Quand la recommandation rencontre la personnalisation

Ou comment générer des recommandations (requêtes MDX) en adéquation avec les préférences de l'utilisateur

Elsa Negre

Laboratoire d'Informatique (EA-2101),
Université François Rabelais Tours,
Antenne Universitaire de Blois, 3 place Jean Jaurès, F-41000 Blois
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.*

ABSTRACT. *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.*

MOTS-CLÉS : *Recommandations, OLAP, requêtes MDX, Personnalisation, Préférences*

KEYWORDS: *Recommendations, OLAP, MDX queries, Personalization, Preferences*

RSTI - TSI – 30/2011. Entrepôt de données, pages 933 à 952

1. Introduction

La recommandation de requêtes dans les bases de données est un axe de recherche prometteur (Khoussainova *et al.*, 2009; Chatzopoulou *et al.*, 2009), en particulier dans les systèmes OLAP où l'utilisateur navigue interactivement dans un cube en lançant une séquence de requêtes (et plus particulièrement des requêtes MDX[1] (Microsoft Corporation, 1998)) 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).

Les systèmes de recommandation existants sont généralement classés selon deux catégories : les méthodes basées sur le contenu et celles basées sur le filtrage collaboratif (Adomavicius *et al.*, 2005). Les méthodes basées sur le contenu recommandent à l'utilisateur des objets similaires à ceux qui l'ont intéressé dans le passé, tandis que les méthodes basées sur le filtrage collaboratif recommandent des objets qui ont interessé des utilisateurs similaires. Au vu de la nature multi-utilisateur des entrepôts de données, dans nos travaux précédents (Giacometti *et al.*, 2008; Giacometti *et al.*, 2009a; Giacometti *et al.*, 2009b; Negre, 2009), nous avons proposé des techniques issues du filtrage collaboratif pour recommander des requêtes OLAP à l'utilisateur. L'idée principale de (Giacometti *et al.*, 2008; Giacometti *et al.*, 2009a; Negre, 2009) est de calculer une similarité entre la séquence de requêtes de l'utilisateur courant et les séquences de requêtes précédentes qui ont été enregistrées par le serveur. Dans (Giacometti *et al.*, 2009b), nous avons étendu ce principe afin de prendre en compte ce que les utilisateurs cherchaient, c'est-à-dire les mesures.

Nous présentons, dans cet article, une instanciation particulière du cadre générique de génération de recommandations élaboré lors de nos précédents travaux, en faisant un lien avec la personnalisation. Notre cadre (Giacometti *et al.*, 2008) 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.

Notre cadre générique est fondé sur le processus suivant : (i) prétraitement du log, (ii) génération 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, (iii) ordonnancement des 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 prétraitement 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 sous-jacent à cette proposition d'un cadre générique est que les recommandations dépendent fortement des utilisateurs et des données sur lesquelles les recommandations

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.

sont calculées. Adomavicius *et al.* (2005) expliquent 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 : $ClusterSP_{Perso}$. 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 du problème de la recommandation de requêtes OLAP (et particulièrement de requêtes MDX (Negre, 2009)) 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'information (Adomavicius *et al.*, 2005), et dans l'exploitation des usages du web (*Web Usage Mining*) (Srivastava *et al.*, 2000). Par exemple, Baeza-Yates *et al.* (2004) emploient l'algorithme des k-means pour grouper des requêtes soumises à un moteur de recherche, génèrent des recommandations candidates et ordonnent les candidats.

Notre contribution est d'adapter ces techniques existantes à OLAP. Dans le domaine OLAP, les travaux de Sapia (1999), Chatzopoulou *et al.* (2009) et Yang *et al.* (2009) ont un point commun avec notre travail : prévoir la prochaine requête OLAP. Néanmoins, (i) la principale préoccupation de Sapia (1999) est de prétraiter les données mais pas de recommander une requête, (ii) Sapia (1999) 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) et la distance que nous utilisons prend en compte les membres[2]), (iii) Sapia (1999) utilise un modèle de Markov pour prévoir la prochaine requête, tandis que nous avons choisi de ne pas utiliser un modèle probabiliste, (iv) Chatzopoulou *et al.* (2009) et Yang *et al.* (2009) ne tiennent pas compte du séquencement des requêtes.

Jerbi *et al.* (2009) et Bentayeb *et al.* (2009) proposent des méthodes de personnalisation vues comme des méthodes de recommandation qui prennent en compte les préférences de l'utilisateur mais ne s'intéressent pas aux autres analystes qui peuvent avoir des ressemblances avec l'utilisateur courant, et encore moins au séquencement de requêtes lors de la session d'analyse puisqu'elles se limitent à la dernière requête posée. Notons cependant que Bentayeb *et al.* (2009) utilisent l'algorithme de classification non supervisée (*clustering*) des *K-means* dans leur phase de recommandation.

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.

Dans (Giacometti *et al.*, 2008; Giacometti *et al.*, 2009a), nous supposons qu'un log de requêtes est utilisé pour rechercher les similarités entre la session courante et les sessions précédentes afin d'extraire au moins une requête à recommander. Les deux méthodes diffèrent par la manière dont elles considèrent la similarité entre sessions et/ou requêtes. Dans (Giacometti *et al.*, 2009a), nous utilisons la distance de Levenshtein (entre sessions) et la distance de Hausdorff (entre requêtes). Dans (Giacometti *et al.*, 2008), nous groupons les requêtes et détectons si la session courante est un préfixe d'une session existante. Ces méthodes diffèrent aussi par leur manière de prédire les requêtes recommandées. Giacometti *et al.* (2008) identifient une position où la concordance existe et recommandent la requête située après la position. Tandis que Giacometti *et al.* (2009a) recommandent la dernière requête de la session la plus proche.

3. Exemple

Dans cette section, nous illustrons avec un exemple simple, détaillé figure 1, le principe général de notre approche.

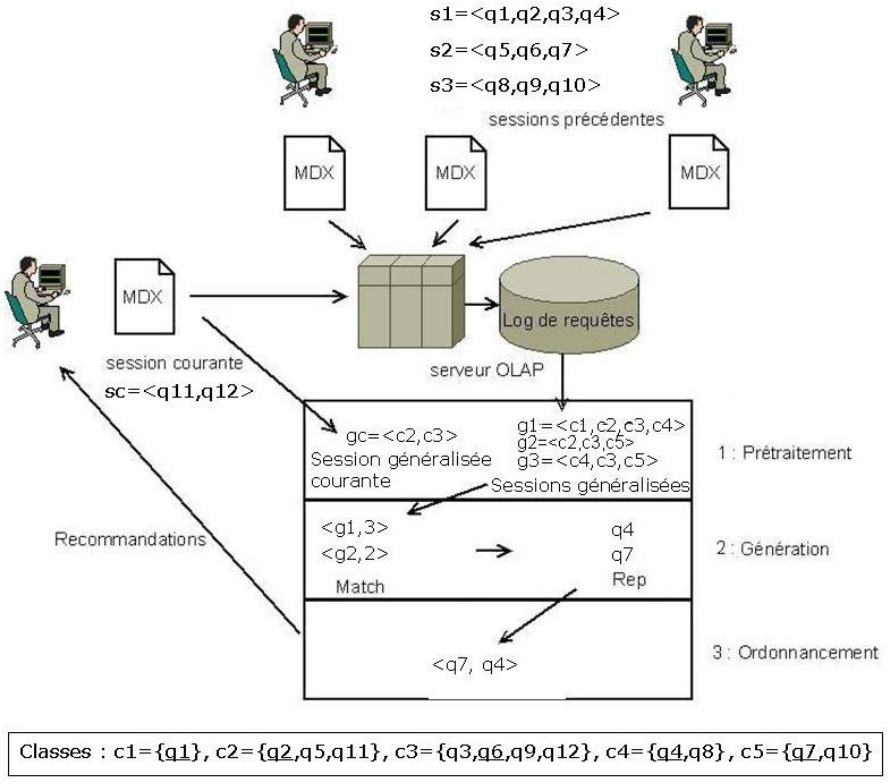

Figure 1. *Exemple de fonctionnement de notre système de recommandation*

Considérons un serveur OLAP utilisé par plusieurs utilisateurs. Chaque utilisateur navigue dans le cube de données en lançant une séquence de requêtes (ou session). Le serveur enregistre ces sessions dans un log de requêtes. 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 ($\forall i \in [1, 10]$) sont des requêtes MDX.

Supposons maintenant qu'un utilisateur courant lance une nouvelle session, appelée *session courante*, telle que $s_c = \langle q_{11}, q_{12} \rangle$ où q_{11}, q_{12} sont des requêtes MDX. Cet utilisateur courant pourrait être intéressé par la façon dont les autres utilisateurs ont exploré le cube et utiliser cette information pour concevoir sa prochaine requête.

Le nombre d'utilisateurs ainsi que le nombre de sessions peuvent être très importants. Ainsi, le log peut être très volumineux, mais pas autant que le cube lui-même. En outre, les divers utilisateurs ont des intérêts différents et leurs requêtes naviguent 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 sont groupées dans des classes de requêtes où chaque classe est un ensemble de requêtes proches les unes des autres. Puis, pour chaque session du log et pour la session courante, chaque requête est remplacée par la classe à laquelle elle appartient, nous obtenons des *sessions généralisées*. Dans notre exemple, les trois sessions généralisées correspondant aux trois sessions du log[3] 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 c_2, q_3, q_6, q_9 appartiennent à c_3, ... et la *session généralisée courante* est $g_c = \langle c_2, c_3 \rangle$ (figure 1 : Prétraitement).

La session généralisée courante g_c est une sous-séquence de g_1 à la position 3 et de g_2 à la position 2. Ainsi, g_1, g_2 sont des sessions candidates (figure 1 : Génération). Supposons que les requêtes intéressantes soient celles qui font suite, dans les *sessions candidates*, à la position où la concordance a été trouvée, c'est-à-dire à la position $p + 1$. Ici, de telles requêtes appartiennent aux classes c_4 et ${c_5}^4$, appelées les *classes candidates*. 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. Supposons que pour la classe c_4 se soit la requête q_4 et q_7 pour c_5. q_4, q_7 sont les *requêtes candidates*.

Enfin, ces requêtes candidates sont *ordonnées* en fonction de leur adéquation avec le profil de l'utilisateur (figure 1 : 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, les techniques de prétraitement du log,

3. Le séquencement des classes est le même que le séquencement initial des requêtes.
4. Par exemple, g_1 coïncide avec g_c à la position 3. La classe située à la position 4 dans g_1 est c_4.

de génération des recommandations et d'ordonnancement sont laissées comme paramètres. Diverses instanciations du cadre sont possibles (Negre, 2009). L'une d'entre elles, $ClusterSP_{Perso}$, 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 sorte $sort(R) = \{A_1, \ldots, A_N\}$. Nous notons $adom(A_i)$ le domaine actif de chaque attribut $A_i \in sort(R)$.

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 sorte $sort(D_i) = \{L_i^0, \ldots, L_i^{d_i}\}$; 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, c'est-à-dire une instance de relation de sorte $sort(F) = \{L_1^0, \ldots, L_N^0, m\}$ où m est un attribut numérique (une mesure).

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*[5], 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 $\bigcup_{i\in[1,N]} adom(D_i)$.

Distance entre membres Soit un cube $C = \langle D_1, ..., D_N, F\rangle$ et soit les membres de la dimension D_i, les valeurs de la table D_i, notés $adom(D_i)$, une distance entre membres est une fonction de $adom(D_i) \times adom(D_i)$ vers l'ensemble des nombres réels.

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 $i \in$

5. Dimension plate.

$[1, N]$. De plus, soit un cube C, nous notons par $ref(C)$, l'ensemble de toutes les références de C.

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 $R_1 \times \ldots \times R_N$. Enfin, 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 positifs.

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. $ClusterSP_{Perso}$

Nous présentons dans cette section une instanciation particulière de notre cadre générique (Giacometti *et al.*, 2008). Notons que, 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 :

– $sort($PRODUIT$) = \{IdProd, Type, All\}$,

– $sort($LOCALISATION$) = \{Ville, Departement, Region, Pays, All\}$,

– $sort($TEMPS$) = \{Jour, Mois, Annee, All\}$,

– $sort($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
$= \{\langle All, Centre, 2008\rangle\} = \{r_2^1\}$

5.1. *Distances*

5.1.1. *Distance entre requêtes*

Comme les requêtes sont vues comme des ensembles de références, comparer deux requêtes revient à comparer deux ensembles de références. Ainsi, la distance de Hausdorff (1914) pour comparer deux ensembles basée sur une distance entre les éléments de l'ensemble peut être utilisée. De manière informelle, deux ensembles sont proches si chaque élément de l'un des ensembles est proche des éléments de l'autre ensemble.

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-après.

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 (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 nœuds du graphe en termes de nombre de nœuds. De nombreuses propositions de résolution de ce problème ont été présentées, citons notamment les algorithmes de Dijkstra (1971), de Bellman (1947), A^* (Hart *et al.*, 1968) et de 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 nœuds,

– 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 nœud, 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.

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 5.3.

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 :

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

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 :

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

5.2. *Prétraitement*

Comme indiqué section 3, pour pallier le problème de volume et de dispersion du log de requêtes, celui-ci peut être prétraité grâce à une méthode de partitionnement. Ainsi, l'ensemble des requêtes peut être partitionné en utilisant un algorithme de clustering simple comme les K-medoids proposé par Kaufmann *et al.* (1987). Dans ce cas, chaque requête est associée 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 (des séquences) d'étiquettes de classes de requêtes (figure 1 : Prétraitement).

5.3. *Génération des recommandations candidates*

5.3.1. *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. Le paramètre choisi est *Approximate String Matching*[6] (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. Cette méthode renvoie pour une session généralisée donnée, la position où la meilleure concordance est trouvée (figure 1 : Génération - Match).

6. L'*Approximate String Matching* est le problème qui consiste à trouver dans un texte où une chaîne de caractères donnée apparaît en permettant un nombre limité d'erreur de concordance.

5.3.2. *Rep*

L'étape *Rep* permet d'obtenir, pour un ensemble de sessions généralisées donné, un ensemble de requêtes candidates. Le paramètre choisi est *Médoïde du Successeur*[7]. La méthode utilisée consiste à renvoyer le médoïde[8] de la $(p+1)^{ème}$ classe de la session généralisée courante, dans chaque session généralisée renvoyée par l'étape *Match* (figure 1 : Génération - Rep).

5.4. *Ordonnancement des recommandations candidates*

L'étape précédente permet d'obtenir des requêtes candidates à la recommandation. Cet ensemble prédit de requêtes candidates, 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 (c'est-à-dire en fonction de leur adéquation au profil de l'utilisateur). Cette étape de personnalisation va permettre de proposer un ordre de requêtes candidates différent selon l'utilisateur (figure 1 : Ordonnancement).

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;

---

7. Notons qu'ici encore, le paramètre pourrait être différent. Par exemple, nous pourrions choisir *Médoïde du Dernier* et la méthode utilisée consisterait à renvoyer le médoïde de 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.

8. Comme la méthode de partitionnement utilisée est l'algorithme des *K-medoids*, le représentant de classe choisi est le médoïde de chaque classe.

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.

5.5. *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 nœud 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 nœud 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 n'avons pas trouvé de données réelles adaptées à notre cadre d'étude[9]. Par conséquent, nous utilisons des données synthétiques produites avec notre propre générateur de données. Pour cela, nous générons des ensembles de sessions sur la base de données FoodMart fournie avec le moteur OLAP Mondrian (Pentaho Corporation, 2009).

Notre générateur utilise les paramètres suivants : un nombre (X) de sessions dans le log, un nombre maximal (Y) de requêtes par session, un seuil (β) de comparaison de cellules. Tous ces paramètres sont définis par l'utilisateur.

Chaque session est générée de la manière suivante :

Pour chacune des X sessions :

1) création de la première requête MDX de la session en tirant aléatoirement deux dimensions et en sélectionnant tous les fils du membre situé au niveau le plus haut de la hiérarchie de chacune des deux dimensions.

Puis les $Y - 1$ requêtes suivantes sont obtenues ainsi :

2) détection dans la requête précédente des paires de cellules provoquant des différences ou des exceptions avec un seuil β *via* les opérations proposées par Sarawagi *et al.* (1998) à savoir, $DIFF$ et $RELAX$, et obtention d'un ensemble de membres pouvant expliquer les anomalies constatées,

3) modification de la requête précédente en faisant apparaître pour la dimension de chaque membre explicatif, soit tous les membres du même niveau que le membre explicatif soit le membre lui-même, opérant ainsi un changement de niveau sur les dimensions déjà affichées et un ajout de membre(s) pour les dimensions non encore affichées;

4) dans le cas où aucune explication n'a été trouvée, c'est-à-dire que l'ensemble de membres explicatifs est vide, modification de la requête précédente en tirant aléatoirement une dimension parmi celles qui ne sont pas encore affichées et en ajoutant tous les fils du membre situé au niveau le plus haut de la hiérarchie de cette dimension.

Notons que nos logs ont été générés avec un seuil β fixé à 1.01 afin d'obtenir presque à chaque fois, des paires de cellules intéressantes. De plus, dans nos tests, nous avons fixé le nombre de références à 300 puisqu'il est raisonnable de considérer que les utilisateurs produiront rarement des tableaux croisés dont la taille excédera 15×20 cellules comme réponse à une requête MDX.

Notre première expérience est une analyse de performance et consiste à évaluer l'efficacité de l'instanciation en termes de temps nécessaire à la proposition d'une

9. En effet, nous avons besoin du log de requêtes MDX ainsi que du schéma et de l'instance de la base de données.

recommandation. La seconde, quant à elle, est une analyse de rappel/précision et consiste à évaluer l'instanciation en termes de rappel, précision et F-mesure.

6.2. *Résultats*

Nous présentons ici les expériences que nous avons menées sur l'instanciation, $ClusterSP_{Perso}$, de notre système de recommandation présentée dans cet article. Notons cependant que des résultats comparatifs avec d'autres instanciations du même système sont proposés dans (Negre, 2009).

6.2.1. *Analyse de performance*

Le temps est calculé pour différentes tailles de log (en termes de nombre de requêtes). Les résultats sont présentés figure 2 en fonction des différentes tailles de log. Les tailles de log sont obtenues en multipliant les paramètres X (nombre de sessions) et Y (nombre maximal de requêtes par session). X varie entre 25 et 600 et Y vaut 20. Nous obtenons donc des logs dont la taille varie entre 100 et 12 000 requêtes. La meilleure recommandation est calculée pour chacun de ces logs, pour des sessions courantes SC de tailles variées (le nombre de requêtes par session est égal à 1, 5 ou 10), générées elles aussi par notre générateur de sessions.

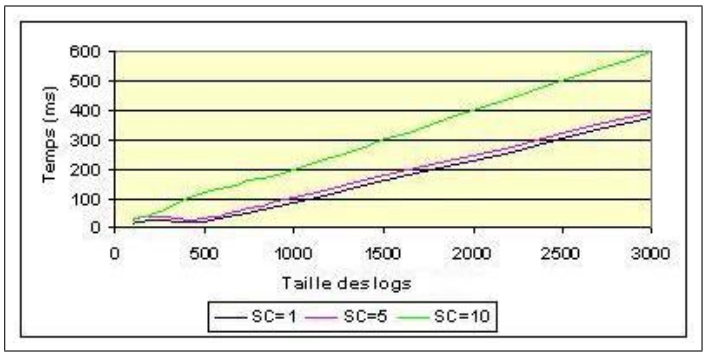

Figure 2. *Analyse de performance*

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.

La figure 2 montre que le temps nécessaire pour générer une recommandation augmente linéairement avec la taille des logs mais reste acceptable. Nous remarquons que ce temps est légèrement influencé par la taille de la session courante, en effet, plus la session courante est longue, plus le temps de génération augmente. Notons

cependant que ce temps ne dépasse pas la seconde pour générer une recommandation à la suite d'une session courante dont la taille peut atteindre 10 requêtes.

6.2.2. *Rappel/précision*

Nous présentons ici, la seconde expérimentation que nous avons menée, dans laquelle nous utilisons une *10-fold cross validation* pour évaluer notre méthode, dans le même esprit que la validation expérimentale réalisée dans (Chatzopoulou *et al.*, 2009).

L'ensemble généré de sessions est partitionné en dix sous-ensembles de taille égale et, à chaque itération, neuf sous-ensembles sont utilisés en tant que log et chaque session du sous-ensemble restant est utilisée comme support pour la session courante. Plus précisément, pour chacune de ces sessions (du dixième sous-ensemble) s_c de taille n, nous utilisons la séquence des $n-1$ premières requêtes comme session courante et nous calculons les recommandations pour la $n^{ième}$ requête. La $n^{ième}$ requête de s_c est appelée la *requête attendue*.

Nous évaluons le rappel et la précision (Baeza-Yates *et al.*, 1999) des recommandations en utilisant les métriques suivantes :

– $Précision = \frac{|membres(q_{at}) \cap membres(q_{rec})|}{|membres(q_{rec})|}$,

– $Rappel = \frac{|membres(q_{at}) \cap membres(q_{rec})|}{|membres(q_{at})|}$.

où $membres(q)$ est l'ensemble des membres de la requête q, q_{rec} est une requête recommandée et q_{at} est la requête attendue. Pour chaque session, nous calculons le rappel maximum parmi toutes les requêtes recommandées et la précision pour la requête obtenant le rappel maximum. Les logs générés pour ces tests ont pour taille : 120 requêtes (25 sessions) pour les logs de petite taille, 505 requêtes (100 sessions) pour les logs de taille moyenne et 979 requêtes (200 sessions) pour les logs de grande taille.

La figure 3 montre l'inverse de la distribution de fréquences cumulées (inverse CFD) des valeurs enregistrées de rappel, précision et F-mesure[10] pour les sessions. Un point (x, y) de ces différents graphiques signifie que x % des sessions ont une précision ou un rappel ou une F-mesure $\geq y$.

La figure 3 montre l'efficacité de notre méthode : la précision atteint 0.8 pour plus de 65 % des sessions avec un rappel d'environ 0.7 et une F-mesure de 0.7 dans le cas de logs de grande taille alors que la précision atteint 0.8 pour environ 25 % des sessions avec un rappel d'environ 0.6 et une F-mesure de 0.65 dans le cas de logs de petite taille. Ainsi, la qualité des recommandations générées augmente en fonction de la taille des logs. En effet, plus le log est de grande taille, meilleure est la qualité. Ce résultat était attendu puisque, plus le log est de grande taille, plus le nombre de possibilités pour proposer une recommandation est grand.

10. La F-mesure (van Rijsbergen, 1979), $F = \frac{2.(\text{Précision} \cdot \text{Rappel})}{(\text{Précision} + \text{Rappel})}$, est une mesure de performance.

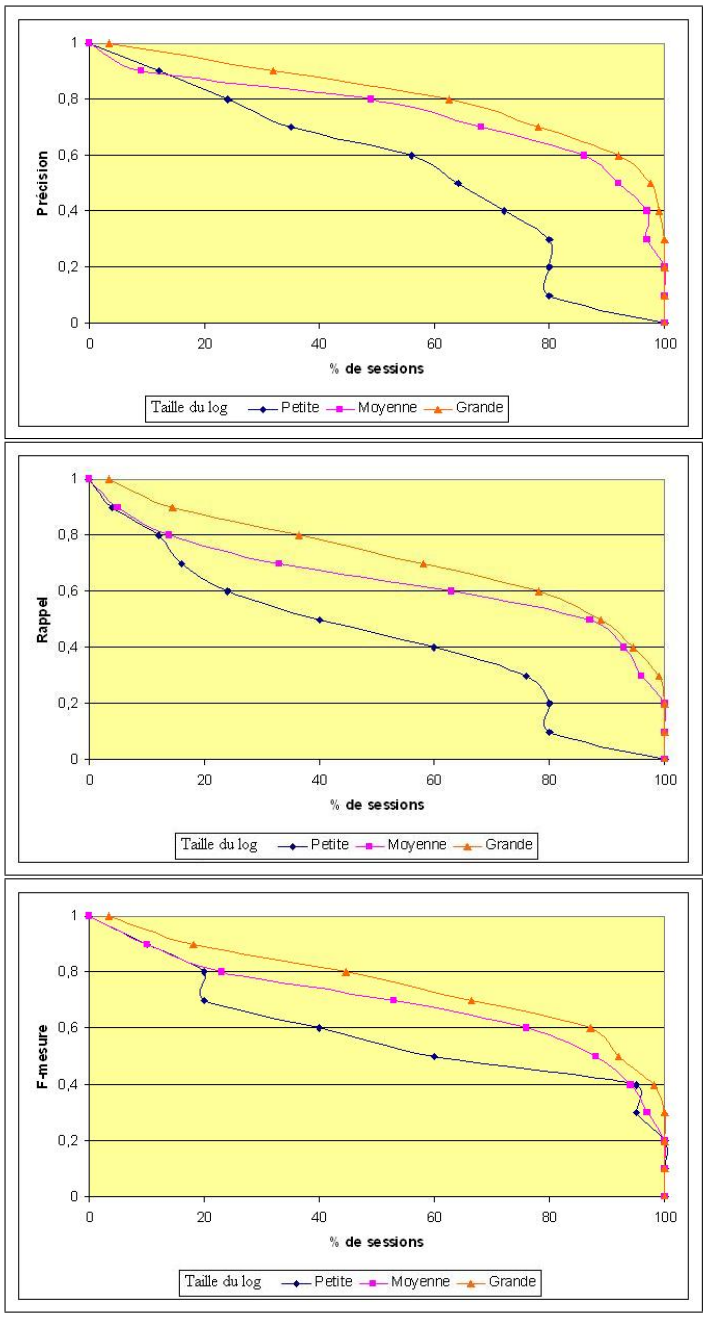

Figure 3. *Précision, Rappel et F-Mesure*

7. Conclusion

Dans cet article, nous présentons un système de recommandation de requêtes MDX qui est une évolution du cadre générique présenté dans (Giacometti *et al.*, 2008). Notre système (i) tient compte des précédentes navigations réalisées dans le cube, (ii) utilise une distance entre requêtes liée à une distance entre membres basée sur le plus court chemin prenant ainsi en compte une caractéristique essentielle en OLAP : les hiérarchies, (iii) partitionne le log de requêtes qui peut être volumineux et épars et (iv) permet de proposer des recommandations en adéquation avec le profil de l'utilisateur. Notre approche est implémentée dans un système intégrant Mondrian (moteur OLAP open source). Les expérimentations que nous avons menées sur des données synthétiques (générées par notre propre générateur de logs de requêtes) montrent que des recommandations peuvent être calculées efficacement et que notre système peut être réglé afin d'obtenir des recommandations de bonne qualité (objectivement).

Nos travaux futurs devraient nous permettre de :

– 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,

– incorporer les algorithmes de personnalisation de Bellatreche *et al.* (2006) dans notre prototype existant de recommandation,

– proposer d'autres instanciations de notre cadre comme dans (Giacometti *et al.*, 2009a; Giacometti *et al.*, 2009b; Negre, 2009) ou en 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.

A plus long terme, nous voudrions contribuer à un système collaboratif de management de requêtes comme celui de Khoussainova *et al.* (2009) et concevoir une plate-forme de génération de recommandations de requêtes MDX en donnant aux utilisateurs et aux administrateurs $OLAP$ la possibilité d'adapter l'approche à leurs besoins en proposant diverses méthodes de calcul des sessions et/ou requêtes candidates. Cette plate-forme intégrera notre cadre générique et devrait également inclure une nouvelle méthode, prenant en compte les valeurs des mesures et pas seulement les références des cellules, proposée dans (Giacometti *et al.*, 2009b) ainsi que des techniques basées sur le contenu (Chatzopoulou *et al.*, 2009) aussi bien que les méthodes combinant le contexte et le profil de l'utilisateur (Jerbi *et al.*, 2009), (Bellatreche *et al.*, 2005), (Golfarelli *et al.*, 2009).

8. Bibliographie

Adomavicius G., Tuzhilin A., « Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions », *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, n° 6, p. 734-749, 2005.

Baeza-Yates R. A., Hurtado C. A., Mendoza M., « Query Recommendation Using Query Logs in Search Engines. », *EDBT Workshops*, vol. 3268 of *Lecture Notes in Computer Science*, Springer, p. 588-596, 2004.

Baeza-Yates R. A., Ribeiro-Neto B. A., *Modern Information Retrieval*, ACM Press / Addison-Wesley, 1999.

Bellatreche L., Giacometti A., Marcel P., Mouloudi H., « Personalization of MDX Queries », *BDA*, 2006.

Bellatreche L., Giacometti A., Marcel P., Mouloudi H., Laurent D., « A personalization framework for OLAP queries », *DOLAP*, p. 9-18, 2005.

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

Bentayeb F., Favre C., « RoK: Roll-Up with the K-means Clustering Method for Recommending OLAP Queries », *DEXA*, 2009.

Chatzopoulou G., Eirinaki M., Polyzotis N., « Query Recommendations for Interactive Database Exploration », *SSDBM*, p. 3-18, 2009.

Dijkstra E. W., « A short introduction to the art of programming », August, 1971, circulated privately.

Floyd R. W., « Algorithm 97: Shortest path », *Commun. ACM*, vol. 5, n° 6, p. 345, 1962.

Giacometti A., Marcel P., Negre E., « A framework for recommending OLAP queries », *DOLAP*, p. 73-80, 2008.

Giacometti A., Marcel P., Negre E., « Recommending Multidimensional Queries », *DaWaK*, p. 453-466, 2009a.

Giacometti A., Marcel P., Negre E., Soulet A., « Query recommendations for OLAP discovery driven analysis », *DOLAP*, p. 81-88, 2009b.

Golfarelli M., Rizzi S., « Expressing OLAP Preferences », *SSDBM*, Springer-Verlag, Berlin, Heidelberg, p. 83-91, 2009.

Hamming R. W., « Error detecting and error correcting codes », *Syst. Tech. J.*, vol. 29, p. 147-160, 1950.

Hart P., Nilsson N., Raphael B., « A formal basis for the heuristic determination of minimum cost paths », *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, p. 100-107, 1968.

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

Jerbi H., Ravat F., Teste O., Zurfluh G., « Applying Recommendation Technology in OLAP Systems », *ICEIS*, n° 24 in *LNBIP*, Springer, p. 220-233, 2009.

Kaufmann L., Rousseeuw P. J., « Clustering by means of medoids », *in* Y. Dodge (ed.), *Statistical Data Analysis based on the L1 Norm*, Elsevier/North Holland, Amsterdam, p. 405-416, 1987.

Khoussainova N., Balazinska M., Gatterbauer W., Kwon Y., Suciu D., « A Case for A Collaborative Query Management System. », *CIDR*, www.crdrdb.org, 2009.

Kleinberg J. M., « Authoritative Sources in a Hyperlinked Environment », *J. ACM*, vol. 46, n° 5, p. 604-632, 1999.

Microsoft Corporation, « Multidimensional Expressions (MDX) Reference », 1998. Available at http:// msdn.microsoft.com/en-us/library/ms145506.aspx.

Navarro G., « A guided tour to approximate string matching », *ACM Comput. Surv.*, vol. 33, n° 1, p. 31-88, 2001.

Negre E., Exploration collaborative de cubes de données, PhD thesis, Université François Rabelais Tours, France, 2009.

Pentaho Corporation, « Mondrian open source OLAP engine », 2009. Available at http://mondrian.pentaho.org.

Sapia C., « On Modeling and Predicting Query Behavior in OLAP Systems », *DMDW*, p. 2.1-2.10, 1999.

Sarawagi S., « User-Adaptive Exploration of Multidimensional Data », *VLDB*, p. 307-316, 2000.

Sarawagi S., Agrawal R., Megiddo N., « Discovery-Driven Exploration of OLAP Data Cubes », *EDBT*, p. 168-182, 1998.

Srivastava J., Cooley R., Deshpande M., Tan P.-N., « Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data », *SIGKDD Explorations*, vol. 1, n° 2, p. 12-23, 2000.

van Rijsbergen C. J., *Information Retrieval*, Butterworth, 1979.

Yang X., Procopiuc C. M., Srivastava D., « Recommending Join Queries via Query Log Analysis », *ICDE*, p. 964-975, 2009.

Article reçu le 12 octobre 2009
Accepté après révisions le 17 novembre 2010

Elsa Negre *a obtenu son doctorat en informatique pour ses travaux sur l'exploration collaborative de cubes de données en 2009 à l'Université François Rabelais de Tours (France). Après deux stages postdoctoraux, l'un au LAboratoire de Recherche sur l'Information Multimédia de l'Université du Québec en Outaouais (Canada) et l'autre au Laboratoire Informatique de Nantes-Atlantique (France), elle est désormais Maître de conférences à l'Université Paris-Dauphine (Paris IX). Ses intérêts de recherche sont les bases de données multidimensionnelles, les entrepôts de données, OLAP, la découverte de connaissances dans les bases de données, les réseaux sociaux, la recommandation et la personnalisation. Elle est l'auteur d'articles publiés dans des conférences et des journaux nationaux et internationaux relatifs à ces sujets.*