' diffèrent. r est moins intéressant que r', noté $r \leq_\Gamma r'$, si pour toute dimension $D_i \in max_{<_d}\Delta(r, r')$, nous avons $r(D_i) <_i r'(D_i)$.
Notons que l'ordre $\leq_\Gamma$ défini sur les références est un ordre lexicographique total.

5 Une instanciation particulière du cadre

Dans cette section, nous présentons une instanciation particulière de notre cadre.

5.1 Étape 1 : Partitionnement - Génération des sessions généralisées

5.1.1 Distance entre requêtes

Dans cet article, nous nous intéressons uniquement à des requêtes MDX simples comme définies dans Giacometti et al. (2008), c'est-à-dire, des ensembles de références.

Exemple : Les données
Soit un cube $C = \langle$PRODUIT, LOCALISATION, TEMPS, VENTES$\rangle$ où :

- sch(PRODUIT) $= \{IdProd, Type, All\}$,
- sch(LOCALISATION) $= \{Ville, Departement, Region, Pays, All\}$,
- sch(TEMPS) $= \{Jour, Mois, Annee, All\}$,
- sch(VENTES) $= \{IdProd, Ville, Jour, Quantite\}$

et deux requêtes q_1 et q_2 telles que :

$q_1 =$ Quantité vendue de produit P_1 dans la ville de Blois pour les années 2007 et 2008
$= \{\langle P_1, Blois, 2007\rangle, \langle P_1, Blois, 2008\rangle\} = \{r_1^1, r_1^2\}$

$q_2 =$ Quantité vendue de tous les produits dans la région Centre pour l'année 2008
$= \{\langle All, Centre, 2008\rangle\} = \{r_2^1\}$

Pour calculer la distance entre de tels ensembles, la distance de Hausdorff [Hausdorff (1914); Matheron (1975)] peut être utilisée car elle permet de comparer deux ensembles. Cette distance est fondée sur le calcul de la distance entre les éléments des ensembles, les références dans notre cas.

Définition 5.1 *La distance de Hausdorff d_{H} entre deux ensembles q_1 et q_2 est définie par :*

$$d_{\mathrm{H}}(q_1, q_2) = \max\{\, \max_{r_1 \in q_1} \min_{r_2 \in q_2} d(r_1, r_2), \\ \max_{r_2 \in q_2} \min_{r_1 \in q_1} d(r_1, r_2) \,\}$$

Dans notre cas, q_1 et q_2 sont des requêtes, c'est-à-dire, des ensembles de références, r_1 et r_2 sont des références et la distance d utilisée est d_G, définie ci-dessous.

5.1.2 Distance entre références

Afin de calculer une distance entre références, dans Giacometti et al. (2008), nous avons considéré la distance de Hamming [Hamming (1950)]. Cette distance a pour qualité sa simplicité d'utilisation et de compréhension. Cependant, elle ne permet pas de prendre en compte certaines caractéristiques d'OLAP, essentiellement, les hiérarchies.

En effet, deux membres à comparer peuvent appartenir à des niveaux différents d'une même hiérarchie. Or, nous pensons que deux membres issus du même niveau sont plus proches que deux membres issus de niveaux différents de la même hiérarchie. Par conséquent, il s'agissait de trouver une méthode permettant de comparer des références en prenant en compte leur différence de niveau dans une hiérarchie. Une solution possible est la résolution du problème classique du plus court chemin.

Concept issu de la théorie des graphes, le problème du plus court chemin, adapté à notre problématique, consiste à trouver la longueur du chemin le plus court entre deux noeuds du graphe en terme de nombre de noeuds. De nombreuses propositions de résolution de ce problème ont été présentées, citons notamment les algorithmes de Dijkstra [Dijkstra (1971)], de Bellman-Ford [Bellman (1947)], A^* [Hart et al. (1968)] et de Floyd-Warshall [Floyd (1962)].

En OLAP, une hiérarchie est vue comme un arbre. Or, un arbre est un graphe particulier, à savoir, un graphe non-orienté connexe sans cycle. Par conséquent, nous pouvons rechercher le plus court chemin entre deux membres d'une hiérarchie.

La longueur du plus court chemin entre deux membres est une distance $d_{m_G} : Membre \times Membre \rightarrow \mathbb{R}^+$, au sens mathématique, puisqu'elle vérifie les propriétés suivantes :

- positivité : $\forall x, y \in Membre, d_{m_G}(x, y) \geq 0$, nous comptons des noeuds,
- symétrie : $\forall x, y \in Membre, d_{m_G}(x, y) = d_{m_G}(y, x)$, le plus court chemin de x à y est égal au plus court chemin de y à x,
- séparation : $\forall x, y \in Membre, d_{m_G}(x, y) = 0 \Leftrightarrow x = y$, si la taille du chemin est égale à 0, cela signifie que le chemin ne contient pas de noeud, donc $x = y$,
- inégalité triangulaire : $\forall x, y, z \in Membre, d_{m_G}(x, z) \leq d_{m_G}(x, y) + d_{m_G}(y, z)$, par définition, le plus court chemin est le plus court.

Nous avons donc :

Définition 5.2 *La distance d_{m_G} entre deux membres m_1 et m_2 d'une hiérarchie est la longueur minimale du plus court chemin entre m_1 et m_2 et se note : $d_{m_G}(m_1, m_2)$.*

Dorénavant, nous sommes capables de comparer des membres grâce à la distance d_{m_G} appliquée à deux membres d'une même hiérarchie. Or, une référence est un tuple de membres répartis par dimension. Afin d'obtenir une distance entre références, nous proposons d'additionner les distances entre membres d_{m_G}, ce qui nous permet également de garder les propriétés de distance mathématique. Nous obtenons la définition suivante :

Définition 5.3 *Soit C un cube de dimension n et soit $r_1 = \langle r_1^1, ..., r_1^n \rangle$ et $r_2 = \langle r_2^1, ..., r_2^n \rangle$ deux références telles que $r_j^i \in adom(D_i)$, pour $i \in [1, n]$ et $j \in [1, 2]$, la distance entre références est :*

$$d_G(r_1, r_2) = \sum_{i=1}^{n} d_{m_G}(r_1^i, r_2^i)$$

Exemple : Distance entre références
La distance d_G entre les références $r_1^2 = \langle P_1, Blois, 2008 \rangle$ et $r_2^1 = \langle All, Centre, 2008 \rangle$ vaut :

$$\begin{aligned} d_G(r_1^2, r_2^1) &= d_{m_G}(P_1, All) + d_{m_G}(Blois, Centre) + d_{m_G}(2008, 2008) \\ &= 2 + 2 + 0 \\ &= 4 \end{aligned}$$

5.1.3 Retour sur la distance entre requêtes

La distance entre requêtes que nous utilisons, à savoir, la distance de Hausdorff (Définition 5.1), nécessite une distance entre références comme celle que nous venons de définir (Définition 5.3) basée sur la longueur du plus court chemin entre deux membres. Nous proposons donc l'utilisation de la distance de Hausdorff couplée avec notre distance entre références. Nous obtenons :

$$\begin{aligned} d_{\mathrm{H}}(q_1, q_2) = \max\{ & \max_{r_1 \in q_1} \min_{r_2 \in q_2} d_G(r_1, r_2), \\ & \max_{r_2 \in q_2} \min_{r_1 \in q_1} d_G(r_1, r_2) \} \end{aligned}$$

Exemple : Distance entre requêtes
La distance de Hausdorff d_H entre les requêtes $q_1 = \{r_1^1, r_1^2\}$ et $q_2 = \{r_2^1\}$ vaut :

$$\begin{aligned} d_H(q_1, q_2) &= max\{max\{min\{d_G(r_1^1, r_2^1)\}, min\{d_G(r_1^2, r_2^1)\}\}, \\ &\quad max\{min\{d_G(r_2^1, r_1^1), d_G(r_2^1, r_1^2)\}\}\} \\ &= max\{max\{min\{6\}, min\{4\}\}, max\{min\{6, 4\}\}\} \\ &= max\{max\{6, 4\}, max\{4\}\} \\ &= max\{6, 4\} \\ &= 6 \end{aligned}$$

5.1.4 Partitionnement de requêtes et classification

Le partitionnement de l'ensemble des requêtes peut être fait en utilisant un algorithme de clustering simple comme les K-medoids proposé par Kaufmann et Rousseeuw (1987). Dans ce cas, le classificateur de requête associe la requête à la classe pour laquelle le représentant de classe est le plus proche de cette requête, au sens de la distance entre requêtes utilisée. Cette étape nous permet d'obtenir des sessions généralisées : chaque requête de la session est remplacée par la classe à laquelle elle appartient. Nous obtenons ainsi des sessions d'étiquettes de classes de requêtes.

5.2 Étape 2 : Prédiction - Génération des recommandations candidates

Nous utilisons les mêmes paramètres que dans l'instanciation présentée dans Giacometti et al. (2008).

5.2.1 Match

Grâce à l'étape précédente de partitionnement/classification, nous avons obtenu des sessions généralisées et une session généralisée courante. L'étape *Match* permet de trouver un ensemble de sessions généralisées qui contiennent la session généralisée courante (Figure 1 : Partie 2 : Prédiction, Gauche : Match).
Le paramètre choisi est *Approximate String Matching* [Navarro (2001)].
La méthode d'*Approximate String Matching* utilisée ici se limite à supprimer le premier élément de la séquence si une correspondance exacte n'existe pas, et ainsi de suite jusqu'à ce que la séquence ne contienne plus d'élément. Ce mode opératoire est utilisable dans notre cas puisque les sessions généralisées sont vues comme des séquences d'étiquettes et que l'*Approximate String Matching* manipule des séquences de lettres.

5.2.2 Predict

L'étape *Predict* permet d'obtenir, pour un ensemble de sessions généralisées donné, un ensemble de classes candidates (Figure 1 : Partie 2 : Prédiction, Centre : Predict).
Le paramètre choisi est *Successeur*[4].
La méthode utilisée consiste à renvoyer la classe qui suit la dernière classe de la session généralisée courante, dans chaque session généralisée renvoyée par l'étape *Match*.

5.2.3 ClassRep

L'étape *Predict* renvoie un ensemble de classes candidates à la recommandation, or, nous souhaitons proposer un ensemble de requêtes candidates. Par conséquent, l'étape *ClassRep* consiste à extraire un représentant de chaque classe renvoyée (Figure 1 : Partie 2 : Prédiction, Droite : ClassRep).
Le paramètre choisi est *Médoïde de la classe*.
Comme la méthode de partitionnement utilisée est l'algorithme des *K-medoids*, le représentant de classe choisi est le médoïde de chaque classe.

5.3 Étape 3 : Ordonnancement des recommandations candidates

L'étape précédente permet d'obtenir des requêtes candidates à la recommandation. Cet ensemble de requêtes candidates prédit, pour une session courante donnée, sera le même quel que soit l'utilisateur. Mais, nous souhaitons donner une allure plus personnelle à ces recommandations. Il va s'agir de ranger les requêtes candidates en fonction des préférences de l'utilisateur

[4]Notons qu'ici encore, le paramètre pourrait être différent. Par exemple, nous pourrions choisir *Dernière* et la méthode utilisée consisterai à renvoyer la dernière classe de chaque session généralisée renvoyée par l'étape *Match*. Puisque, dans certains cas, l'utilisateur préférera sans doute "sauter" les requêtes intermédiaires pour aller directement aux parties du cube les plus intéressantes.

(c'est-à-dire en fonction de leur adéquation au profil utilisateur). Cette étape de personnalisation va permettre de proposer un ordre de requêtes candidates différent selon l'utilisateur.

Inspirés par les travaux de Bellatreche et al. (2006), nous proposons un algorithme qui permet de comparer des requêtes par rapport au profil utilisateur. En effet, les auteurs proposent une définition pour savoir si une requête est plus intéressante qu'une autre :

Définition 5.4 *Soit un profil utilisateur Γ sur un cube C, soit q et q' deux requêtes, q est moins intéressante que q', noté $q \preceq_\Gamma q'$, si :*

$$(\forall r \in ref(q))(\exists r' \in ref(q'))(r \leq_\Gamma r')$$

où $ref(q)$ (respectivement $ref(q')$) représentent l'ensemble des références de la requête q (respectivement q').

Ce qui revient à ordonner les requêtes en fonction des références préférées. Une solution possible pour réaliser un tel ordonnancement est la suivante :

1. Pour chaque requête q_i de l'ensemble Q_R des recommandations, trier ces références en fonction de l'ordre $\leq_\Gamma$.
2. Pour chaque ensemble trié de références, prendre le premier élément, c'est-à-dire la référence préférée que nous appelons le maximum.
3. Trier ces maximums en fonction de l'ordre $\leq_\Gamma$.
4. Retourner l'ensemble ordonné Q_R^* des requêtes tel que chaque maximum a été remplacé par la requête qui lui correspond.

Finalement, l'ensemble Q_R^* des requêtes candidates ordonnées en fonction du profil utilisateur est recommandé à l'utilisateur.

5.4 Recommandation par défaut

Nous sommes conscients que l'ensemble de recommandations candidates peut être vide. Dans ce cas, il pourrait être utile de toujours pouvoir fournir à l'utilisateur une recommandation. Diverses recommandations *par défaut* peuvent être proposées à l'utilisateur.

Nous proposons, par exemple, d'emprunter une idée de Kleinberg (1999) qui consiste à calculer la popularité (*"hub"*) et l'autorité (*"authority"*) d'une page web. En effet, le web est vu comme un immense graphe où un noeud est une page web et les arcs sont les liens entrants et sortants entre deux pages. Ainsi, une page qui pointe vers de nombreuses bonnes *authorities* est un bon *hub* et une page pointée par de nombreux bons *hubs* est une bonne *authority*.
Sur le même principe, notre ensemble de sessions généralisées contenues dans le log peut être vu comme un graphe où chaque noeud est une classe de requêtes et les arcs sont les enchaînements de classes de chaque session généralisée. Il nous est ainsi possible de proposer comme recommandation par défaut, le représentant de la classe faisant autorité (respectivement, la classe centrale (*hub*)), c'est-à-dire, la classe qui a le nombre le plus élevé de successeurs (respectivement, prédécesseurs).

6 Expérimentations

Dans cette section, nous présentons les résultats des expériences que nous avons menées pour évaluer les capacités de notre système. Nous utilisons des données synthétiques produites grâce à notre propre générateur.

Le prototype de recommandation de requêtes ainsi que le générateur sont développés en Java utilisant JRE 1.6.0_13. Tous les tests ont été réalisés avec un Core 2 Duo - E4600 avec 4GB de RAM sous Linux CentOS5.

6.1 Génération des logs de requêtes

Nous utilisons notre générateur pour générer un ensemble de sessions sur la base de données FoodMart fournie avec le moteur OLAP Mondrian [Pentaho Corporation (2009)].

Pour générer un log, notre générateur utilise les paramètres suivants : un nombre (X) de sessions dans le log et un nombre (Y) de requêtes par session. Dans nos tests, nous avons limité le nombre maximum de références par requête à 100 puisqu'il est raisonnable de considérer que les utilisateurs ne produiront pas de tableaux croisés plus grands que 10×10 comme réponse à une requête MDX.

Dans une session, les requêtes sont générées de la manière suivante : une première requête est sélectionnée aléatoirement parmi les exemples de requêtes fournies par Mondrian. Puis, chaque requête suivante est générée en choisissant aléatoirement une dimension et en modifiant l'ensemble des membres de cette dimension de la requête précédente, pour simuler le comportement d'un utilisateur.

6.2 Analyse de performance

Cette expérience permet d'évaluer le temps nécessaire à la génération d'une recommandation pour différentes tailles de log. Les résultats sont présentés Figure 2.

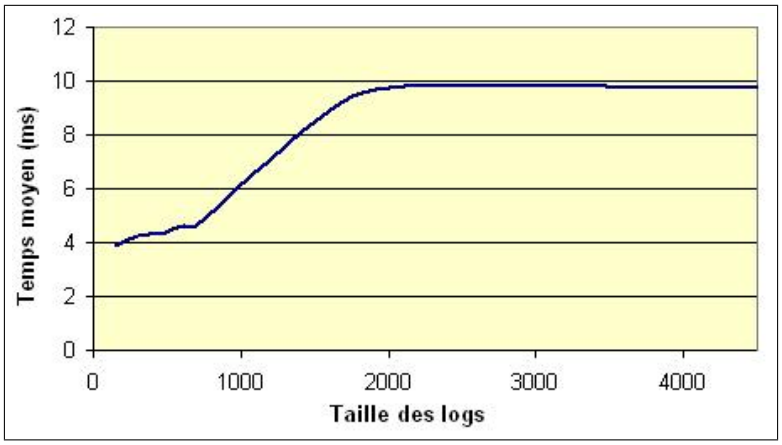

FIG. 2 – *Analyse de performance*

Les tailles de logs sont obtenues en multipliant les paramètres X (nombre de sessions) et Y (nombre maximum de requêtes par session). X varie entre 25 et 500 et Y entre 10 et 20. Nous obtenons ainsi des tailles de logs variant entre 150 et 4500 requêtes.

Notons que ce qui est mesuré ici est le temps nécessaire au calcul de la session généralisée courante et aux étapes de prédiction et de génération des recommandations candidates. Le temps nécessaire à la classification, au calcul de l'ensemble des sessions généralisées et à l'ordonnancement n'est pas pris en compte ici.

La figure 2 montre que le temps nécessaire pour générer une recommandation augmente avec la taille des logs mais reste tout de même acceptable (inférieur à 10 ms).

7 Conclusion

Dans cet article, nous proposons une instanciation particulière du cadre générique proposé par Giacometti et al. (2008) pour recommander des requêtes MDX. Ce cadre est générique dans le sens où il peut être instancié pour changer la manière dont les recommandations sont calculées. Nous donnons donc une instanciation particulière qui nous permet de proposer des recommandations en adéquation avec le profil de l'utilisateur en utilisant des paramètres qui prennent en compte certaines particularités d'OLAP, à savoir les hiérarchies.

Nos travaux futurs incluent :

- Incorporer les algorithmes de personnalisation de Bellatreche et al. (2006) dans notre prototype existant de recommandation.
- Mener des expériences sur des données réelles afin d'améliorer la qualité des requêtes recommandées, surtout pour l'évaluation des diverses instanciations de notre cadre et ainsi déterminer à quel contexte elles sont le mieux adaptées. À cet effet, nous recherchons des "feedbacks" d'utilisateurs.
- Rechercher d'autres instanciations de notre cadre proposant des requêtes recommandées de meilleure qualité. Par exemple, en personnalisant les requêtes au plus tôt dans la démarche, ...
- Étendre notre définition de requête pour prendre en compte une plus grande partie du langage MDX.

Références

Adomavicius, G. et A. Tuzhilin (2005). Toward the next generation of recommender systems : A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng. 17*(6), 734–749.

Baeza-Yates, R. A., C. A. Hurtado, et M. Mendoza (2004). Query recommendation using query logs in search engines. In *EDBT Workshops*, pp. 588–596.

Bellatreche, L., A. Giacometti, P. Marcel, et H. Mouloudi (2006). Personalization of mdx queries. In *BDA*.

Bellatreche, L., A. Giacometti, P. Marcel, H. Mouloudi, et D. Laurent (2005). A personalization framework for olap queries. In *DOLAP*, pp. 9–18.

Bellman, R. (1947). *Dynamic programming*. Princeton, New Jersey : Princeton University Press. XXV, 342 p.

Dijkstra, E. W. (1971). A short introduction to the art of programming. circulated privately.

Floyd, R. W. (1962). Algorithm 97 : Shortest path. *Commun. ACM 5*(6), 345.

Giacometti, A., P. Marcel, et E. Negre (2008). A framework for recommending olap queries. In *DOLAP*, pp. 73–80.

Hamming, R. W. (1950). Error detecting and error correcting codes. *Syst. Tech. J. 29*, 147–160.

Hart, P., N. Nilsson, et B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 100–107.

Hausdorff, F. (1914). *Grundzüge der Mengenlehre*. von Veit.

Kaufmann, L. et P. J. Rousseeuw (1987). Clustering by means of medoids. In Y. Dodge (Ed.), *Statistical Data Analysis based on the L1 Norm*, pp. 405–416. Amsterdam : Elsevier/North Holland.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM 46*(5), 604–632.

Matheron, G. (1975). *Random Sets and Integral Geometry*. John Wiley and sons.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Comput. Surv. 33*(1), 31–88.

Pentaho Corporation (2009). Mondrian open source OLAP engine. Available at http ://mondrian.pentaho.org/.

Sapia, C. (1999). On modeling and predicting query behavior in OLAP systems. In *DMDW*, pp. 2.1–2.10.

Sapia, C. (2000). Promise : Predicting query behavior to enable predictive caching strategies for OLAP systems. In *DaWaK*, pp. 224–233.

Sarawagi, S. (2000). User-adaptive exploration of multidimensional data. In *VLDB*, pp. 307–316.

Srivastava, J., R. Cooley, M. Deshpande, et P.-N. Tan (2000). Web usage mining : Discovery and applications of usage patterns from web data. *SIGKDD Explorations 1*(2), 12–23.

Summary

An OLAP analysis session can be defined as an interactive session during which a user launches queries to navigate within a cube. Very often choosing which part of the cube to navigate further, and thus designing the forthcoming query, is a difficult task. In this paper, we propose to use what the OLAP users did during their former exploration of the cube as a basis for recommending MDX queries to the user.