UNIVERSITÉ FRANÇOIS RABELAIS

TOURS

École Doctorale SST
Laboratoire d'Informatique, Équipe BdTln

THÈSE présentée par :

Elsa NEGRE

soutenue le : **01 décembre 2009**

pour obtenir le grade de : **Docteur de l'Université François Rabelais Tours**

Discipline / Spécialité : **Informatique**

EXPLORATION COLLABORATIVE DE CUBES DE DONNÉES

THÈSE dirigée par :

M. Arnaud **Giacometti**	Professeur, Université François Rabelais Tours
M. Patrick **Marcel**	Maître de Conférences, Université François Rabelais Tours

RAPPORTEURS :

M. Pascal **Poncelet**	Professeur, Université Montpellier 2
M. Franck **Ravat**	Maître de Conférences, HDR, Université Toulouse I

JURY :

M. Mokrane **Bouzeghoub**	Professeur, Université de Versailles
M. Arnaud **Giacometti**	Professeur, Université François Rabelais Tours
M. Georges **Hebrail**	Professeur, Ecole Nationale Supérieure des Télécommunications, Paris
M. Patrick **Marcel**	Maître de Conférences, Université François Rabelais Tours
M. Franck **Ravat**	Maître de Conférences, HDR, Université Toulouse I

Volia pas morir la vièlha que totjorn aprenia.

(Proverbe occitan)

À tous ceux qui sont partis trop tôt...

Remerciements

Je souhaite tout d'abord exprimer ma sincère gratitude envers mes encadrants : Arnaud Giacometti, Professeur à l'Université François Rabelais Tours et directeur de ma thèse et Patrick Marcel, Maître de conférences à l'Université François Rabelais Tours. Leurs efforts en terme d'encadrement actif, d'encouragements et surtout de patience ainsi que leurs qualités scientifiques et humaines ont contribué à l'épanouissement de ce travail.

Je suis très reconnaissante à Pascal Poncelet, Professeur à l'Université de Montpellier 2 et à Franck Ravat, Maître de conférences HDR à l'Université de Toulouse I, d'avoir été rapporteurs de mon mémoire. Je les remercie vivement pour le travail qu'ils ont consacré à son évaluation.

Je remercie également Mokrane Bouzeghoub, Professeur à l'Université de Versailles et Georges Hébrail, Professeur à l'Ecole Nationale Supérieure des Télécommunications de Paris, qui m'ont fait l'honneur d'accepter de participer à mon jury de thèse.

Mes remerciements vont aussi à toute l'équipe de l'Antenne Universitaire de Blois, enseignants-chercheurs et IATOS.

J'ai cependant une pensée toute particulière pour mes compagnons de galère que ce soit les anciens doctorants : Adriana, Cheikh Ba, Ahmed Cheriat, Hassina Mouloudi et Tonio Wandmacher ou les futurs anciens doctorants : Lamine Baldé, Eynollah Khandjari, Marie Ndiaye, Cheikh Niang et Damien Nouvel à qui je souhaite une bonne fin de thèse.

Enfin, je n'oublie pas : mes parents, ma famille et mes amis qui, de près ou de loin, m'ont soutenue et encouragée.

Résumé

Les entrepôts de données stockent de gros volumes de données multidimensionnelles, consolidées et historisées dans le but d'être explorées et analysées par différents utilisateurs. L'exploration de données est un processus de recherche d'informations pertinentes au sein d'un ensemble de données. Dans le cadre de nos travaux, l'ensemble de données à explorer est un cube de données qui est un extrait de l'entrepôt de données que les utilisateurs interrogent en lançant des séquences de requêtes *OLAP* (*On-Line Analytical Processing*). Cependant, cette masse d'informations à explorer peut être très importante et variée, il est donc nécessaire d'aider l'utilisateur à y faire face en le guidant dans son exploration du cube de données afin qu'il trouve des informations pertinentes.

Le travail présenté dans cette thèse a pour objectif de proposer des recommandations, sous forme de requêtes *OLAP*, à un utilisateur interrogeant un cube de données. Cette proposition tire parti de ce qu'ont fait les autres utilisateurs lors de leurs précédentes explorations du même cube de données.

Nous commençons par présenter un aperçu du cadre et des techniques utilisés en Recherche d'Informations, Exploration des Usages du Web ou e-commerce. Puis, en nous inspirant de ce cadre, nous présentons un état de l'art sur l'aide à l'exploration des bases de données (relationnelles et multidimensionnelles). Cela nous permet de dégager des axes de travail dans le contexte des bases de données multidimensionnelles.

Par la suite, nous proposons donc un cadre générique de génération de recommandations, générique dans le sens où les trois étapes du processus sont paramétrables. Ainsi, à partir d'un ensemble de séquences de requêtes, correspondant aux explorations du cube de données faites précédemment par différents utilisateurs, et de la séquence de requêtes de l'utilisateur courant, notre cadre propose un ensemble de requêtes pouvant faire suite à la séquence de requêtes courante. Puis, diverses instanciations de ce cadre sont proposées.

Nous présentons ensuite un prototype écrit en *Java*. Il permet à un utilisateur de spécifier sa séquence de requêtes courante et lui renvoie un ensemble de recommandations. Ce prototype nous permet de valider notre approche et d'en vérifier l'efficacité avec une série d'expérimentations.

Finalement, afin d'améliorer cette aide collaborative à l'exploration de cubes de données et de permettre, notamment, le partage de requêtes, la navigation au sein des requêtes posées sur le cube de données, ou encore de les annoter, nous proposons un cadre d'organisation de requêtes. Ainsi, une instanciation adaptée à la gestion des recommandations est présentée.

Mots clés : Cubes de données, Exploration collaborative, Recommandations, Requêtes *OLAP*.

Abstract

Data warehouses store large volumes of consolidated and historized multidimensional data to be explored and analysed by various users. The data exploration is a process of searching relevant information in a dataset.

In this thesis, the dataset to explore is a data cube which is an extract of the data warehouse that users query by launching sequences of *OLAP* (*On-Line Analytical Processing*) queries. However, this volume of information can be very large and diversified, it is thus necessary to help the user to face this problem by guiding him/her in his/her data cube exploration in order to find relevant information.

The present work aims to propose recommendations, as *OLAP* queries, to a user querying a data cube. This proposal benefits from what the other users did during their previous explorations of the same data cube.

We start by presenting an overview of the used framework and techniques in Information Retrieval, Web Usage Mining or e-commerce. Then, inspired by this framework, we present a state of the art on collaborative assistance for data exploration in (relationnal and multidimensional) databases. It enables us to release work axes in the context of multidimensional databases.

Thereafter, we propose thus a generic framework to generate recommendations, generic in the sense that the three steps of the process are customizable. Thus, given a set of sequences of queries, corresponding to the previous explorations of various users, and given the sequence of queries of the current user, our framework proposes a set of queries as recommendations following his/her sequence. Then, various instantiations of our framework are proposed.

Then, we present a *Java* prototype allowing a user to specify his/her current sequence of queries and it returns a set of recommendations. This prototype validates our approach and its effectiveness thanks to an experimentations collection.

Finally, in order to improve this data cube exploration collaborative assistance and, in particular, to share, navigate or annotate the launched queries, we propose a framework to manage queries. Thus, an instantiation to manage recommendations is presented.

Keywords : Data Cubes, Collaborative exploration, Recommendations, *OLAP* queries.

Table des matières

Liste des figures

Chapitre 1

Introduction

Sommaire

Ce chapitre apporte une introduction au sujet principal de la thèse. Nous introduisons Section 1.1 le contexte de notre travail sur l'exploration collaborative de cubes de données, c'est-à-dire la recommandation de requêtes dans les bases de données multidimensionnelles. Nous présentons Section 1.2 les principales contributions, avant de terminer, Section 1.3 par une présentation de la structure générale du document.

1.1 Contexte de la thèse

Les systèmes d'aide à la décision synthétisent l'information et permettent aux utilisateurs d'explorer leurs données pour faciliter la prise de décision. L'exploration de données est un processus de recherche d'informations pertinentes, au sein d'un ensemble de données, destiné à détecter des corrélations cachées ou des informations nouvelles. Or, les utilisateurs doivent faire face à un volume toujours plus important d'informations en raison de l'accroissement des capacités de calcul et de stockage ([LVC+03] [1]). De sorte qu'il est de plus en plus difficile de savoir exactement quelles informations rechercher et où les chercher. Des techniques informatiques pour faciliter cette recherche ainsi que l'extraction des informations pertinentes sont donc nécessaires. L'une d'entre elles est la recommandation d'informations et, plus particulièrement, dans le cas de l'exploration d'une masse de données importante dont le paradigme d'accès à l'information est basé sur la formulation de requêtes.

Nous nous intéressons donc à la recommandation d'informations dans le contexte de l'exploration collaborative de cubes de données par requêtes *OLAP*.

Problèmes

Comment guider l'utilisateur dans son exploration des données afin qu'il trouve des informations qui sont pertinentes ?

Cela pose un certain nombre de problèmes.

Le premier est le temps de réponse élevé. Ce problème est d'autant plus crucial que l'utilisateur ne sachant pas, à priori, ce qu'il cherche, pose plusieurs requêtes pour trouver ce qui l'intéresse.

Le deuxième est la recherche de l'information en elle-même. Comme mentionné précédemment, la base de données peut être très volumineuse et l'utilisateur peut ne pas savoir où exactement chercher les informations pertinentes.

C'est pourquoi, nous faisons appel aux outils de recommandation afin de trouver le plus rapidement possible des informations pertinentes pour l'utilisateur.

Solutions

Le processus de recommandation va guider l'utilisateur lors de son exploration de la quantité d'informations à sa disposition en cherchant pour lui, les informations qui paraissent pertinentes. Il s'agit d'une forme particulière de filtrage de l'information visant à présenter les éléments d'information (films, musique, livres, news, images, pages Web, ...) qui sont susceptibles d'intéresser

1. Depuis l'an 2000, *School of Information Management and Systems* de l'Université de Berkeley en Californie, a étudié la quantité de nouvelles informations créées chaque année. Elle prend en compte toute information nouvellement créée, stockée sur quatre supports physiques (impression, film, magnétique et optique) et vue ou entendue dans quatre flux d'information empruntant des canaux électroniques (téléphone, radio, télévision et internet). L'étude estime la quantité d'informations nouvellement créée chaque année à environ 2 exaoctets d'informations par an ($1 exaoctet = 10^{18} octets$).

l'utilisateur. Généralement, à partir de certaines caractéristiques de référence[2], le processus de recommandation cherche à prédire l'"avis" que donnerait l'utilisateur à chaque élément et lui recommande les éléments d'information obtenant le meilleur "avis".

1.2 Contributions de la thèse

Les requêtes *OLAP*[3] ([CCS93]) permettent l'analyse interactive d'un gros volume de données multidimensionnelles à l'aide d'opérateurs spécifiques de consolidation et d'agrégation pour résumer l'information et de présentation et structuration pour naviguer et explorer l'information. Ces données sont organisées sous forme de cubes de données et proviennent d'un entrepôt de données, c'est-à-dire d'une base de données dont les tables constituent un schéma en étoile ([Kim96]). Un cube est une représentation multidimensionnelle des données dans cet entrepôt. Il est décrit par les tables de dimensions (les axes d'analyses) et une table de faits (les mesures).

Exemple 1.2.1 *Considérons l'exemple du cube 'MesVentes' dont le schéma conceptuel en étoile est donné Figure 1.1. La table de faits 'Ventes' détaille les valeurs des mesures des ventes réalisées de véhicules à différentes dates dans différents lieux. Les dimensions sont 'Temps', 'Géographie', 'Véhicules', 'NomMesures'*[4]*. Elles permettent de varier les points de vue (plus ou moins synthétiques) sur les faits.*

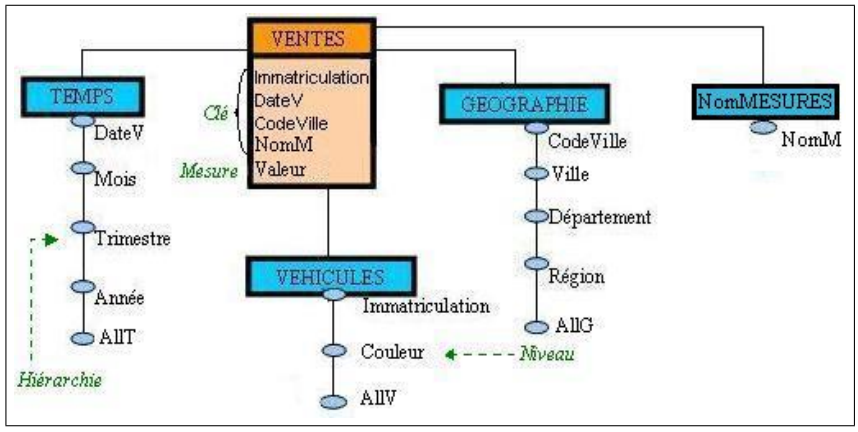


FIGURE 1.1 – Exemple de modélisation en étoile du cube *MesVentes* [GMR98]

Dans ce cas, un utilisateur accède aux données du cube à l'aide d'une requête *OLAP* dont la réponse est composée d'un ensemble de tuples de données détaillées ou agrégées, présentées sous forme de tableau croisé. Pour cela, on peut utiliser un langage spécifique tel que *MDX*[5] ([Mic98]).

Exemple 1.2.2 *Supposons que la requête utilisateur sur ce cube soit la suivante : le montant des ventes de véhicules en région Centre pour l'année 2008. Cette requête est exprimée en MDX comme*

2. Ces caractéristiques peuvent, par exemple, provenir des éléments d'information eux-même, on parle d'"approche basée sur le contenu" ou de l'environnement social, on parle de "filtrage collaboratif".

3. *OLAP* : *On-Line Analytical Processing*

4. La dimension *NomMesures* enregistre les noms des différentes mesures (Montant, Quantité), les valeurs sont, quant à elles, représentées par l'attribut *Valeur* dans la table de faits *Ventes*.

5. *MDX* : *MultiDimensional Expressions*

suit :

$SELECT$ $\{[Géographie].[AllG].[Région].[Centre]\}$ $ON\ COLUMNS$,
$\{[NomMesures].[Montant], [NomMesures].[Quantité]\}$ $ON\ ROWS$
$FROM$ $Ventes$
$WHERE$ $\{[Temps].[AllT].[Année].[2008]\}$

La requête ci-dessus, dont la réponse est visualisée par le tableau croisé de la Figure 1.2, définit les informations suivantes :

- *la clause SELECT spécifie l'ensemble des membres*[6] *devant apparaître sur les axes du tableau croisé (le membre 'Centre' du niveau 'Région' de la dimension 'Géographie', les membres 'Montant' et 'Quantité' du niveau 'NomM' de la dimension 'NomMesures'),*
- *la clause FROM indique que la source des données est la table de faits 'Ventes',*
- *la clause WHERE sélectionne le membre '2008' du niveau 'Année' de la dimension 'Temps'.*

De plus, dans cette requête, la syntaxe MDX distingue l'identification des axes d'affichage par COLUMNS et ROWS. Ainsi, l'emplacement des informations de la requête sur ces axes est précisé.

Ventes (2008)	Centre
Montant	12 000 000
Quantité	9 250

FIGURE 1.2 – Tableau croisé : *Montant des ventes de véhicules en région Centre en 2008*

Problèmes

Lorsqu'un utilisateur recherche des informations, il ne va pas se limiter à lancer une seule requête. Ainsi, pour accéder à l'information qui l'intéresse réellement, il va devoir lancer une séquence de requêtes. Mais quelles requêtes lancer exactement ? Dans quelle partie du cube de données naviguer pour obtenir l'information intéressante ?

L'exemple suivant illustre le problème de recommandation que nous proposons de traiter.

Exemple 1.2.3 *Supposons qu'un utilisateur A commence par lancer une requête q_1 permettant de voir toutes les données à leur niveau le plus agrégé, puis ne s'intéressant qu'à la ville de Blois, il lance une seconde requête q_2 en raffinant sa recherche sur le membre 'Blois' du niveau 'Ville' de la dimension 'Géographie'. Nous notons cette séquence de requête ainsi : $q_1 \rightarrow q_2$. Il ne sait pas où aller chercher l'information à partir de là : $q_1 \rightarrow q_2 \rightarrow ?$*
Supposons maintenant qu'un autre utilisateur B ne s'intéressant aussi qu'à la ville de Blois, ait déjà lancé des requêtes sur le même cube de données et se soit intéressé, par exemple, aux ventes de véhicules rouges à Blois via la requête q_6. Pourquoi ne pas proposer à l'utilisateur A la requête q_6 pour l'aider à avancer dans son exploration en lui proposant sa prochaine requête ?

Par conséquent, notre problème est le suivant : *Comment aider l'utilisateur à avancer dans son exploration du cube de données en lui proposant des requêtes pertinentes ?*

Dans notre contexte particulier, recommander des requêtes soulève les problèmes suivants :

6. Un membre est une valeur d'un attribut d'une table de dimension.

1. Si nous voulons proposer à l'utilisateur une requête issue d'autres utilisateurs mais relativement proche de la sienne, comment déterminer la similarité / proximité entre deux requêtes ? De la même manière, comment tenir compte du fait que deux requêtes syntaxiquement différentes peuvent retourner le même tableau croisé ?
2. Si nous prenons en compte la nature séquentielle de la recherche, si la même séquence de requêtes a déjà été lancée, comment trouver une telle séquence et proposer une recommandation à l'utilisateur ?
3. Dans le cas où plusieurs requêtes peuvent être recommandées à l'utilisateur, laquelle présenter en premier ?
4. Sachant que le nombre de requêtes lancées sur le cube de données peut être très grand, comment faciliter l'accès à ces requêtes ainsi que leur exploitation ?

Solutions
Ce travail propose d'apporter les réponses suivantes à ces problèmes :

1. Deux requêtes doivent pouvoir être comparées en déterminant leur similarité / proximité. Ainsi, une distance entre requêtes sera exploitée pour proposer des recommandations au plus proche des attentes de l'utilisateur.
2. Toujours afin d'être au plus près des attentes de l'utilisateur et afin de trouver des séquences proches de celles de l'utilisateur, deux séquences de requêtes doivent pouvoir être comparées. Ainsi, une distance entre séquences de requêtes sera exploitée par l'algorithme de recommandation de requêtes.
3. Les requêtes candidates à la recommandation doivent être ordonnées pour être présentées à l'utilisateur. Cet ordre sera exploité par l'algorithme de recommandation pour proposer en premier les requêtes les plus pertinentes pour l'utilisateur.
4. Les requêtes précédemment lancées sur le cube de données doivent être organisées afin de faciliter leur exploitation. Ainsi, une organisation des requêtes sera proposée.

1.3 Plan de la thèse

La suite du document est structurée de la manière suivante :

Le Chapitre 2 est consacré à l'état de l'art. Nous y présentons les notions inhérentes à l'exploration collaborative de cube de données. Ce chapitre commence par présenter un aperçu des travaux concernant les bases de données multidimensionnelles. Il se poursuit sur les outils de gestion de logs, puis les systèmes de recommandations. Enfin, il se termine par la présentation et la comparaison des méthodes d'aide à l'exploration existantes dans le domaine des bases de données.

Les Chapitres 3, 4 et 5 constituent le cœur de notre travail. Nous y détaillons respectivement notre cadre générique de génération de recommandations, des instanciations particulières et notre cadre générique d'organisation des analyses.

Le Chapitre 3 présente notre cadre générique de génération de recommandations de requêtes *MDX* basé sur une méthode de filtrage collaboratif. Chaque étape générique du processus de recommandation est présentée puis instanciée via différents paramètres. Enfin, quatre combinaisons

de ces paramètres sont proposées.

Le Chapitre 4 présente l'implémentation de notre travail, à savoir notre système de recommandation de requêtes MDX et les données sur lesquelles nous avons menés les expérimentations dont le protocole et les résultats sont exposés.

Le Chapitre 5 présente notre cadre générique d'organisation des analyses. Cette présentation se fait en précisant les deux niveaux constitutifs de notre cadre ainsi que les langages associés. Puis, une instanciation de ce cadre générique est proposée dans le contexte de la recommandation de requêtes MDX.

Le Chapitre 6 termine ce mémoire par une conclusion qui présente les apports de notre travail ainsi que les perspectives de recherche envisagées.

Chapitre 2

État de l'art

Sommaire

Les systèmes de bases de données multidimensionnelles deviennent de plus en plus populaires pour contrôler et analyser de grands volumes de données issues de l'entrepôt de données. Le principal avantage de ces systèmes de bases de données est qu'ils permettent l'exécution efficace de requêtes complexes donnant ainsi la possibilité aux utilisateurs d'explorer interactivement les données et de rechercher des informations intéressantes. Cependant, les utilisateurs sont rapidement confrontés à une tâche difficile et pénible afin d'analyser de telles bases de données avec des critères multiples et la découverte des informations utiles demeure un défi important.

Ils voudraient répondre à des questions du type :

- Comment connaître quelles parties de la base de données fournissent des informations intéressantes ?
- Comment trouver rapidement quelle partie du cube de données contient une valeur très différente de celles prévues ?
- Comment découvrir rapidement les dimensions qui sont les plus corrélées les unes aux autres ?

Cela gêne clairement l'exploration de données, et réduit ainsi les avantages d'employer un système de bases de données. Nous pensons que l'effort manuel et le temps passé dans l'analyse pourraient être réduits en prévoyant la stratégie de l'utilisateur et en recommandant des données appropriées pour la prise de décision.

L'objectif de ce chapitre est de présenter un état de l'art sur l'exploration collaborative des bases de données multidimensionnelles afin de proposer des recommandations à l'utilisateur. A cet effet, dans la Section 2.1 nous présentons un cadre formel général pour définir les concepts de base de la modélisation et de l'analyse des bases de données multidimensionnelles. La Section 2.2 présente les méthodes existantes de gestion des logs. Puis, la Section 2.3 est consacrée aux concepts et méthodes liés aux systèmes de recommandations. Enfin, nous présentons dans la Section 2.4, les méthodes utilisées dans les bases de données relationnelles et multidimensionnelles à des fins d'aide à l'exploration.

2.1 Bases de données multidimensionnelles et Analyse

De nos jours, les volumes de données à traiter sont de plus en plus importants [LVC+03]. Apparus pour gérer de tels volumes de données issues de sources hétérogènes, les entrepôts de données constituent l'outil essentiel de collecte et de mise à disposition des données en vue de leur analyse interactive, rapide et dynamique [Inm92, JLVV01].

Pour gérer les données, les entreprises disposent d'un côté, de systèmes transactionnels (OLTP [1]) qui traitent les données sources de l'entrepôt et de l'autre, de systèmes analytiques (OLAP [2]) [CCS93] qui, quant à eux, traitent les données contenues dans l'entrepôt permettant ainsi une analyse efficace des données grâce à différents outils tels que les requêteurs ou les logiciels d'analyses et de fouille.
La Figure 2.1 [CD97] résume cette démarche où les données sources sont entreposées puis modélisées sous forme de cubes de données avant d'être analysées.

Nous allons présenter dans cette section, les notions relatives à la manipulation des données de l'entrepôt.

1. *On-Line Transactional Processing*
2. *On-Line Analytical Processing*

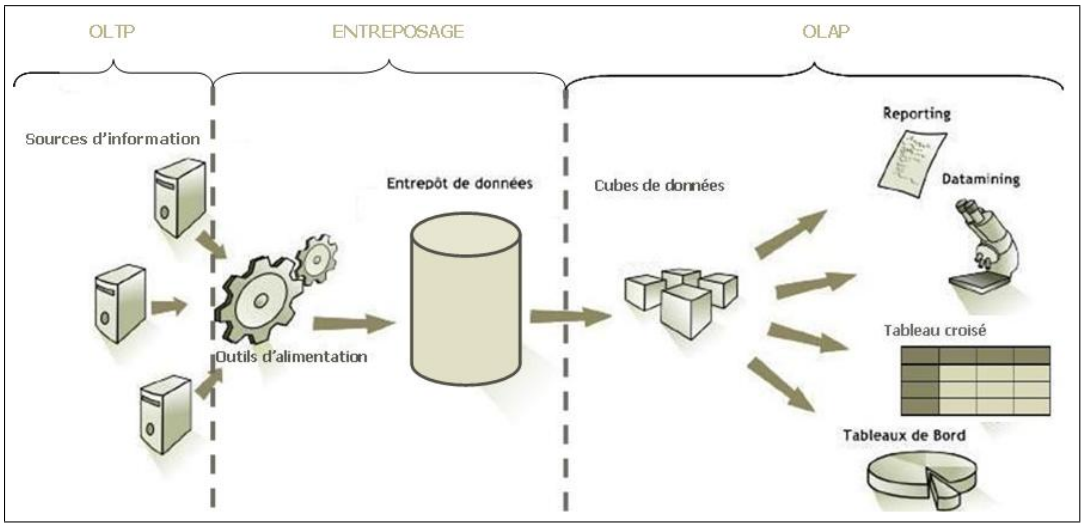


FIGURE 2.1 – Démarche décisionnelle

2.1.1 Modélisation multidimensionnelle dans les entrepôts de données

2.1.1.1 Les modèles de données pour les entrepôts

Pour être analysées, les données multidimensionnelles issues de l'entrepôt sont structurées selon plusieurs perspectives d'analyses pouvant représenter des notions variées telles que le temps ou la localisation géographique [Inm92, KR02]. Un *cube de données* contient le sujet d'analyse (les *mesures* du *fait*) qui est placé au centre d'un espace à plusieurs perspectives d'analyses (les *dimensions*).

Exemple 2.1.1 *La Figure 2.2 représente une instance du cube MesVentes des données relatives à des ventes (les faits) analysées en fonction des dimensions Véhicules, Temps, Géographie et Nom-Mesures (Montant, Quantité).*

Les trois modèles multidimensionnels les plus utilisés sont :
- le schéma en étoile (*star schema* [Kim96, JLVV01]) qui représente visuellement une étoile. Une table de faits centrale en $BCNF$[3] contenant les faits à analyser est reliée aux tables de dimensions par des clés étrangères. Chaque dimension est décrite par une seule table. Ce schéma en étoile est le plus implanté et est à la base des deux modèles suivants.
- le schéma en flocon (*snowflake schema* [JLS99]) qui représente la table de faits selon le même principe que le schéma en étoile, mais les tables de dimensions sont normalisées afin de représenter plus explicitement les hiérarchies et réduire les redondances.
- le schéma en constellation (*factflake schema* [CD97]) qui représente plusieurs tables de faits partageant des tables de dimensions.

Une dimension modélise donc une perspective d'analyse selon laquelle sont visualisées les données. Elle est composée d'attributs qui représentent les niveaux de granularité de visualisation des données. Les attributs d'une dimension sont organisés selon leur niveau de détail (granularité) en une ou plusieurs hiérarchies. Une hiérarchie organise les attributs d'une dimension de la granularité la plus fine vers la granularité la plus générale.

3. $BCNF$: la forme normale de *Boyce-Codd* [AHV95]

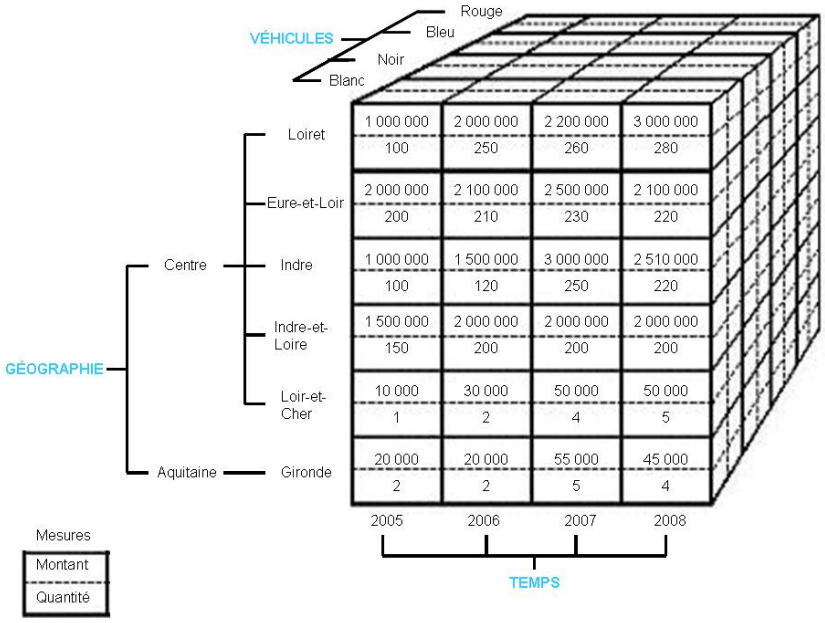


FIGURE 2.2 – Exemple d'une instance de cube $MesVentes$

Le fait, quant à lui, modélise le sujet d'analyse. Un fait est composé de valeurs de mesures représentant les informations des données à analyser via des fonctions d'agrégation. Les valeurs de mesures sont majoritairement numériques, non nulles et donc aptes à être manipulées via des fonctions d'agrégation.
Rappelons que selon [CCS93], dimensions et mesures doivent pouvoir être traitées symétriquement.

2.1.1.2 Le modèle de données dans notre contexte

Dans notre approche, nous modélisons un cube de données de manière classique et fréquemment utilisée comme un schéma en étoile. Ainsi, d'un point de vue conceptuel, la modélisation multidimensionnelle est liée aux concepts de *cube*, de *fait* et de *dimension*.

Dans ce qui suit, le lecteur doit posséder des connaissances de base en bases de données [AHV95].

Définition 2.1.1 *(Dimension et Hiérarchie)*
Une dimension, de nom D, est une instance de relation appelée table de dimensions de sorte[4] *$sort(D) = \{L^0, \ldots, L^d\}$.*
Pour une dimension D, chaque attribut L^j décrit un niveau de la hiérarchie, j étant la profondeur de ce niveau. L^0 est le niveau le plus bas qui est aussi la clé primaire de D. Chaque niveau L^j de la hiérarchie est l'enfant d'un seul parent présent au niveau immédiatement supérieur L^{j+1} de la hiérarchie. Un niveau parent représente une agrégation du niveau enfant. Ainsi, $\forall L^j, L^{j+1} \in sort(D)$, nous avons la dépendance fonctionnelle $L^j \rightarrow L^{j+1}$.

Dans la suite, notons que le nom d'une dimension D, est aussi utilisé pour dénoter un attribut de domaine actif $adom(D) = \bigcup_{j=0}^{d} adom(L^j)$, $adom(D)$ étant l'ensemble de tous les membres de la

4. Nous utilisons ici la terminologie de [AHV95], où *sort* correspond à l'ensemble des attributs d'une relation.

dimension D.
De plus, notons que les dimensions plates, comme la dimension contenant les mesures du fait, sont considérées comme arrangées dans une hiérarchie où tous les membres ont un ancêtre commun : la racine de la hiérarchie.

Exemple 2.1.2 *Nous définissons les dimensions Véhicules, Temps, Géographie et NomMesures :*
- *VÉHICULES : Un véhicule possède un identifiant immatriculation et une couleur : $sort(VÉHICULES) = \{Immatriculation, Couleur, AllV\}$*
- *TEMPS : La dimension temporelle permet de décomposer une donnée par date, mois, trimestre et année : $sort(TEMPS) = \{DateV, Mois, Trimestre, Année, AllT\}$*
- *GÉOGRAPHIE : La dimension géographique consiste en un code, une ville, un département et une région : $sort(GÉOGRAPHIE) = \{CodeVille, Ville, Département, Région, AllG\}$.*
- *NomMESURES : Cette dimension est une dimension plate qui contient les noms des mesures : $sort(NomMESURES) = \{NomM\}$.*

Définition 2.1.2 *(Fait)*
Une table de faits de nom F est une instance de relation de sorte $sort(F) = \{L_1^0, \ldots, L_N^0, m\}$ où m est un attribut numérique représentant des valeurs de mesures et $\{L_i^0, i \in [1, N]\}$ forme la clé primaire de F où, pour tout $i \in [1, N]$, L_i^0 est la clé primaire de la dimension D_i. Un fait est un tuple de la table de faits F.

Exemple 2.1.3 *La table de faits VENTES en vue de l'analyse des valeurs de mesures Montant et Quantité pour les dimensions VÉHICULES, TEMPS, GÉOGRAPHIE et NomMESURES a pour sorte : $sort(VENTES) = \{Immatriculation, DateV, CodeVille, NomM, Valeur\}$ où l'attribut numérique $Valeur$ donne la valeur des mesures Montant et Quantité.*

Définition 2.1.3 *(Cube)*
Un cube N-dimensionnel C est un n-uplet $C = \langle D_1, \ldots, D_N, F\rangle$ de sorte $sort(C) = \{D_1, \ldots, D_N, F\}$ où :
- *Pour $i \in [1, N]$, D_i est une dimension,*
- *F est une table de faits.*

Nous notons $\mathscr{D}(C)$ l'ensemble des dimensions du cube C, c'est-à-dire $\mathscr{D}(C) = \langle D_1, ..., D_N\rangle$.

Définition 2.1.4 *(Cellule)*
Soit un cube N-dimensionnel $C = \langle D_1, \ldots, D_N, F\rangle$, une cellule de cube est un $(N+1)$-uplet $\langle r_1, \ldots, r_N, m\rangle$ où m est une valeur d'attribut mesure [5] ou la constante $NULL$ si $\sigma_{L_1^0=r_1 \wedge \ldots \wedge L_N^0=r_N}(F) = \emptyset$ et $r_i \in adom(D_i)$ pour tout $i \in [1, N]$.

Définition 2.1.5 *(Référence)*
Soit un cube N-dimensionnel $C = \langle D_1, \ldots, D_N, F\rangle$, une référence de cellule de cube (ou référence par abus) est un N-uplet $\langle r_1, \ldots, r_N\rangle$ où $r_i \in adom(D_i)$ pour tout $i \in [1, N]$.
Nous notons $ref(C)$ l'ensemble de toutes les références du cube C.

Exemple 2.1.4 *La Figure 2.3 représente le schéma en étoile de l'instance du cube présenté Exemple 2.1.1 selon le formalisme graphique de [GMR98]. Il s'agit de l'analyse des valeurs de Montant et Quantité des VENTES selon les dimensions : TEMPS, VÉHICULES, GÉOGRAPHIE et NomMESURES.*

5. Dans le cas où il y a plusieurs mesures, il faut les mettre dans des cellules différentes en faisant varier le membre de la dimension $NomMESURES$.

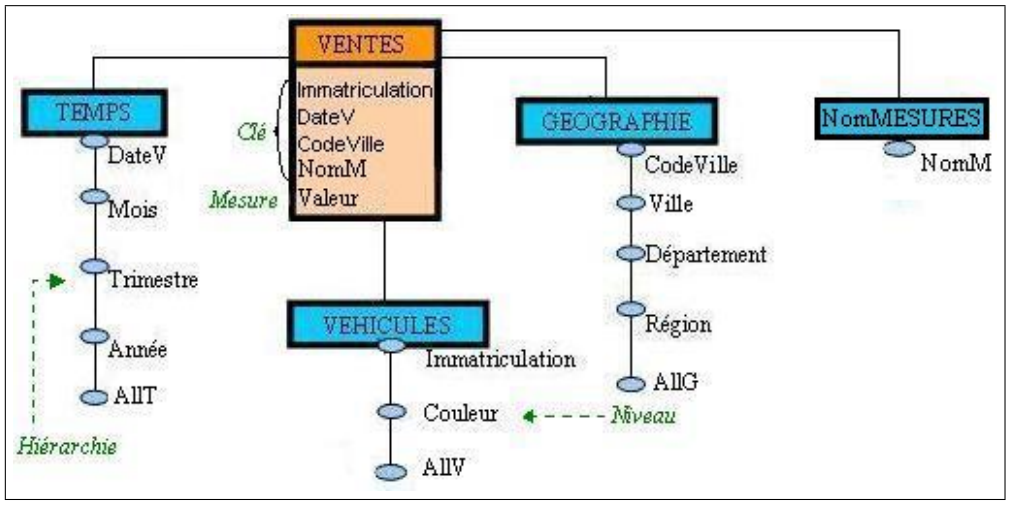


FIGURE 2.3 – Exemple de modélisation en étoile du cube $MesVentes$

Ainsi, le cube MesVentes est composé de la table de faits VENTES et des quatre dimensions NomMESURES, TEMPS, VÉHICULES et GÉOGRAPHIE. Les cellules de ce cube contiennent les valeurs des mesures Montant et Quantité pour un véhicule particulier sur une localisation particulière pour une période donnée.

2.1.2 Les langages de manipulation de données

Afin d'explorer un cube, l'utilisateur a besoin d'opérations et d'un langage de manipulation de données. Dans le domaine des bases de données relationnelles, il existe des opérations relationnelles et le langage standard est le langage SQL[6]. Nous présentons dans cette section les opérations propres au requêtage multidimensionnel ainsi qu'un langage approprié.

2.1.2.1 Les opérations et langages existants

Les opérations OLAP élémentaires [CD97, RA07] sont traditionnellement réparties en trois catégories :

- les opérations de restructuration qui permettent de changer la structure selon laquelle les données sont visualisées,
- les opérations de granularité qui permettent de naviguer au travers des données multidimensionnelles selon plusieurs niveaux de granularité,
- les opérations ensemblistes qui permettent la manipulation de données multidimensionnelles.

Nous ne rentrerons pas dans le détail de chaque catégorie d'opérations. Notons cependant que les opérations OLAP les plus utilisées sont [JLVV01] :

- Roll-up et Drill-down : ces opérateurs de granularité permettent de représenter les données d'un cube à un niveau de granularité immédiatement supérieur (agrégation des données pour le Roll-up) ou inférieur (détail/forage des données pour le Drill-down) selon une dimension,

6. *Structured Query Language*

- Slice et Dice : ces opérateurs ensemblistes permettent de réaliser des sélections et des projections sur les données d'un cube,
- Pivot (ou Rotate) : cet opérateur de restructuration réoriente la vue multidimensionnelle en tournant le cube.

La communauté des bases de données s'est intéressée à la définition d'un langage dédié aux données multidimensionnelles. Plusieurs travaux ont été proposés, citons par exemple, [AGS97, GL97, CT98], le travail de synthèse [JLVV01] ou, plus récemment [RTTZ08]. Chaque langage présente des propriétés qui lui sont spécifiques et aucun langage multidimensionnel formel standard n'a été établi.

Notons cependant qu'en pratique, le langage utilisé par tous les serveurs OLAP, que ce soit [SAS09, Pen09] ou récemment [ORA09], est le langage MDX [7] proposé par [Mic98]. Il est doté d'une syntaxe de type SQL et représente un moyen syntaxique d'interrogation des données multidimensionnelles d'un cube.

2.1.2.2 Le langage utilisé dans notre approche

Dans notre approche, le langage utilisé est MDX, qui est le langage le plus utilisé pour la manipulation de données multidimensionnelles. En particulier, nous nous intéressons à un sous-ensemble du langage MDX. Nous en donnons une définition simple dans ce qui suit. Notons qu'une tentative de formalisation complète a été proposée par [MVSV03].

A la différence d'une requête SQL, une requête MDX renvoie une grille multidimensionnelle de cellules (ou tableau croisé) comme résultat de la requête, où les lignes et les colonnes sont les axes selon lesquels le cube est exploré et où chaque cellule contient la (ou les) mesure(s) calculée(s).

Exemple 2.1.5 *Considérons la requête suivante :*

```
SELECT  Crossjoin({[Véhicules].[AllV].[Couleur].[Rouge]},
        {[Géographie].[AllG].[Région].[Centre],
        [Géographie].[AllG].[Région].[Limousin] })      ON ROWS,
        {[Temps].[AllT].[Année].Members}                ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]} ;
```

Cette expression MDX recherche les cellules contenant le montant (clause $WHERE$) des ventes (clause $FROM$) de véhicules rouges en régions Centre et Limousin pour toutes les années possibles (clause $SELECT$).
Elle indique aussi que les membres des dimensions Véhicules et Géographie seront imbriqués ($Crossjoin$) et seront affichés en lignes ($ON\ ROWS$), tandis que les membres de la dimension Temps apparaîtront en colonnes ($ON\ COLUMNS$). La structure d'affichage du résultat est donc spécifiée par l'utilisateur grâce aux clauses $CROSSJOIN$ et ON dans la clause $SELECT$.

Exemple 2.1.6 *La requête MDX considérée dans l'exemple précédent (Exemple 2.1.5) contient l'expression d'ensemble $[Temps].[AllT].[Année].Members$. Cette expression est une requête qui peut être exprimée dans l'algèbre relationnelle par : $\pi_{Année}(Temps)$*

7. *MultiDimensional Expressions*

Plus précisément, nous nous intéressons uniquement à des requêtes MDX dont la forme syntaxique est celle présentée ci-dessus et dont la représentation peut être faite sous forme de tuple, comme défini ci-après.

Définition 2.1.6 *(Requête MDX)*
Soit un cube de données N-dimensionnel $C = \langle D_1, \ldots, D_N, F \rangle$*, une requête MDX sur* C*, notée* q_{MDX}*, est une requête exprimée dans le langage MDX qui peut se représenter par un n-uplet :*

$$q_{MDX} = \langle q_1, ..., q_N \rangle$$

où $\forall i \in [1, N]$*,* q_i *est une requête de l'algèbre relationnelle. Elle permet d'obtenir les membres de la dimension* D_i*.*
Les références de la requête q_{MDX} *sont l'ensemble des références de* q_{MDX}*,* $ref(q_{MDX}) = q_1(D_1) \times ... \times q_N(D_N) = \times_{i=1}^{N} q_i(D_i)$*.*

Exemple 2.1.7 *Considérons la requête MDX de l'Exemple 2.1.5 posée sur le cube MesVentes. Nous pouvons représenter cette requête par le n-uplet* $\langle q_1, q_2, q_3, q_4 \rangle$*, où les requêtes sur les tables de dimensions de l'instance du cube MesVentes sont :*

- $q_1 = \{Rouge\}$
- $q_2 = \{Centre, Limousin\}$
- $q_3 = \pi_{Année}(Temps)$
- $q_4 = \{Montant\}$

Dans la suite, nous considérons une seule instance de cube, ainsi, une requête q est assimilée à ses références telle que $ref(q) = q$.

Exemple 2.1.8 *Reprenons la requête MDX considérée dans l'Exemple 2.1.5 et supposons que les membres du niveau Année* $\in [2005, 2008]$*, l'ensemble de références correspondant à la requête est :*

$$\{\langle Montant, Rouge, Centre, 2005\rangle, \langle Montant, Rouge, Limousin, 2005\rangle$$
$$\langle Montant, Rouge, Centre, 2006\rangle, \langle Montant, Rouge, Limousin, 2006\rangle$$
$$\langle Montant, Rouge, Centre, 2007\rangle, \langle Montant, Rouge, Limousin, 2007\rangle$$
$$\langle Montant, Rouge, Centre, 2008\rangle, \langle Montant, Rouge, Limousin, 2008\rangle\}$$

Le résultat de cette requête est présenté Figure 2.4.
Enfin, la cellule colorée pointée est : $\langle Montant, Rouge, Centre, 2008, 11000000\rangle$*.*

VENTES : Montant		2005	2006	2007	2008
Rouge	Centre	7 000 000	9 000 000	12 000 000	11 000 000
	Limousin	6 000 000	8 000 000	15 000 000	11 000 000

Cellule

FIGURE 2.4 – Résultat d'une requête MDX

2.1.3 Analyse multidimensionnelle

L'analyse multidimensionnelle consiste en l'interrogation d'un cube de données afin de manipuler et d'analyser les données multidimensionnelles.

2.1.3.1 Intuition d'une analyse et modèles existants

Malgré le développement des systèmes décisionnels, les bases de données multidimensionnelles et la technologie OLAP souffrent d'un manque de formalisation standard. En effet, la communauté des bases de données s'intéresse à la définition générale de ce qu'est une analyse multidimensionnelle qui prendrait en compte tous les aspects de la navigation de l'utilisateur. En effet, la navigation ou analyse OLAP [Sar00, TSM01, DKK05] est un terme utilisé pour caractériser le fait qu'un décideur explorant interactivement un cube de données cherche souvent à obtenir des résultats ou à expliquer ces résultats.

[RA07] propose un état de l'art en passant par [VS00] et [FK04]. Cependant, chaque approche présente une vision particulière des besoins, de la terminologie et du formalisme de l'analyse multidimensionnelle. Par conséquent, il n'y a aucune définition formelle standard établie. D'après [DKK05], un utilisateur commence à analyser des données en choisissant une première requête. Cette requête est composée d'un ensemble de dimensions sur l'axe des abscisses et l'axe des ordonnées comme un ensemble de conditions de filtrage. Puis, l'utilisateur modifie la requête interactivement en ajoutant ou supprimant des colonnes (drill-down et roll-up), en ajoutant ou supprimant des conditions de filtrage (slice), en déplaçant des colonnes d'un axe à l'autre (dice) et ainsi de suite.

En outre, d'après [Sar00], une analyse pilotée par la découverte [8] démarre typiquement au niveau le plus haut des hiérarchies des dimensions du cube. Puis, la navigation dans le cube se fait en appliquant une séquence d'opérations comme *Drill-down*, *Roll-up* et *Slice*. A partir du plus haut niveau de la hiérarchie, le décideur observe un niveau plus bas d'une hiérarchie en regardant les valeurs agrégées et en identifiant visuellement des valeurs intéressantes. Si une exploration ne donne pas de résultats intéressants, le décideur remonte au niveau le plus haut des hiérarchies des dimensions du cube et continue son analyse dans une autre direction, en allant observer d'autres dimensions.

Le but de ces manipulations est de pouvoir découvrir des aspects insoupçonnables dans la masse de données de l'entrepôt de données permettant ainsi l'enrichissement de l'analyse exploratoire du décideur.

Exemple 2.1.9 *Supposons qu'un utilisateur navigue dans le cube de données présenté Figure 2.3 afin d'analyser le Montant des Ventes en fonction des axes Véhicules, Géographie et Temps. Son interrogation est la suivante :*

Quels sont les véhicules qui réalisent les plus mauvaises ventes en région Centre et plus particulièrement, dans quelle ville ?

Sa première requête va consister à obtenir le montant des ventes de tous les véhicules, quelle que soit la localisation géographique et quelles que soient les informations temporelles :

$$
\begin{array}{lllll}
q_1 = & SELECT & \{[Véhicules].[AllV].Members\} & ON\ COLUMNS, \\
 & & \{[Géographie].[AllG].Members\} & ON\ ROWS \\
 & FROM & [Ventes] & \\
 & WHERE & \{[NomMesures].[Montant]\}; &
\end{array}
$$

8. *Discovery driven analysis*

Il s'intéresse à la répartition géographique de ces ventes, il va donc observer cette dimension et plus particulièrement le montant des ventes de véhicules, quelles que soient les informations temporelles, dans le département d'Indre-Et-Loire :

$$\begin{array}{lllll} q_2 = & SELECT & \{[Véhicules].[AllV].Members\} & & ON\ COLUMNS, \\ & & \{[Géographie].[AllG].[Centre].[IndreEtLoire]\} & & ON\ ROWS \\ & FROM & [Ventes] & & \\ & WHERE & \{[NomMesures].[Montant]\}; & & \end{array}$$

Il remarque des similitudes entre certains véhicules, par exemple, les montants des ventes de véhicules rouges et de véhicules bleus en Indre-Et-Loire sont significativement plus bas que pour les véhicules de couleurs différentes. Il va donc s'intéresser plus avant à ces deux couleurs de véhicules en lançant une requête permettant d'obtenir le montant des ventes de véhicules rouges ou de véhicules bleus dans le département d'Indre-Et-Loire, et plus particulièrement pour chaque ville, afin de déterminer si cette tendance se retrouve, quelles que soient les informations temporelles :

$$\begin{array}{lllll} q_3 = & SELECT & \{[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]\} & & ON\ COLUMNS, \\ & & \{[Géographie].[AllG].[Centre].[IndreEtLoire].Children\} & & ON\ ROWS \\ & FROM & [Ventes] & & \\ & WHERE & \{[NomMesures].[Montant]\}; & & \end{array}$$

En observant la réponse à cette dernière requête, l'utilisateur observe que les montants des ventes de véhicules rouges et de véhicules bleus sont significativement plus bas que pour les véhicules de couleurs différentes uniquement dans la ville de Tours. L'utilisateur obtient donc une réponse à son interrogation : Les véhicules réalisant les plus mauvaises ventes sont les véhicules rouges et les véhicules bleus dans la ville de Tours.

2.1.3.2 Analyse multidimensionnelle dans notre contexte

Comme présentée Section 2.1.3.1, une analyse multidimensionnelle ne se limite pas à un enchaînement de requêtes et sera définie Chapitre 5. Cependant, nous nous intéressons à cet enchaînement de requêtes ou séquence de requêtes sur un cube de données afin de répondre à un besoin décisionnel. Cette séquence est ce que nous appelons une session d'analyse.

Définition 2.1.7 *(Session de requêtes MDX)*
Soit un cube C, une session d'analyse (session par abus) $s = \langle q_1, \ldots, q_p \rangle$ sur C est une séquence finie de requêtes de query(C) (Définition 2.1.6).
Nous notons query(s) l'ensemble des requêtes de la session s, session(C) l'ensemble de toutes les sessions sur un cube C et $s[i]$ la $i^{ème}$ requête de la session s.

Exemple 2.1.10 *La session correspondant à l'enchaînement de requêtes présenté Exemple 2.1.9 est : $q_1 \rightarrow q_2 \rightarrow q_3$. Cette session constitue la session courante des données présentées Annexe A.*

Dans le cadre de nos travaux, les différentes sessions utilisateur sont enregistrées dans ce que nous avons appelé un log.

Définition 2.1.8 *(Log)*
Soit un cube C, un log de base de données (log par abus) est un ensemble fini de sessions de session(C).
Nous notons query($\mathscr{L}$) l'ensemble de requêtes issues du log $\mathscr{L}$.

2.1.4 Synthèse

Un entrepôt de données extrait, transforme et charge des données issues de systèmes OLTP de manière autonome, sans interaction avec l'utilisateur. Il organise les données dans des cubes de données en classant les données en *faits* et *dimensions*. Les faits contiennent les données à analyser, tandis que les dimensions représentent les différentes perspectives de visualisation des données. Une fois que les données sont chargées, les décideurs réalisent des sessions d'analyses en utilisant des outils OLAP tels que des requêtes *MDX* sur les cubes de données afin de trouver les solutions à différentes tâches décisionnelles. Enfin, ces sessions sont enregistrées dans un log de sessions de requêtes. La Figure 2.5 synthétise cette démarche.

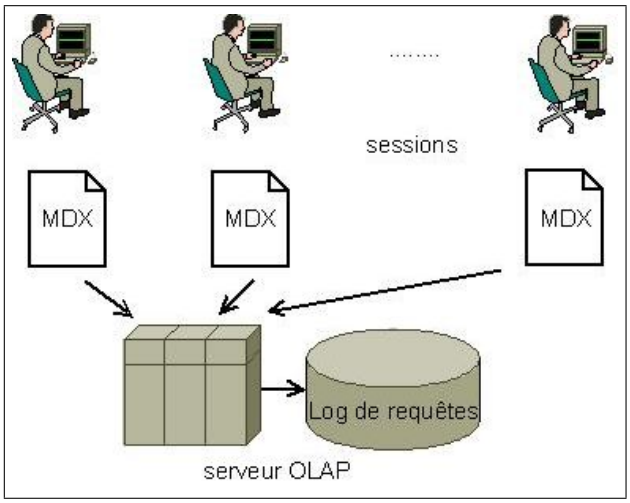


FIGURE 2.5 – Synthèse des bases de données multidimensionnelles

2.2 Outils pour la gestion des logs

Dans cette section, nous rappelons brièvement les caractéristiques de certains outils de gestion de logs de requêtes.

2.2.1 Problématique et gestion de requêtes

Les SGBD[9] actuels, comme DB2 ou ORACLE, fournissent des possibilités de gestion de requêtes diverses et variées. Citons par exemple la composition graphique de requêtes, le stockage de requêtes dans des logs, le partage de requêtes ou de logs de requêtes, la visualisation des requêtes stockées, l'analyse des requêtes stockées (notamment avant de les exécuter pour assurer de bonnes performances et donc une bonne exécution [DB209b]).

2.2.2 Les outils existants

Nous allons présenter brièvement les différents outils proposés actuellement, à savoir : DB2 Query Management Facility [DB209a], DB2 Query Patroller [DB209b] et EMS SQL Management Studio pour Oracle [EMS09].

9. *Système de Gestion de Bases de Données*

DB2 Query Management Facility
DB2 Query Management Facility (QMF) est un outil intégré qui permet d'extraire des données et de les présenter sous la forme d'un rapport. L'interface interactive de QMF permet à des utilisateurs inexpérimentés dans le domaine du traitement des données d'extraire, de créer, de mettre à jour, d'insérer ou de supprimer des données stockées dans DB2. QMF est conçu pour accéder à de gros volumes de données et partager des requêtes et des rapports d'entreprise.

DB2 Query Patroller
DB2 Query Patroller (QP) est un outil intégré qui contrôle le flux de requêtes lancées sur une base de données. Les fonctions de QP permettent d'ajuster la charge de travail de la base de données pour l'exécution rapide des requêtes et pour l'utilisation efficace des ressources système. QP permet de collecter et d'analyser des informations sur les requêtes terminées pour déterminer des tendances comme, les utilisateurs à l'origine de charges intensives ainsi que les tables et index fréquemment consultés. QP permet aussi à l'utilisateur de contrôler les requêtes soumises, de stocker les résultats des requêtes en vue de leur extraction et réutilisation ultérieures pour éviter les soumissions de requêtes répétitives. La Figure 2.6 présente un rapport contenant l'histogramme du nombre de requêtes lancées entre le 30/08/2006 et le 30/01/2007.

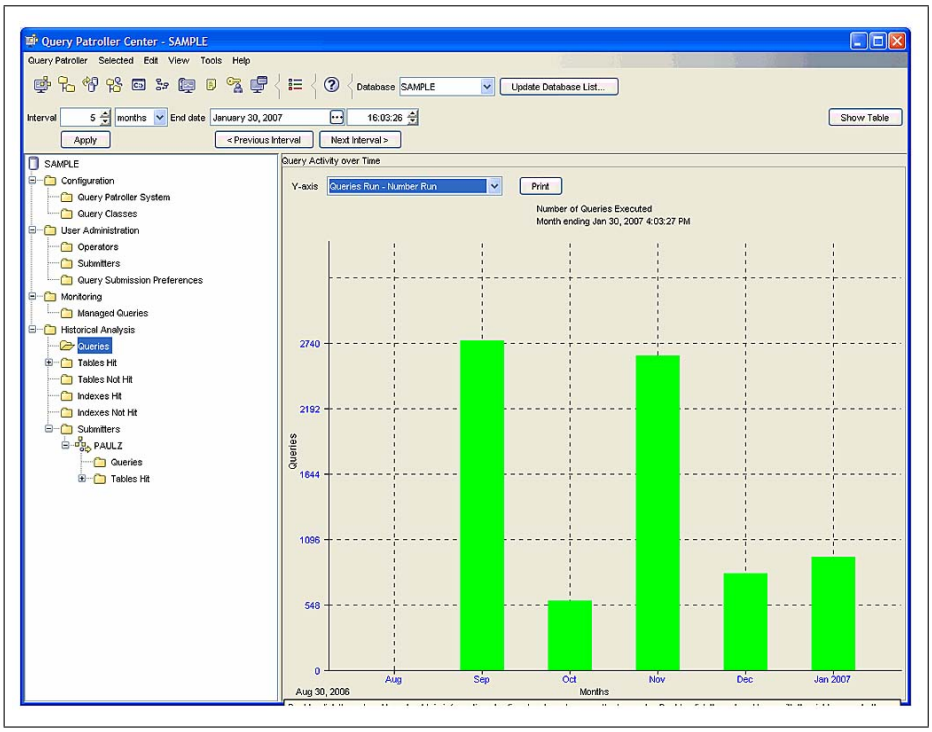


FIGURE 2.6 – Nombre de requête lancées sur une période donnée (*source* : [DB209b])

EMS SQL Management Studio pour Oracle
EMS SQL Management Studio for Oracle (SQL Studio) est une solution complète d'administration et de développement de bases de données Oracle. SQL Studio permet l'administration de bases de données, la gestion des schémas et des objets ainsi que la conception de la base de données, la

migration, l'extraction, la construction de requêtes, l'importation de données, l'exportation et la comparaison de bases de données.

2.2.3 Comparatif

Le tableau suivant récapitule les fonctionnalités et les manques des différents outils existants :

	DB2 QMF	DB2 QP	SQL Studio
Composition graphique de requêtes	×	×	×
Stockage de requêtes	×	×	×
Partage de requêtes ou de logs	×	×	×
Visualisation des requêtes stockées	×	×	×
Analyse des requêtes avant exécution		×	
Annotation de requêtes			
Recherche avancée dans les logs			

2.2.4 Synthèse

Les SGBD actuels ne permettent pas une gestion totale des logs de requêtes puisque, par exemple, aucun des outils présentés précédemment ne permet aux utilisateurs d'annoter des requêtes pour indiquer, par exemple, si une requête est intéressante, ou d'exécuter des recherches avancées sur les requêtes stockées dans les logs.

Des solutions à ce problème de gestion des logs de requêtes seront apportées dans les chapitres suivants.

2.3 Les systèmes de recommandation

Les systèmes de recommandation sont une forme particulière de filtrage de l'information visant à présenter les éléments d'information (films, musique, livres, images, pages Web, ...) qui sont susceptibles d'intéresser l'utilisateur.

2.3.1 Présentation

Les systèmes de recommandation ont été étudiés dans de nombreux domaines : les sciences cognitives, la recherche d'informations [BYRN99, Sal83, Ken71], le web [BYHM04, WBC07], le e-commerce [SKR01], le web usage mining (*Exploitation des usages du web*) [PPPS03, SCDT00, FS02, BYHMD05] et bien d'autres. Le problème de la recommandation peut se résumer par le problème d'estimation de scores pour des articles qui n'ont pas encore été vus par un utilisateur. En effet, le nombre d'articles ainsi que le nombre d'utilisateurs du système peuvent être très importants, il est, de ce fait, difficile que chaque utilisateur voit tous les articles ou que chaque article soit évalué par tous les utilisateurs. Il est donc nécessaire d'estimer les scores pour les articles non encore évalués.

Intuitivement, cette évaluation est habituellement basée sur les scores donnés par un utilisateur à d'autres articles et sur une autre information qui sera formellement décrite ci-dessous. Lorsqu'il est possible d'estimer les scores pour les articles non encore évalués, alors les articles ayant les scores les plus élevés peuvent être recommandés à l'utilisateur. Plus formellement, [AT05] formule le problème de recommandation dans le domaine du e-commerce comme suit.

Définition 2.3.1 *(Recommandation en e-commerce)*
Soit C l'ensemble de tous les utilisateurs et S l'ensemble de tous les articles possibles qui peuvent

être recommandés (comme des livres, des films, ou des restaurants). Soit u une fonction qui mesure l'utilité d'un article s à l'utilisateur c, c'est-à-dire, $u : C \times S \rightarrow \mathbb{R}$. Alors pour chaque utilisateur $c \in C$, nous voulons choisir l'article $s' \in S$ qui maximise l'utilité à l'utilisateur : $\forall c \in C, s'_c = argmax_{s \in S} u(c, s)$.

Dans les systèmes de recommandation, l'utilité d'un article est habituellement représentée par un score qui indique comment un utilisateur particulier a aimé un article particulier. Par exemple, l'utilisateur Arnaud a donné au film "Harry Potter" le score de 3 (sur 10).

Exemple 2.3.1 *Dans cet exemple, les articles sont des films que les utilisateurs Arnaud, Patrick, Marie et Elsa ont noté. Nous obtenons la matrice $C \times S$:*

$u(c,s)$	*Harry Potter*	*L'Age de glace*	*L'Age de glace 2*	*OSS 117*	*Bienvenue chez les Ch'tis*
Elsa		*8*			*7*
Marie	*9*	*8*			*6*
Arnaud	*3*	*5*		*5*	*5*
Patrick	*5*	*3*		*3*	*3*

Notons qu'une cellule (i, j) de la matrice correspond au score d'utilité assigné au film j par l'utilisateur i.

Le problème central des systèmes de recommandation vient du fait que cette utilité u n'est pas habituellement défini sur l'espace complet $C \times S$, mais seulement sur un certain sous-ensemble de celui-ci. Ceci signifie que u doit être extrapolée à l'espace entier $C \times S$. Dans les systèmes de recommandation, l'utilité est typiquement représentée par les scores et est d'abord définie sur les articles précédemment évalués par les utilisateurs. Par conséquent, le moteur de recommandation devrait pouvoir estimer / prévoir les scores des combinaisons non-évaluées d'article / utilisateur et publier des recommandations appropriées basées sur ces prévisions.

Une fois que les scores inconnus sont estimés, des recommandations réelles d'un article à un utilisateur sont émises en choisissant le score le plus élevé parmi tous les scores prévus pour cet utilisateur, selon la formule donnée Définition 2.3.1. Alternativement, nous pouvons recommander les N meilleurs articles à un utilisateur ou un ensemble d'utilisateurs à un article.

2.3.2 Catégorisation des systèmes de recommandation

Les systèmes de recommandation peuvent être classifiés selon trois approches : soit en fonction de la méthode d'estimation des scores, soit en fonction des données exploitées pour l'estimation des scores, soit en fonction de l'objectif principal du système.

Catégorisation par la méthode d'estimation des scores
L'extrapolation des scores inconnus à partir des scores connus peut être faite via une heuristique ou un modèle :

- une heuristique : en spécifiant l'heuristique qui définit la fonction d'utilité et en validant empiriquement son exécution puis, en estimant la fonction d'utilité qui optimise certains critères d'exécution.

  Exemple 2.3.2 *Considérons la matrice Utilisateur $\times$ Films de l'Exemple 2.3.1, un exemple d'heuristique est : valeur("L'Age de glace 2") = valeur("L'Age de glace")+1. A partie de cette heuristique, le système assigne au film "L'Age de glace 2", les valeurs* 9 *pour Elsa et Marie,* 6 *pour Arnaud,* 4 *pour Patrick.*

– un modèle : en utilisant la collection de scores pour "apprendre" un modèle, qui est alors employé pour faire des prévisions de scores que ce soit grâce à une approche probabiliste [BHK98], à des techniques d'apprentissage automatique [BP98] ou à des modèles statistiques [UFA+98], ...

Exemple 2.3.3 *Considérons la matrice Utilisateur × Films de l'Exemple 2.3.1, un exemple de modèle consiste à considérer que : Tous les utilisateurs ayant évalués le film "Bienvenue chez les Ch'tis" ont évalué de la même manière le film "OSS 117". A partir de ce modèle, le système assigne au film "OSS 117", les valeurs* 7 *pour Elsa et* 6 *pour Marie.*

Catégorisation par l'exploitation des données

Les systèmes de recommandation sont habituellement classifiés en fonction des scores déjà évalués, utilisés pour estimer les scores manquants [HSRF95, BS97, KK06] :

– méthode basée sur le contenu : l'utilisateur se verra recommander des éléments semblables (au sens d'une mesure de similarité entre articles) à ceux qu'il a préféré dans le passé.

Exemple 2.3.4 *Considérons la matrice Utilisateur × Films de l'Exemple 2.3.1, le système va affecter une utilité à des films qu'Elsa n'a pas encore évalué. Cette utilité est calculée en fonction des scores des films déjà évalués. Les films "L'Age de glace", "OSS 117" et "Bienvenue chez les Chti's" ont été évalués quasiment de la même manière. A partir de cette observation, le système assigne au film "OSS 117" pour Elsa, le meilleur score attribué par Elsa aux films "L'Age de glace" et "Bienvenue chez les Chti's", c'est-à-dire 8.*

– méthode collaborative : l'utilisateur se verra recommander des éléments que d'autres utilisateurs ayant des goûts et des préférences similaires (au sens d'une similarité entre utilisateurs et articles) ont aimé dans le passé.

Exemple 2.3.5 *Considérons à nouveau la matrice Utilisateur × Films de l'Exemple 2.3.1, le système va chercher les utilisateurs similaires à Elsa : ici, Marie car Marie et Elsa ont noté tous les films quasiment de la même manière. Puis, à partir de l'observation des scores que Marie a donné aux films, le système assigne au film "Harry Potter" pour Elsa, le score que Marie a donné à "Harry Potter", c'est-à-dire 9.*

– méthode hybride : combinaison des deux méthodes précédentes.

Catégorisation par l'objectif

Enfin, une autre catégorisation des systèmes de recommandation a récemment été proposée par [HG08], basée non pas sur la méthode d'estimation ou sur les données utilisées mais sur l'objectif du système de recommandation.

L'auteur introduit les notions de *filtre* et de *guide*. Le *filtre* est responsable du choix des articles intéressants/utiles parmi la grande quantité d'articles recommandables possibles, c'est-à-dire qu'il doit répondre à quels sont les articles utiles/intéressants candidats et qui vont devenir les articles recommandés. Le *guide*, quant à lui, est responsable de l'ordonnancement des articles à recommander, c'est-à-dire qu'il doit répondre à quand et comment chaque recommandation doit être montrée à l'utilisateur.

Notation 2.3.1 *(Filtre)*
Par la suite, nous notons Filtre, la sélection des recommandations candidates parmi la quantité importante de recommandations possibles.

Notation 2.3.2 *(Guide)*
Par la suite, nous notons Guide, le filtrage et l'ordonnancement des recommandations candidates.

Finalement, les systèmes de recommandation aident l'utilisateur dans sa recherche d'informations. Quel que soit le système, il peut être catégorisé selon son objectif, les données utilisées pour l'estimation des scores ou sa méthode d'estimation des scores.

2.4 Bases de données et Recommandations

2.4.1 Motivations

Les systèmes de recommandation sont plus connus pour leur utilisation dans le domaine du web, notamment sur les sites de e-commerce, où ils conseillent un client sur le choix d'un article en fonction de ses goûts. Cependant, certains travaux se sont intéressés à la recommandation dans le domaine des bases de données et ont proposé des méthodes et algorithmes afin d'aider l'utilisateur.

Bien qu'un SGBD offre les moyens d'exécuter des requêtes complexes sur de grands ensembles de données, la découverte d'informations pertinentes pour l'utilisateur reste un grand défi. En effet, en raison du grand volume de données manipulées, l'utilisateur peut, par exemple, ignorer des requêtes qui renvoient des données intéressantes ou, ne pas savoir quelle partie de la base de données interroger pour obtenir des informations utiles.
Ce problème gène clairement l'exploration des données et réduit les avantages d'employer un SGBD.

Ainsi, les bases de données doivent évoluer pour permettre de nouvelles initiatives telles que fournir un accès à l'information plus en rapport avec les goûts et les intérêts de l'utilisateur ou encore, aider les utilisateurs à trouver rapidement des données pertinentes.

Nous nous intéressons à l'exploration de bases de données dans un contexte multi-utilisateurs que nous voyons comme des sessions de requêtes et où chaque session est une séquence de requêtes lancées sur la base de données. Etant donnée une session courante, nous allons étudier les propositions de suggestion de requêtes pour la requête courante de cette session.

Dans les sections suivantes, nous proposons un parallèle entre les recommandations dans les bases de données (relationnelles ou multidimensionnelles) et les recommandations en e-commerce. Puis, nous présentons les différents travaux existants sur l'aide à l'exploration dans les bases de données. Enfin, nous en proposons un comparatif.

2.4.2 Parallèle entre e-commerce et recommandations de requêtes en bases de données

Dans un premier temps, nous faisons la distinction entre recommandation de requêtes et personnalisation de requêtes, puis, nous présentons les notions inhérentes à notre contexte, à savoir ce que nous entendons par recommandations dans les bases de données.

2.4.2.1 Recommandation vs. personnalisation

Un point important est de différencier la recommandation de requête de la personnalisation de requête.

Personnalisation
Ainsi, dans [KI04] et [BGM+05], la personnalisation de requête est considérée comme un ajout de contraintes, issues du profil de l'utilisateur, à une requête donnée et se traduit par l'ajout de conditions de sélection. La démarche de personnalisation, à partir d'une requête Q sur une instance de la base de données, renvoie une requête $Q*$ telle que $Q* \subseteq Q$ au sens de l'inclusion de requêtes MDX telle que définie dans [BGMM06].

Exemple 2.4.1 *Supposons que la requête Q à personnaliser concerne les ventes de véhicules à Tours en 2007, et supposons que les préférences de l'utilisateur se portent uniquement sur les véhicules bleus ou rouges, une requête personnalisée $Q*$ de la requête Q sera, par exemple, les ventes de véhicules bleus ou rouge à Tours en 2007.*

Recommandation
De notre point de vue, une requête recommandée est une requête existante, par exemple issue d'un ensemble de requêtes déjà posées sur la base de données ou une requête calculée, par exemple à partir des requêtes déjà posées sur la base de données. La requête recommandée peut, par conséquent, être complètement différente de la requête initiale car elles ne vont pas forcément avoir le même schéma. Contrairement à la personnalisation, la démarche de recommandation, à partir d'une requête initiale Q sur une instance de la base de données, renvoie une requête $Q*$ telle que $Q \nsubseteq Q*$.

De plus, notons que dans notre contexte, les requêtes recommandées pourront être personnalisées et réciproquement.

Exemple 2.4.2 *Supposons que la requête Q de l'utilisateur concerne les ventes de véhicules à Tours en 2007, une requête recommandée pour la requête Q pourrait être, par exemple, la requête $Q*$ concernant les ventes de véhicules en région Centre en 2008.*
Notons que la sélection sur l'année ayant changée, Q (la requête initiale) $\nsubseteq Q*$ (la recommandation).

2.4.2.2 Bases de données et e-commerce

Une recommandation en e-commerce, comme définie Définition 2.3.1, est l'article $a \in A$ (ensemble de tous les articles (films, livres, ...)) tel que l'utilité pour un utilisateur $c \in C$ (ensemble de tous les utilisateurs) est maximale. Par analogie, nous pouvons définir une recommandation en bases de données comme étant une requête telle que son utilité pour une session $s \in S$ (ensemble de toutes les sessions possibles) est maximale.
Afin de proposer une définition la plus générale possible, notons que, pour une instance donnée de base de données, un ensemble de tuples peut être vu comme une requête.

Définition 2.4.1 *(Recommandation en bases de données)*
Soit Q_T l'ensemble de toutes les requêtes possibles sur la base de données[10]*, et S l'ensemble de toutes les sessions possibles sur la base de données, étant donné un log de sessions de requêtes, une base de données et une session courante, et soit u une fonction qui mesure l'utilité d'une requête q pour une session s, c'est-à-dire, $u : S \times Q_T \to \mathbb{R}$. Alors pour chaque session $s \in S$, la requête recommandée $q' \in Q_T$ est celle qui maximise l'utilité pour la session : $\forall s \in S, q'_s = argmax_{q \in Q_T} u(s, q)$.*

Exemple 2.4.3 *Dans cet exemple, les sessions et requêtes sont présentées Annexe A. Nous obtenons la matrice $S \times Q$:*

10. En pratique, comme en e-commerce, nous considérons un sous-ensemble fini de cet ensemble.

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	3		5		9			
s_2	2			5	5	8		
s_3	2		5				6	9
s_c	2	3		5				

Notons qu'une cellule (i, j) de la matrice correspond au score d'utilité assigné à la requête j pour la session i.

2.4.3 Les méthodes existantes d'aide à l'exploration de bases de données

Dans notre étude bibliographique, nous avons recensé des travaux qui suggèrent une requête, un ensemble de requêtes ou un ensemble de tuples pour aider l'exploration de bases de données.
Ces travaux ne sont pas tous décrits par leurs auteurs comme étant des travaux relatifs à la recommandation, notons cependant que nous les décrivons selon le point de vue issu de la Définition 2.4.1.
Dans cette section, nous nous intéressons à l'exploration de bases de données en général, c'est-à-dire dans les bases de données relationnelles ou multidimensionnelles, bien que le terme *exploration* soit plus approprié aux cubes de données.
Enfin, dans les méthodes étudiées, lorsqu'il s'agira de calculer ou de sélectionner une requête, un ensemble de tuples ou un ensemble de requêtes, nous verrons cela comme le calcul ou la sélection d'un ensemble de requêtes.

2.4.3.1 Critères d'étude des algorithmes de recommandation

Nous présentons dans cette section les différents travaux traitant de la recommandation pour l'exploration de bases de données. Nous étudions les algorithmes relatifs aux méthodes existantes. De ce fait, nous nous intéressons aux caractéristiques des algorithmes. Ici, les critères selon lesquels nous allons étudier les algorithmes existants sont : les entrées, les sorties et la démarche de l'algorithme. Notre objectif est de présenter une synthèse de ces méthodes (résumée dans le tableau 2.7).

Les entrées de l'algorithme
Chaque algorithme présenté a besoin d'un certain nombre de paramètres au choix parmi les cinq suivants :

- Le log : contenant un ensemble de sessions, comme défini Définition 2.1.8,
- Le schéma de la base de données,
- Une instance de la base de données,
- La session courante : contenant la session (comme définie Définition 2.1.7) en cours de l'utilisateur,
- Le profil d'utilisation : composé soit du profil de l'utilisateur, soit d'un profil global pour l'exploitation de la base de données.

Les sorties de l'algorithme
Un algorithme peut retourner des résultats de types différents. Dans le cas de la recommandation dans les bases de données, il peut s'agir de :

- Une requête : l'utilisateur n'a plus qu'à la lancer s'il la trouve intéressante,
- Un ensemble de requêtes, éventuellement ordonné,
- Un ensemble de tuples : c'est-à-dire une partie de la base de données ou du cube considérée intéressante et assimilable à une requête.

Les étapes de recommandation
Un algorithme de recommandation peut être décomposé en trois parties : l'approche de l'évaluation des scores utilisée, le filtre et le guide, comme présentés Section 2.3.2.

Approche
Dans les travaux étudiés, il peut s'agir d'une approche :
- basée sur le contenu,
- collaborative,
- hybride.

Filtre
Cette partie se décompose en trois sous-parties :
- L'estimation des scores :
  - La base : une heuristique ou un modèle, comme présenté Section 2.3.1,
  - La technique de calcul : selon la méthode de recommandation utilisée (Section 2.3.2), différentes techniques de recommandation existent ([AT05]),
- La génération : est obtenue soit par sélection de requêtes ou de tuples existants dans les entrées de l'algorithme, soit par calcul de requêtes ou de tuples à partir des paramètres d'entrée.

Guide
Cette partie traite l'ordonnancement des résultats s'il y en a plusieurs.

2.4.3.2 Étude comparative des algorithmes

Dans cette section, nous comparons sept méthodes qui n'abordent pas le problème de la même manière. Ainsi, pour chacune d'entre elles, afin de les présenter de manière homogène, nous nous positionnons dans le cadre de l'exploitation de cubes de données. Puis, nous illustrons chaque algorithme en donnant un exemple de fonction d'utilité ainsi qu'un exemple de requête recommandée pour une session courante et une requête courante données. Les méthodes présentées ici sont celles de Sapia ([Sap99, Sap00]), Sarawagi ([Sar99, Sar00, SS01]), Jerbi et al. ([JRTZ09]), Cuppens et Demolombe ([CD91]), Chatzopoulou et al. ([CEP09]), Cariou et al. ([CCD+08]) et Yang et al. ([YPS09]). Comme remarqué précédemment, il n'existe pas beaucoup de travaux traitant de la recommandation pour l'exploration de bases de données. Notre choix de présentation s'est donc porté sur des méthodes exploitant un profil d'utilisation ([CD91], [JRTZ09]), celles basées sur une analyse pilotée par la découverte ([Sar99, Sar00, SS01], [CCD+08]) et celles exploitant les logs de requêtes ([Sap99, Sap00], [CEP09],[YPS09]).

2.4.3.2.1 Exploitation du profil

Recommandations et Bases de données coopératives [CD91]

Nous décrivons d'abord la méthode proposée par Cuppens et Demolombe ([CD91]) qui construit une requête pour une session donnée selon le principe suivant. A partir du schéma de la base de données relationnelle, de la session courante et d'un profil d'utilisation (contenant des prédicats, vus comme des règles de réécriture, sur les attributs de la base de données), il s'agit d'intégrer les prédicats du profil d'utilisation à la requête courante. La nouvelle requête ainsi obtenue par transformation de la requête courante est retournée à l'utilisateur.

Cette méthode de réponse à une requête donnée peut être vue comme une méthode de recommandation basée sur le contenu qui calcule l'utilité d'une requête pour une session donnée. L'utilité maximale est obtenue pour la requête qui correspondra à la requête initiale enrichie avec tous les prédicats applicables. Un prédicat du profil est applicable à une requête s'il existe un prédicat de la requête qui implique logiquement le prédicat du profil. Cette méthode utilise une heuristique pour l'extrapolation des scores, consistant à assigner aux scores inconnus, le nombre de règles du profil d'utilisation qu'il faut appliquer pour enrichir la requête courante. Cette méthode a pour objectif d'ajouter de l'information à une requête donnée en la transformant, par exemple, en relâchant ou en ajoutant des contraintes sur les attributs de la requête, ou en ajoutant des attributs à la requête.

Remarque : Les auteurs ne présentent pas cette méthode comme une méthode de recommandation mais elle permet tout de même de recommander une requête en intégrant un profil d'utilisation. Cependant, les requêtes antérieures de l'utilisateur ne sont pas prises en considération dans le calcul de la recommandation.

Exemple 2.4.4 *Nous illustrons la méthode de Cuppens et Demolombe en cherchant quelle requête pourrait être recommandée à la suite de la requête q_3 de la session s_c présentée Annexe A en considérant que le profil d'utilisation P_1 indique que la ville de Blois est préférée à Tours et seules les données de l'année 2008 sont intéressantes.*
Voici la représentation sous forme de prédicats de q_3 :

$$(CouleurVéhicules(Rouge) \vee CouleurVéhicules(Bleu)) \wedge Ville(Tours) \wedge Montant(x)$$

Voici la représentation sous forme de règles de réécriture du profil d'utilisation P_1.
- *r_1 : $Ville(Tours) \rightarrow (Ville(Tours) \vee Ville(Blois)) \wedge Année(2008)$*
- *r_2 : $Année(y) \rightarrow Année(2008)$*

Comme nous proposons de voir cette méthode comme une méthode de recommandation, nous considérons que la fonction d'utilité calcule le nombre de règles de réécriture appliquées sur la requête. Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	$q_3 + r_1$	$q_3 + r_2$	$q_3 + r_1 + r_2$
$s_c = q_3$	*1*	*1*	*2*

où l'utilité de la requête q_3 enrichie d'une règle, noté q_3+r_1, vaut 1 et lorsque q_3 est enrichie de deux règles, l'utilité vaut 2. L'utilité maximale est donc obtenue lorsque q_3 est enrichie de deux règles : $q_3 + r_1 + r_2$.
En intégrant r_1 et r_2 à la requête initiale, nous obtenons la requête recommandée :

```
SELECT  Crossjoin({[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]},
        {[Temps].[AllT].[2008]})                                  ON COLUMNS,
        {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours],
        [Géographie].[AllG].[Centre].[LoirEtCher].[Blois]}        ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]} ;
```

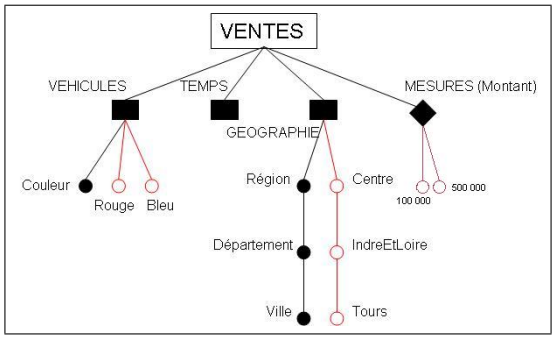
Recommandations par l'exploitation du profil [JRTZ09]

Nous décrivons ici la méthode de recommandation proposée par Jerbi et al. ([JRTZ09]). Cette méthode calcule l'utilité d'une requête pour un utilisateur donné selon le principe suivant. A partir du schéma et d'une instance de la base de données multidimensionnelle, de la session courante et du

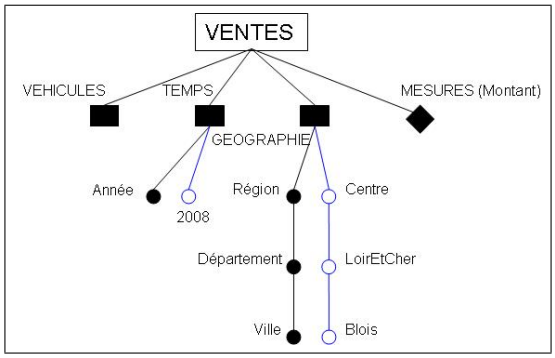
profil de l'utilisateur, il s'agit de représenter sous forme d'arbre la requête courante de l'utilisateur, de faire de même avec son profil utilisateur, puis d'appliquer un algorithme de *Tree Matching* pour comparer les deux arbres et ainsi ajouter à l'arbre de la requête, les parties de l'arbre des préférences, à savoir les nœuds et les arcs, qui n'y apparaissent pas. Enfin, la requête résultant de la transformation de l'arbre de la requête initiale est proposée comme recommandation à l'utilisateur. Cette méthode de recommandation de requêtes $OLAP$ basée sur le contenu et utilisant une heuristique pour l'extrapolation des scores consistant à assigner aux scores inconnus le nombre de nœuds de l'arbre des préférences qui apparaissent dans la requête finale, a pour objectif de raffiner et / ou d'enrichir la requête courante de l'utilisateur en fonction de ses préférences (issues du profil de l'utilisateur).

Remarque : Cette méthode de recommandation permet aussi de faire de la personnalisation. Cependant, les requêtes précédentes de l'utilisateur ne sont pas prises en compte dans le calcul de la recommandation.

Exemple 2.4.5 *Nous illustrons la méthode de Jerbi et al. en cherchant quelle requête pourrait être recommandée à la suite de la requête q_3 de la session s_c présentée Annexe A en considérant que l'utilisateur P_1 préfère la ville de Blois et l'année 2008 et que l'utilisateur P_2 préfère l'année 2009. Voici la représentation sous forme d'arbre de la requête q_3 :*



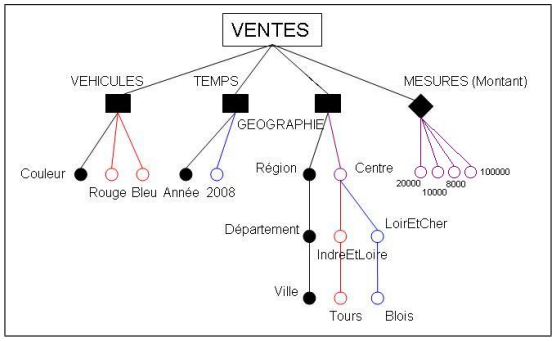
Voici la représentation sous forme d'arbre des préférences de l'utilisateur P_1 :



Nous proposons de considérer que la fonction d'utilité calcule le nombre de nœuds de l'arbre des préférences présents dans la requête potentiellement recommandable. Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	$q_3 + P_1$	$q_3 + P_2$
$S_c = q_3(P_1)$	*4*	*1*
$S_c = q_3(P_2)$	*0*	*1*

où, par exemple, pour un utilisateur ayant le profil P_1 et lançant la requête q_3, si l'arbre de q_3 est fusionné avec l'arbre de P_1 alors, les nœuds "2008", "Centre", "LoirEtCher" et "Blois" apparaissent dans la requête recommandable, tandis que si l'arbre de q_3 est associée à l'arbre de P_2 alors, seul le nœud "2009" apparait. Ainsi, l'utilité maximale est obtenue pour $q_3 + P_1$ pour un utilisateur ayant pour profil P_1 et pour $q_3 + P_2$ pour un utilisateur de profil P_2.
En appliquant l'algorithme de Tree-matching entre l'arbre représentant q_3 et l'arbre des préférences de P_1, nous obtenons l'arbre suivant :



qui correspond à la requête recommandée :

SELECT Crossjoin({[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]}, {[Temps].[AllT].[2008]}) ON COLUMNS,
{[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours], [Géographie].[AllG].[Centre].[LoirEtCher].[Blois]} ON ROWS
FROM [Ventes]
WHERE {[NomMesures].[Montant]} ;

2.4.3.2.2 Analyse pilotée par la découverte

Opérateurs pour la découverte de faits inattendus [Sar99, Sar00, SS01]

Nous décrivons ici la méthode proposée par Sarawagi ([Sar99, Sar00, SS01]). A partir du schéma et d'une instance de la base de données multidimensionnelle et de la requête courante, il s'agit de trouver les cellules du cube de données qui permettent d'expliquer des anomalies constatées grâce aux opérateurs $DIFF$, $EXCEP$, $RELAX$ et $INFORM$ qui ont en paramètres les cellules à expliquer.

- L'opérateur $DIFF$ est une sorte de "drill-down". Il examine les sous-cubes qui détaillent les deux cellules passées en paramètres et retourne les paires de cellules constituant les différences les plus pertinentes.
- L'opérateur $EXCEP$ vérifie si la tendance, observée entre les deux cellules passées en paramètres, est confirmée et retourne les paires de cellules constituant les exceptions.

- L'opérateur *RELAX* est une sorte de "roll up". Il vérifie si la tendance, vue entre les deux cellules passées en paramètres, est confirmée à plus haut niveau. Il retourne les paires de cellules des niveaux supérieurs dans lesquels la tendance est retrouvée.
- L'opérateur *INFORM* tente de trouver les parties du cube qu'un utilisateur trouverait les plus surprenantes à partir de ce que l'utilisateur a vu dans sa session et d'un modèle du cube basé sur l'entropie maximale.

Enfin, tous les ensembles de tuples correspondants aux cellules explicatives des anomalies sont retournées. Dans notre contexte, l'union des ensembles de tuples permet de retourner une requête recommandée.
Cette méthode n'est pas présentée comme telle par ses auteurs mais elle peut être vue comme une méthode de recommandation basée sur le contenu qui calcule l'utilité d'un ensemble de tuples quel que soit l'utilisateur, c'est-à-dire que, l'utilité est la même pour tous les utilisateurs. Elle utilise un modèle pour l'extrapolation des scores et cherche à expliquer des anomalies constatées lors de l'observation du résultat d'une requête lancée sur un cube de données.

Remarque : Les auteurs ne présentent pas cette méthode comme une méthode de recommandation mais elle permet de proposer une requête à l'utilisateur en faisant l'union des tuples renvoyés. Cependant, cette méthode s'applique sur le résultat de la requête (qui change selon l'instance de la base de données).

Exemple 2.4.6 *Nous illustrons la méthode de Sarawagi en cherchant quels ensembles de tuples, c'est-à-dire quelle requête pourrait être recommandée. Les requêtes sont lancées sur le cube de données MesVentes dont le schéma ainsi qu'une instance sont présentés Figure 2.3 et Figure 2.2 de la Section 2.1.1. Nous illustrons chacun des opérateurs proposés par Sarawagi ([Sar99, Sar00, SS01]), à savoir INFORM, DIFF, EXCEP et RELAX.*

1. *Opérateur INFORM*
   Soit la requête q_1 lancée sur le cube de données :

   SELECT {[Géographie].[AllG].[Centre]} *ON ROWS,*
   {[Temps].[AllT].[2005]} *ON COLUMNS*
   FROM [Ventes]
   WHERE {[NomMesures].[Quantité]} ;

   Le tableau croisé correspondant est :

VENTES (Quantité)	2005
Centre	551

   Sachant que la quantité des ventes en région Centre atteint 551*, l'entropie maximale prévoit que chaque département de cette région atteindra* $\frac{551}{5} = 110.2$.
   *Nous proposons de voir cette méthode comme une méthode de recommandation, nous considérons donc que la fonction d'utilité calcule la valeur absolue de la différence entre la valeur attendue (*110.2*) et la valeur obtenue.*
   Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	$q_{LoirEtCher}$	$q_{IndreEtLoire}$	q_{Indre}	$q_{EureEtLoir}$	q_{Loiret}
$s_c = q_1$	109.2	39.8	10.2	89.8	10.2

L'utilité maximale est obtenue pour $q_{LoirEtCher}$.
Une requête pouvant être proposée à la suite de q_1 serait donc :

```
SELECT  {[Géographie].[AllG].[Centre].[LoirEtCher]}  ON ROWS,
        {[Temps].[AllT].[2005]}                      ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

2. *Opérateur DIFF*
Soit la requête q_2 lancée sur le cube de données :

```
SELECT  {[Géographie].[AllG].[Centre].Children}  ON COLUMNS,
        {[Temps].[AllT]}                         ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

Le tableau croisé correspondant est :

VENTES (Quantité)	LoirEtCher	IndreEtLoire	Indre	EureEtLoir	Loiret
AllT	12	750	690	860	890

Le nombre de ventes en Indre est inférieur à celui en Eure-Et-Loir : nous sélectionnons ces deux cellules. Le système va alors détailler la dimension Temps pour tenter de trouver une explication.
Nous considérons que la fonction d'utilité calcule la différence entre la valeur en Eure-Et-Loir et la valeur en Indre pour chaque année. Si cette valeur est négative, alors l'utilité est nulle.
Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_{2005}	q_{2006}	q_{2007}	q_{2008}
$s_c = q_2$	100	90	0	0

L'utilité maximale est obtenue pour q_{2005}. Ce qui signifie que la tendance se retrouve pour l'année 2005.
Une requête pouvant être proposée à la suite de q_2 serait :

```
SELECT  {[Géographie].[AllG].[Centre].[EureEtLoir],
         [Géographie].[AllG].[Centre].[Indre]}   ON ROWS,
        {[Temps].[AllT].[2005]}                  ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

3. *Opérateur EXCEP*
Soit, de nouveau, la requête q_2 lancée sur le cube de données. Le tableau croisé correspondant est :

VENTES (Quantité)	LoirEtCher	IndreEtLoire	Indre	EureEtLoir	Loiret
AllT	12	750	690	860	890

Le nombre de ventes en Indre est inférieur à celui en Eure-Et-Loir : nous sélectionnons ces deux cellules. Le système va alors détailler la dimension Temps pour tenter de trouver une exception.
Nous considérons que la fonction d'utilité calcule la différence entre la valeur en Indre et la

valeur en Eure-Et-Loir. Si cette valeur est négative, alors l'utilité est nulle.

Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_{2005}	q_{2006}	q_{2007}	q_{2008}
$s_c = q_2$	0	0	20	0

L'utilité maximale est obtenue pour q_{2007}. Ce qui signifie que la tendance s'inverse pour l'année 2007.
Une requête pouvant être proposée à la suite de q_2 serait :

```
SELECT  {[Géographie].[AllG].[Centre].[EureEtLoir],
        [Géographie].[AllG].[Centre].[Indre]}        ON ROWS,
        {[Temps].[AllT].[2007]}                      ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

4. *Opérateur RELAX*
Soit la requête q_3 lancée sur le cube de données :

```
SELECT  {[Géographie].[AllG].[Gironde]}   ON ROWS,
        {[Temps].[AllT].Children}         ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

Le tableau croisé correspondant est :

VENTES (Quantité)	2005	2006	2007	2008
Gironde	2	2	5	4

Le nombre de ventes de véhicules en 2006 est inférieur à celui de 2007 : nous sélectionnons ces deux cellules. Le système va alors agréger les départements de la dimension Géographie et les couleurs de véhicules de la dimension Véhicules pour tenter de trouver une explication.
Nous considérons que la fonction d'utilité calcule la différence entre la valeur en 2007 et la valeur en 2006. Si cette valeur est négative, alors l'utilité est nulle.
Une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_{AllV}	$q_{Aquitaine}$
$s_c = q_3$	0	3

L'utilité maximale est obtenue pour $q_{Aquitaine}$. Ce qui signifie que la tendance se retrouve pour toute la région Aquitaine en général.
Une requête pouvant être proposée à la suite de q_3 serait :

```
SELECT  {[Géographie].[AllG].[Aquitaine]}                ON ROWS,
        {[Temps].[AllT].[2006], [Temps].[AllT].[2007]}   ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]} ;
```

Recommandations basées sur l'association entre dimensions [CCD+08]

Nous décrivons ici la méthode de recommandation proposée par Cariou et al.([CCD+08]). Cette méthode calcule l'utilité d'un ensemble de tuples quel que soit l'utilisateur, c'est-à-dire que l'utilité est la même pour tous les utilisateurs. Le principe de cette approche "pas à pas" est le suivant. A partir du schéma et d'une instance de la base de données multidimensionnelle et de la session courante, il s'agit de mesurer le degré d'intérêt de chaque dimension non encore détaillée dans la tranche [11] de cube interrogée. Le degré d'intérêt est basé sur la quantité d'information fournie par la partition obtenue en ajoutant la dimension. Plus formellement, le degré d'intérêt est basé sur la déviation, au sens du T de Tschuprow [12], des valeurs des cellules observées par rapport à l'hypothèse du modèle d'indépendance. Puis les dimensions sont ordonnées par ordre décroissant d'intérêt.

L'utilisateur est libre de choisir la dimension à détailler mais le système propose de détailler celle qui obtient le meilleur degré. Notons que le degré d'intérêt des dimensions restantes dépend des dimensions déjà détaillées. Dans notre contexte, nous considérons que la requête recommandée contient les fils du niveau le plus haut de la dimension ayant le plus fort degré d'intérêt.

Cette méthode de recommandation basée sur le contenu et utilisant une heuristique pour l'extrapolation des scores donnant aux scores inconnus la valeur du T de Tschuprow entre les dimensions déjà détaillées et les autres, a pour objectif de guider les utilisateurs dans leur navigation au travers de données multidimensionnelles en leur proposant, à chaque étape de leur analyse, les dimensions à détailler en premier.

Remarque : Cette méthode de recommandation utilise les choix antérieurs de l'utilisateur pour proposer la dimension suivante à détailler. Cependant, les auteurs ne recommandent pas de requête.

Exemple 2.4.7 *Nous illustrons la méthode de Cariou et al. en cherchant quelle dimension pourrait être détaillée et donc quelle requête proposée à la suite d'une requête. Nous utilisons le cube de données MesVentes dont le schéma ainsi qu'une instance sont présentés Figure 2.3 et Figure 2.2 de la Section 2.1.1. Notons que seule la mesure 'Quantité' est utilisée.*
L'ensemble des dimensions du cube est D={Véhicules, Géographie,Temps}=$\{d_1, d_2, d_3\}$.
Supposons que la requête courante q_1 *est :*

SELECT {[Véhicules].[AllV].Children} ON ROWS
FROM [Ventes]
WHERE {[NomMesures].[Quantité]}

Nous supposons que la fonction d'utilité calcule le T de Tschuprow entre les dimensions déjà détaillées dans la requête et celles qui ne le sont pas encore, une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_{d_2}	q_{d_3}
$s_c = q_1$	*0.0135*	*0.07*

où, comme seule la dimension d_1 *est détaillée dans* q_1*, l'utilité de la requête permettant de détailler* d_2 *:* q_{d_2} *est la valeur du T de Tschuprow entre* d_1 *et* d_2*. L'utilité maximale est obtenue pour* q_{d_3}*,*

11. Une tranche de cube consiste à fixer la valeur d'un attribut pour une dimension donnée.

12. Dans notre contexte, le T de Tschuprow est un test statistique non paramétrique permettant de calculer l'association entre deux dimensions. C'est une normalisation par les degrés de liberté du test du χ^2 : $T(d_i, d_j) = \sqrt{\frac{\chi^2(d_i,d_j)}{n\sqrt{ddl}}}$ où n est la somme des valeurs de la mesure de la tranche et ddl est le degré de liberté, à savoir le produit (des nombres de membres dans la hiérarchie de chacune des deux dimensions - 1).

c'est-à-dire la requête qui va permettre de détailler la dimension d_3 : Temps.
Une requête pouvant être recommandée à la suite de q_1 est :

```
SELECT  {[Véhicules].[AllV].Children}  ON ROWS,
        {[Temps].[AllT].Children}      ON COLUMNS
FROM    [Ventes]
WHERE   {[NomMesures].[Quantité]}
```

2.4.3.2.3 Exploitation des logs

Modèle Markovien pour la prévision de requêtes [Sap99, Sap00]

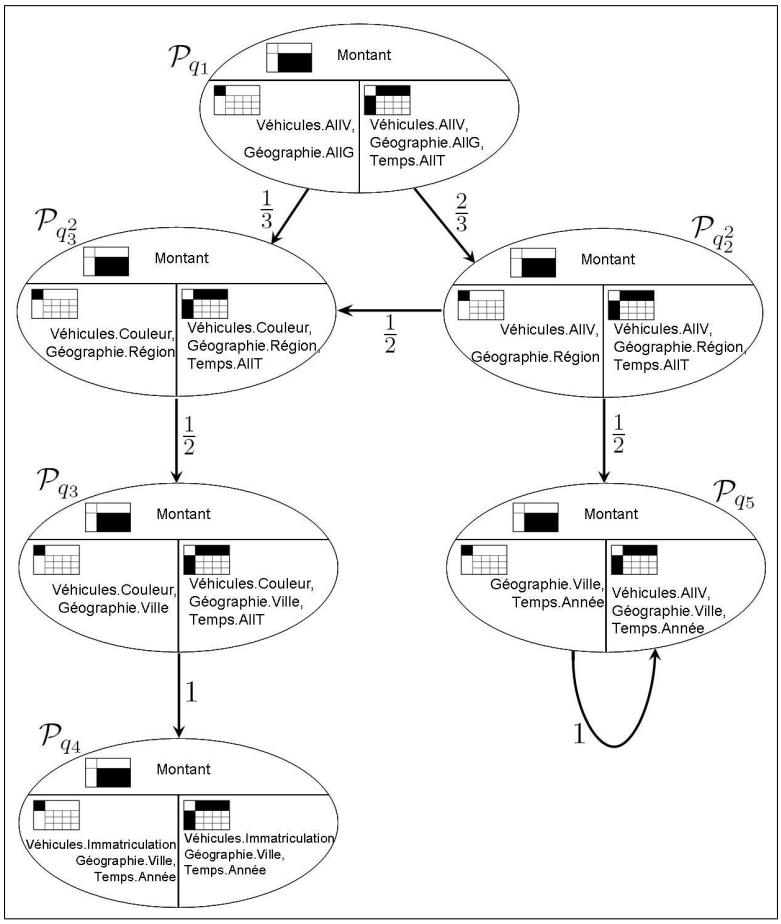
Nous décrivons ici la méthode proposée par Sapia ([Sap99, Sap00]). A partir d'un log de sessions de requêtes *OLAP*, du schéma et d'une instance de la base de données multidimensionnelle et de la session courante, il s'agit de repérer pour chaque requête du log, les sélections et les attributs sur lesquels ont lieu les projections. Puis il s'agit de construire le prototype de chaque requête des sessions du log, qui détaille la mesure, les niveaux des dimensions des membres sélectionnés et les niveaux des dimensions des membres projetés. Les prototypes de requêtes sont comparés grâce à une distance qui donne le nombre d'opérations pour passer d'un prototype de requête à un autre. Pour chaque session du log, chaque requête est remplacée par le prototype qui lui correspond. Puis un modèle de Markov permet de prédire la probabilité d'"apparition" de chaque prototype de requête dans l'ensemble des sessions du log. Enfin, le système fait de même avec la session courante où chaque requête est remplacée par le prototype qui lui correspond. Une telle session est alors comparée au modèle de Markov. Les prototypes de requêtes ayant la plus forte probabilité d'apparition servent de base à la requête recommandée.

Cette méthode collaborative de préchargement[13] de requêtes *OLAP* peut être vue comme une méthode de recommandation, basée sur un modèle probabiliste (modèle de Markov) qui calcule l'utilité d'une requête pour une session donnée. Elle a pour objectif initial de prétraiter les données de la base en prévoyant le prochain prototype de requête afin d'améliorer les performances des systèmes *OLAP*. Dans notre contexte, chaque prototype de requête prédit permet de proposer plusieurs recommandations.

Remarque : L'auteur ne présente pas cette méthode comme une méthode de recommandation mais elle permet de recommander une requête en allant au delà du préchargement, qui peut être vu comme une recommandation pour le système, en proposant des requêtes à l'utilisateur. Cette méthode prend en compte le séquencement de requêtes ainsi que le fait qu'une base de données est exploitée par plusieurs décideurs. Cependant, bien qu'il s'agisse de requêtes multidimensionnelles, la méthode ne traite pas de requêtes *MDX*.

Exemple 2.4.8 *Nous illustrons la méthode de Sapia en cherchant quelle requête pourrait être recommandée à la suite de la requête q_1 de la session s_c présentée Annexe A en cherchant parmi les requêtes des sessions s_1, s_2 et s_3 du log présenté Annexe A.*
Le modèle de Markov correspondant est le suivant :

13. traduction de l'anglais *prefetching*



Chaque nœud de ce graphe correspond à un prototype de requête. Par exemple, le premier nœud est le prototype de la requête q_1 *:* $\mathcal{P}_{q_1}$*={{Montant},{Véhicules.AllV,Géographie.AllG},{Véhicules.AllV, Géographie.AllG, Temps.AllT}}.*
Notons que le prototype $\mathcal{P}_{q_6}$ *a disparu car* q_5 *et* q_6 *ont le même prototype, c'est-à-dire que la distance entre le prototype de* q_5 *et celui de* q_6 *est nulle :* $|\mathcal{P}_{q_5} - \mathcal{P}_{q_6}| = 0$.
Nous proposons de voir cette méthode comme une méthode de recommandation, nous considérons donc que la fonction d'utilité calcule la probabilité d'obtenir tel prototype de requête à la suite de la session contenant les prototypes des requêtes de la session courante : s_c=q_1*. Notons que les requêtes ayant le même prototype ont la même utilité. Une partie de la matrice d'utilité, concernant la session courante, est :*

$u(s,q)$	$q_{\mathcal{P}_{q_3^2}}$	$q_{\mathcal{P}_{q_2^2}}$	$q_{\mathcal{P}_{q_3}}$	$q_{\mathcal{P}_{q_4}}$	$q_{\mathcal{P}_{q_5}}$	$q_{\mathcal{P}_{q_6}}$
s_c=q_1	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$

où, par exemple, l'utilité des requêtes ayant le même prototype que q_3^2 *:* $q_{\mathcal{P}_{q_3^2}}$ *est la probabilité d'apparition du prototype* $\mathcal{P}_{q_3^2}$ *à la suite de la requête* q_1 *de prototype* $\mathcal{P}_{q_1}$*. L'utilité maximale est obtenue pour* $q_{\mathcal{P}_{q_2^2}}$*, c'est-à-dire pour les requêtes ayant le même prototype que* q_2^2.
Ainsi, à la suite de la requête q_1 *de la session courante, la probabilité de proposer une requête ayant le même prototype que* q_2^2 *est de* $\frac{2}{3}$.
Un exemple de requête recommandée ayant un tel prototype est :

SELECT *{[Véhicules].[AllV].Members}* *ON ROWS,*
{[Géographie].[AllG].[Région].Members} *ON COLUMNS*
FROM *[Ventes]*
WHERE *{[NomMesures].[Montant]} ;*

Recommandations basées sur la contribution des tuples aux sessions [CEP09]

Nous décrivons ici la méthode de recommandation proposée par Chatzopoulou et al. ([CEP09]) qui calcule l'utilité d'une requête pour la session courante selon le principe suivant. A partir d'un log de sessions de requêtes, du schéma et d'une instance de la base de données relationnelle et de la session courante, il s'agit de construire le vecteur pondéré S_i des tuples apparaissant dans chaque session (S_0 pour la session courante). Puis, il s'agit de prédire le vecteur pondéré de tuples de la session courante S_0^{pred} où chaque poids signifie l'importance prédite du tuple. S_0^{pred} est obtenu en comparant S_0 et les S_i grâce à une fonction basée sur la similarité, au sens du cosinus, entre les vecteurs. Enfin, une des solutions proposées consiste à assigner à chaque requête Q du log, une "importance" comme étant la valeur de la similarité, toujours au sens du cosinus, entre le vecteur pondéré S_Q des tuples de la requête et S_0^{pred}. Les requêtes obtenant les meilleures "importances", c'est-à-dire celles qui couvrent au mieux les tuples importants de S_0^{pred} sont recommandées à l'utilisateur.
Cette méthode de recommandation peut être collaborative, basée sur le contenu ou hybride et utilise une heuristique pour l'extrapolation des scores calculant la similarité au sens du Cosinus entre les requêtes du log et la requête prédite représentée par le vecteur S_0^{pred}. Cette méthode a pour objectif de sélectionner, parmi les requêtes du log, celles qui contiennent les tuples considérés comme les plus importants.

Remarque : Cette méthode de recommandation a pour intérêt de permettre différentes approches de recommandation. Cependant, elle ne prend pas en compte le séquencement de requêtes durant la session et se limite à la dernière requête posée.

Exemple 2.4.9 *Nous illustrons la méthode de Chatzopoulou et al. en cherchant, parmi le log de sessions de requêtes présenté Annexe A, quelles requêtes pourraient être recommandées à la suite de la session s_c présentée également Annexe A.*
Les requêtes sont lancées sur le cube de données MesVentes dont le schéma ainsi qu'une instance sont présentés Figure 2.3 et Figure 2.2 de la Section 2.1.1.
Cette méthode manipule des vecteurs de tuples. Il faut donc connaître tous les tuples de la base de données. Puis, il s'agit de mesurer la contribution de chaque tuple à la requête. Pour connaître le nombre de tuples total, il existe deux manières de faire : soit nous prenons uniquement les tuples présents dans la table de faits et les tables de dimensions, soit nous calculons l'ensemble des références du cube interrogeable. Dans cet exemple, nous calculons la contribution de chaque référence du cube à la requête. Nous optons donc pour la seconde manière de faire.
L'instance présentée Figure 2.2 de la Section 2.1.1 contient $1+4+4+12+12=33$ *membres sur la dimension TEMPS,* $1+2+6+8+8=25$ *membres sur la dimension GÉOGRAPHIE et* $1+4+12=17$ *membres sur la dimension VÉHICULES. Nous obtenons donc* $33*25*17=14025$ *tuples* $t_i, i \in [1, 14025]$.

Voici les références résultats de chacune des requêtes de la session s_1 du log sur la dite instance :
- $q_1 : \{\langle Montant, AllV, AllG, AllT\rangle\} = \{t_1\}$

- $q_2^2 : \{\langle Montant, Rouge, Centre, AllT\rangle, \langle Montant, Blanc, Centre, AllT\rangle, \langle Montant, Noir, Centre, AllT\rangle, \langle Montant, Bleu, Centre, AllT\rangle\} = \{t_8, t_9, t_{10}, t_{11}\}$
- $q_3^2 : \{\langle Montant, Rouge, Centre, AllT\rangle, \langle Montant, Bleu, Centre, AllT\rangle\} = \{t_8, t_{10}\}$

Nous pouvons assigner un vecteur pondéré de tuples $t_i, i \in [1, 14025]$ à chaque requête. Par souci de lisibilité, seuls les tuples $t_i, i \in [1, 15]$ sont affichés puisque ce sont les seuls tuples utilisés dans cet exemple. Nous obtenons donc les vecteurs basés sur la présence / absence de chaque tuple dans la requête[14]*. Pour les requêtes de la session s_1, nous obtenons :*

- $S_{q_1} = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
- $S_{q_2^2} = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]$
- $S_{q_3^2} = [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]$

Puis, nous calculons les vecteurs $S_j, j \in [0, 3]$ des sessions s_c, s_1, s_2, s_3 en sommant les vecteurs pondérés S_q des requêtes $q \in Q_j$ qui composent chaque session j tels que $S_j = \sum_{q \in Q_j} S_q$.
Nous obtenons pour la session s_1 :

- $S_1 = [1, 0, 0, 0, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 0]$

Nous pouvons ensuite prédire le vecteur pondéré de tuples de la session courante S_0^{pred} :

- $S_0^{pred} = [1, 0, 0, 0, 0, 0.67, 0.67, 1.16, 0.32, 1.16, 0.32, 0.67, 0.32, 0.16, 0.16]$

En considérant que la fonction d'utilité calcule la similarité au sens du Cosinus entre chaque vecteur pondéré S_q des tuples des requêtes et S_0^{pred}, une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_3^2	q_2^2	q_4	q_5	q_6
s_c	*0.706*	*0.637*	*0.288*	*0.159*	*0.137*

où, par exemple, la distance au sens du Cosinus entre la requête q_4 représentée par le vecteur S_{q_4} et la requête prédite représentée par le vecteur S_0^{pred} vaut 0.288*. L'utilité maximale est obtenue pour q_3^2.*
Ainsi, le système recommande q_3^2 à la suite de s_c.

Recommandations de jointures [YPS09]

Nous décrivons enfin la méthode de recommandation proposée par Yang et al.([YPS09]). Cette méthode calcule l'utilité d'une requête pour une requête courante. A partir d'un log de requêtes, du schéma de la base de données relationnelle et de la session courante, il s'agit de créer le graphe des requêtes du log où chaque nœud est une table de la base de données et où les arcs représentent le chemin emprunté par les requêtes. Un arc représente donc une jointure entre deux tables. En fait, chaque requête emprunte un chemin partant d'un ensemble de tables de départ qui sont spécifiées dans la clause $WHERE$ de la requête et allant jusqu'à un ensemble de tables d'arrivée qui sont spécifiées, quant à elles, dans la clause $SELECT$ de la requête. La requête contient aussi des jointures qui indiquent les tables intermédiaires qui permettent de passer des tables de départ aux tables d'arrivée.
Il est ensuite possible de trouver dans ce graphe de requêtes, tous les chemins empruntables pour passer d'un ensemble de tables de départ à un ensemble de tables d'arrivée (qui passeront par des tables / nœuds intermédiaires). Une requête recommandée est une requête du log empruntant un tel chemin. Elle spécifie les tables d'arrivée dans la clause $SELECT$, les tables intermédiaires dans la clause $FROM$ et les tables de départ ainsi que les conditions de jointure dans la clause

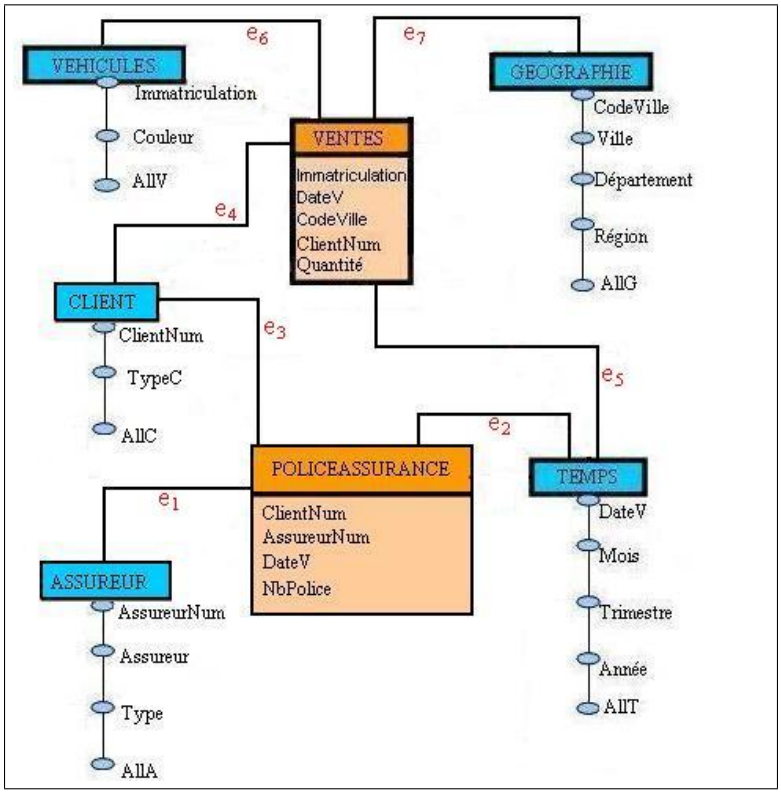
14. Chatzopoulou et al. proposent une autre méthode d'obtention du vecteur pondéré basée sur une division par la taille de réponse : nous nous limitons à la présence / absence.

$WHERE$. Dit différemment, à partir d'un ensemble de tables contenues dans les clauses $SELECT$ et $WHERE$, la méthode recommande les tables contenues dans la clause $FROM$, qui sont les conditions de jointures.

Cette méthode de recommandation collaborative utilise un modèle pour l'extrapolation des scores (le graphe de requêtes du log). Elle a pour objectif de sélectionner, parmi les requêtes du log, celles qui répondent au mieux aux spécifications de l'utilisateur.

Remarque : Cette méthode de recommandation considère le fait qu'une base de données est exploitée par plusieurs décideurs mais ne prend pas en compte le séquencement de requêtes.

Exemple 2.4.10 *Nous illustrons la méthode de Yang et al. en cherchant quelles requêtes pourraient être recommandées à partir d'une requête donnée (contenant un ensemble de tables de départ et un ensemble de tables d'arrivée).*
Nous considérons que cette méthode prend tout son sens dans le cas d'une constellation. En effet, si nous nous limitons à un schéma en étoile, il n'y a pas de jointure à recommander. Ainsi, nous utilisons la constellation suivante, qui est une extension du cube de données MesVentes dont le schéma en étoile est présenté Figure 2.3 de la Section 2.1.1. Le log est constitué de 20 requêtes $q_i, i \in [1, 20]$.



Les arcs $e_i, i \in [1, 7]$, *qui apparaissent sur le graphique correspondent aux arcs du graphe des requêtes du log. Ce graphe est constitué de trois chemins* π_1, π_2 *et* π_3 *tels que :*

Chemin	*Emprunté par*
$\pi_1 = e_1 - e_2 - e_5 - e_7$	$\{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}\}$
$\pi_2 = e_3 - e_2 - e_5$	$\{q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}\}$
$\pi_3 = e_4 - e_6$	$\{q_{10}, q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}, q_{17}, q_{18}, q_{19}, q_{20}\}$

où :

Chemin	*Tables de départ*	*Tables intermédiaires*	*Tables d'arrivée*
π_1	*{Assureur}*	*{PoliceAssurance, Temps, Ventes}*	*{Géographie}*
π_2	*{Client}*	*{PoliceAssurance, Temps}*	*{Ventes}*
π_3	*{Client}*	*{Ventes}*	*{Véhicules}*

Supposons maintenant que la requête courante soit :

```
SELECT  Crossjoin({[PoliceAssurance].[NbPolice]},
        {[Ventes].[Quantité]})                          ON ROWS,
        Crossjoin({[Véhicules].[AllV].[Couleur].Members},
        {[Géographie].[AllG].[Région].Members})         ON COLUMNS
FROM    [VentesEtPoliceAssurance]
WHERE   ([Client].[AllC].[Fonctionnaire],[Assureur].[AllA].[Vie].[GMF]) ;
```

où :

- *l'ensemble des tables de départ est : {Client, Assureur}*
- *l'ensemble des tables d'arrivée est : {PoliceAssurance, Ventes, Véhicules, Géographie}*

Nous considérons que la fonction d'utilité calcule le nombre de requêtes communes aux ensembles de requêtes empruntant chaque chemin, une partie de la matrice d'utilité, concernant la session courante, est :

$u(s,q)$	q_{π_1,π_3}	q_{π_1,π_2}	q_{π_1,π_2,π_3}
s_c=q_1	*1*	*6*	*1*

où, par exemple, l'utilité des requêtes q_{π_1,π_3} correspond au nombre de requêtes empruntant le chemin π_1 et π_3, à savoir 1 : la requête q_{10}. L'utilité maximale est obtenue pour q_{π_1,π_2} car ces deux chemins partagent un plus grand nombre de requêtes communes.
Intuition : Il est plus fréquent de passer par la table Temps que par la table Client, il est donc préférable de recommander Temps.
La requête recommandée est telle que :

- *l'ensemble des tables de départ est : {Client, Assureur}*
- *l'ensemble des tables intermédiaires est : {Temps}*
- *l'ensemble des tables d'arrivée est : {PoliceAssurance, Ventes, Véhicules, Géographie}*

Une requête recommandée pourrait être :

```
SELECT  Crossjoin(Crossjoin({[PoliceAssurance].[NbPolice]},
        {[Ventes].[Quantité]}),
        {[Temps].[AllT].[Années].Members})              ON ROWS,
        Crossjoin({[Véhicules].[AllV].[Couleur].Members},
        {[Géographie].[AllG].[Région].Members})         ON COLUMNS
FROM    [VentesEtPoliceAssurance]
WHERE   ([Client].[AllC].[Fonctionnaire],[Assureur].[AllA].[Vie].[GMF]) ;
```

2.4.4 Synthèse des approches existantes d'aide à l'exploration

Lors de notre étude bibliographique, nous avons pu examiner un certain nombre de travaux traitant de la recommandation ou étant assimilables à des méthodes de recommandation dans le domaine des bases de données (relationnelles ou multidimensionnelles). Même si l'objectif est d'aider à l'exploration de la base de données, les méthodes utilisées n'abordent pas le problème de la

même manière. Nous proposons donc une synthèse des travaux concernant la recommandation dans le domaine des bases de données (relationnelles et multidimensionnelles). Le tableau 2.7 donne une vision synthétique des méthodes de recommandations utilisées dans le domaine des bases de données.

Nous pouvons dégager les points suivants :

- Préoccupation : Tous les travaux présentés ont pour préoccupation de guider l'utilisateur lors de son exploration de la base de données, mise à part [Sap99, Sap00] qui prétraite les données afin de prévoir la prochaine requête à des fins d'optimisation.

- Méthode : La majorité des travaux utilise une méthode de recommandation basée sur le contenu, seuls [Sap99, Sap00], [CEP09] et [YPS09] proposent des approches collaboratives prenant ainsi en considération le fait qu'une base de données est exploitée par plusieurs décideurs pouvant avoir des intérêts communs.

- Filtre : L'estimation des scores se fait équitablement via un modèle obtenu sur les données ou via une heuristique, ne favorisant ainsi aucune des deux méthodes d'estimation.

- Guide : La majorité des travaux ordonne l'ensemble des recommandations obtenu pouvant ainsi choisir quelle recommandation pourra être proposée en premier à quel utilisateur.

- Application : Bien que la majorité des méthodes de recommandation proposées s'appliquent sur les requêtes, celles de [CCD+08] et [Sar99, Sar00, SS01] s'appliquent uniquement sur le résultat de la requête qui change pour chaque instance de la base de données.

- Séquencement : Seuls [CCD+08] et [Sap99, Sap00] prennent en compte le séquencement de requêtes d'une session dans leur approche.

- Requêtes recommandées : Bien qu'ils s'intéressent à des requêtes sur des bases de données multidimensionnelles, ni [JRTZ09], ni [Sap99, Sap00], ni [Sar99, Sar00, SS01], ni [CCD+08] ne traitent de requêtes MDX.

Par conséquent, nous remarquons qu'il existe peu de méthodes de recommandation dans le cadre des bases de données multidimensionnelles ([JRTZ09], [Sar99, Sar00, SS01] et [CEP09]). Parmi ces méthodes, seul [JRTZ09] recommande véritablement des requêtes mais ne fait pas d'exploration collaborative. Finalement, cette synthèse nous a permis de cerner les manques d'un système collaboratif de recommandation de requêtes MDX dans le domaine des bases de données multidimensionnelles, à savoir :

- Prendre en compte le séquencement de requêtes lors de l'exploration du cube,
- Prendre en compte ce qu'ont fait les autres utilisateurs du système,
- Recommander des requêtes MDX.

Référence				[CD91]	[JRTZ09]	[Sar99], [Sar00], [SS01]	[CCD+08]	[Sap99], [Sap00]	[CEP09]	[YPS09]
Application				BDR	BDM	BDM	BDM	BDM	BDR	BDR
Entrée (de l'algorithme de recommandation)		Log						×	×	×
		Schéma de BD		×	×	×	×	×	×	×
		Instance de BD			×	×	×	×	×	
		Session courante		×	×	×	×	×	×	×
		Profil		×	×					
Sortie (de l'algortihme de recommandation)				Requête	Requête	Ensemble de tuples	Ensemble de tuples	Requête	Ensemble de requêtes	Ensemble de requêtes
Algorithme	Approche	Contenu		×	×	×	×		×	
		Collaboratif						×	×	×
		Hybride							×	
	Filtre	Estimation des scores	Modèle			×		×		×
			Heuristique	×	×		×		×	
			Technique	Voisinage	Théorie des graphes	Voisinage	Test statistique	Modèle statistique	Voisinage	Théorie des graphes
		Génération		Calcul	Calcul	Sélection	Sélection	Sélection	Sélection	Sélection
	Guide						×	×	×	×

FIGURE 2.7 – Récapitulatif des travaux sur l'aide à l'exploration dans les bases de données

2.5 Conclusion et perspectives

Au cours de ce chapitre, nous avons présenté une vue d'ensemble des bases de données multidimensionnelles, de la modélisation à l'analyse multidimensionnelle en passant par les langages de manipulation de données, puis un aperçu des outils existants de traitement de logs de requêtes et une vue d'ensemble des systèmes de recommandations. Puis nous avons présenté un état de l'art sur les méthodes d'aide à l'exploration dans le domaine des bases de données (relationnelles ou multidimensionnelles). Enfin, au cours de notre étude bibliographique, nous avons vu que les systèmes de recommandation n'ont pas beaucoup été implantés dans le domaine des bases de données.

Nos travaux se situent dans le domaine des bases de données multidimensionnelles où nous proposons de recommander des requêtes à l'utilisateur afin de le guider dans son analyse de données. Pour aborder ce problème, nous nous sommes inspirés des systèmes de recommandation, utilisés dans le Web ou en Recherche d'Informations, pour l'exploration des données. Ainsi, plus particulièrement dans le cadre de recommandations pour bases de données multidimensionnelles, afin de guider l'utilisateur dans son exploration, nous proposons d'utiliser une méthode collaborative qui recommande une ou plusieurs requêtes, de préférence au format MDX et qui prend en compte le séquencement des requêtes lors de l'exploration.

En particulier, nous nous concentrons sur une approche basée sur le filtrage collaboratif : Si un utilisateur A a un comportement semblable à l'utilisateur B (en terme de requêtes lancées), alors ils sont probablement intéressés par les mêmes données. Par conséquent, les requêtes de l'utilisateur B peuvent servir de guide à l'utilisateur A.

Exemple 2.5.1 *Considérons les deux utilisateurs A et B interrogeant le même cube de données et supposons que le log de sessions de requêtes correspondant à l'exploration de ce cube contienne la séquence de requêtes lancées par B : q_1, puis q_2 puis q_3 telles que :*

- *q_1 : Quantité de véhicules vendus en France en 2007*
- *q_2 : Quantité de véhicules vendus en Loir-et-Cher en 2007*
- *q_3 : Quantité de véhicules vendus à Blois en 2007*

Supposons que l'utilisateur A, quant à lui, commence à interroger le cube via la séquence de requêtes q_4 puis q_5 telles que :

- *q_4 : Quantité de véhicules vendus en France toutes années confondues*
- *q_5 : Quantité de véhicules vendus en Loir-et-Cher*

Les utilisateurs A et B semblent intéressés par les mêmes données. Par conséquent, les requêtes de l'utilisateur B peuvent servir de guide à l'utilisateur A. Intuitivement, la requête q_3 pourrait être proposée à l'utilisateur A.

Notre approche est présentée dans le chapitre suivant, où nous essayons de comparer deux utilisateurs qui ont le même comportement.

Chapitre 3

Un cadre générique de génération des recommandations

Sommaire

Au cours du chapitre précédent, nous avons présenté les principaux travaux relatifs à la recommandation de requêtes dans la communauté des bases de données.

Dans ce chapitre, nous formulons notre problème :

Comment aider l'utilisateur à
avancer dans son analyse de données multidimensionnelles,
en lui proposant des requêtes pertinentes ?

Après la définition des distances utilisées pour comparer des sessions de requêtes MDX dans la Section 3.1, nous présentons dans la Section 3.2, le cadre générique de génération de recommandations afin d'apporter une solution à notre problème sus-cité. Enfin, nous proposons des instanciations du cadre générique dans la Section 3.3.

Notes aux lecteurs
Dans ce chapitre, le terme *session* sera utilisé pour décrire soit une session de requêtes comme définie Définition 2.1.7 soit tout autre type de session qui sera défini par la suite.

3.1 Distances

Dans le cadre de notre approche collaborative, nous allons utiliser des sessions de requêtes MDX déjà lancées sur le cube de données et enregistrées dans un log de sessions de requêtes que nous allons vouloir comparer à la session de requêtes MDX courante d'un utilisateur donné. Ainsi, nous avons besoin de distances entre sessions. Nous avons vu précédemment qu'une session de requêtes MDX est une séquence de requêtes MDX et qu'une requête MDX est un ensemble de références.

Dans cette section, nous proposons deux distances entre références, puis une distance entre requêtes et enfin une distance entre sessions.

3.1.1 Distance entre références

Dans cette sous-section, nous proposons deux distances entre références : la distance de Hamming et la distance basée sur le plus court chemin. Notre choix s'est d'abord porté vers une distance simple d'utilisation : la distance de Hamming puis, afin de pallier certains manques, nous avons opté pour une distance un peu plus complexe : la distance basée sur le plus court chemin.

3.1.1.1 Distance de Hamming entre références

La distance de Hamming [Ham50] permet de quantifier la différence entre deux séquences de symboles de même longueur. Or, nous avons défini une référence comme un n-uplet de membres (Définition 2.1.5). La distance de Hamming est une distance au sens mathématique du terme (ayant les propriétés de positivité, symétrie, séparation et d'inégalité triangulaire). À deux suites de symboles de même longueur, elle associe l'entier désignant le cardinal de l'ensemble des symboles de la première suite qui diffère de la deuxième. Elle a pour qualité sa simplicité d'utilisation et de compréhension.

Définition 3.1.1 *(Distance de Hamming entre références)*
Soit un cube $C = \langle D_1, ...D_N, F\rangle$*,* a *et* b *deux références de* $ref(C)$ *telles que* $a = \langle a_1, ..., a_N\rangle$ *et* $b = \langle b_1, ...b_N\rangle$*, la distance de Hamming* d_h *entre* a *et* b *est :*

$$d_h(a,b) = d_h(\langle a_1,...,a_N\rangle, \langle b_1,...,b_N\rangle) = \sum_{i=1}^{N} compare(a_i,b_i)$$

où $compare(a_i,b_i) = 0$ *si* $a_i = b_i$, 1 *sinon.*

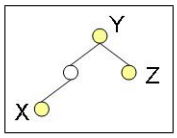
Exemple 3.1.1 *Considérons le cube* $MesVentes = \langle NomMesures, Véhicules, Géographie, Temps, Ventes\rangle$ *présenté Figure 2.3 et les deux références* $r_1 = \langle Montant, Bleu, Tours, AllT\rangle$ *issue de* q_3 *et* $r_2 = \langle Montant, AllV, Blois, 2008\rangle$ *issue de* q_6 *(*q_3 *et* q_6 *sont présentées Annexe A), la distance de Hamming entre* r_1 *et* r_2 *est :*

$$\begin{aligned} d_h(r_1,r_2) &= d_h(\langle Montant, Bleu, Tours, AllT\rangle, \langle Montant, AllV, Blois, 2008\rangle) \\ &= \sum_{i=1}^{4} compare(r_1^i, r_2^i) \\ &= compare(Montant, Montant) + compare(Bleu, AllV) \\ &+ compare(Tours, Blois) + compare(AllT, 2008) \\ &= 0 + 1 + 1 + 1 = 3 \end{aligned}$$

3.1.1.2 Plus court chemin

La distance de Hamming est simple d'utilisation et de compréhension, toutefois, elle ne permet pas de prendre en compte une caractéristique OLAP importante, à savoir les hiérarchies. En effet, les membres sont répartis dans chaque niveau de chaque hiérarchie de chaque dimension. Par conséquent, il s'agissait de trouver une méthode permettant de comparer des membres en prenant en compte leur différence de niveau dans une hiérarchie. Une solution simple est de baser la distance sur la longueur du plus court chemin entre deux membres de la hiérarchie.

Exemple 3.1.2 *Soit la hiérarchie suivante :*



Nous avons $d_h(\langle X\rangle, \langle Y\rangle) = 1$ *et* $d_h(\langle Y\rangle, \langle Z\rangle) = 1$, *or nous pouvons considérer que* Y *est plus proche de* Z *que de* X.

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 terme de nombre de nœuds. De nombreuses propositions de résolution de ce problème ont été présentées, citons notamment les algorithmes de Dijkstra [Dij71], de Bellman-Ford [Bel47], A^* [HNR68] et de Floyd-Warshall [Flo62].

En OLAP, une hiérarchie est souvent vue comme un graphe, voire un arbre de membres. Par conséquent, nous pouvons rechercher le plus court chemin entre deux membres d'une hiérarchie.

Définition 3.1.2 *(Distance entre membres basée sur le plus court chemin)*
Soit un cube C *et l'une de ses dimensions* D_i, *la distance* $d_{members}$ *entre deux membres* m_1, $m_2 \in adom(D_i)$ *d'une même hiérarchie est la longueur du plus court chemin entre* m_1 *et* m_2 *et se note :* $d_{members}(m_1,m_2)$. *Elle vérifie les propriétés de positivité, symétrie, séparation et inégalité triangulaire selon [BH90].*

Afin d'obtenir une distance entre références, nous proposons d'additionner les distances entre membres $d_{members}$, ce qui nous permet également de garder les propriétés de distance mathématique. Nous obtenons la définition suivante :

Définition 3.1.3 *(Distance entre références basée sur le plus court chemin)*
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 au sens du plus court chemin est : $d_{sp}(r_1, r_2) = \sum_{i=1}^{N} d_{members}(r_1^i, r_2^i)$

Preuve 3.1.1 *(Distance entre références basée sur le plus court chemin)*
Par définition, l'addition permet de conserver les propriétés de positivité, symétrie et séparation. L'inégalité triangulaire, quant à elle, se démontre par récurrence sur le nombre n de dimensions :

- *Démonstration pour $n = 1$:*
  $d_{sp}^1(r_1, r_2) = d_{members}(r_1^1, r_2^1)$
  Comme $d_{members}$ vérifie la propriété, d'après la Définition 3.1.2, la propriété est vraie au rang 1.
- *Supposons la propriété vraie au rang n :*
  $d_{sp}^n(r_1, r_2) \leq d_{sp}^n(r_1, y) + d_{sp}^n(y, r_2)$
- *Démonstration au rang $n+1$:*
  $d_{sp}^{n+1}(r_1, r_2) = d_{sp}^n(r_1, r_2) + d_{members}(r_1^{n+1}, r_2^{n+1})$
  $\Leftrightarrow d_{sp}^{n+1}(r_1, r_2) \leq d_{sp}^n(r_1, y) + d_{sp}^n(y, r_2) + d_{members}(r_1^{n+1}, y^{n+1}) + d_{members}(y^{n+1}, r_2^{n+1})$
  or : $d_{sp}^{n+1}(r_1, y) = d_{sp}^n(r_1, y) + d_{members}(r_1^{n+1}, y^{n+1})$
  et : $d_{sp}^{n+1}(y, r_2) = d_{sp}^n(y, r_2) + d_{members}(y^{n+1}, r_2^{n+1})$
  D'où : $d_{sp}^{n+1}(r_1, r_2) \leq d_{sp}^{n+1}(r_1, y) + d_{sp}^{n+1}(y, r_2)$.

Exemple 3.1.3 *Reprenons les données de l'exemple précédent, à savoir le cube $MesVentes = \langle NomMesures, Véhicules, Géographie, Temps, Ventes \rangle$ présenté Figure 2.3 et les deux références $r_1 = \langle Montant, Bleu, Tours, AllT \rangle$ et $r_2 = \langle Montant, AllV, Blois, 2008 \rangle$ issues des requêtes q_3 et q_6 de l'Annexe A, la distance entre r_1 et r_2 au sens du plus court chemin est :*

$$\begin{aligned} d_{sp}(r_1, r_2) &= d_{sp}(\langle Montant, Bleu, Tours, AllT \rangle, \langle Montant, AllV, Blois, 2008 \rangle) \\ &= \sum_{i=1}^{4} d_{members}(r_1^i, r_2^i) \\ &= d_{members}(Montant, Montant) + d_{members}(Bleu, AllV) \\ &+ d_{members}(Tours, Blois) + d_{members}(AllT, 2008) \\ &= 0 + 1 + 4 + 1 = 6 \end{aligned}$$

Notons que les membres $Tours$ et $Blois$ appartiennent au niveau $Ville$ de la hiérarchie de la dimension Géographie. Le plus court chemin entre ces deux nœuds consiste à remonter de deux arcs jusqu'au nœud Région puisque $Tours$ et $Blois$ appartiennent à la même région et de descendre de deux arcs, ce qui correspond à un chemin de taille 4.

Notation 3.1.1 *La distance de Hamming d_h ou la distance basée sur le plus court chemin d_{sp} sont deux alternatives pour calculer la distance entre deux références. Par la suite, l'une comme l'autre seront notées $d_{references}$.*

3.1.2 Distance entre requêtes

Nous avons besoin d'un mécanisme pour comparer deux requêtes. Il existe diverses méthodes dont l'inclusion de requêtes. Or, en ce qui nous concerne, cette approche n'est pas suffisante dans le sens où l'inclusion ne permet pas de calculer une distance au sens mathématique et qu'elle n'est pas symétrique : $Q \subset Q' \not\Leftrightarrow Q' \subset Q$. Ainsi, notre choix s'est tourné vers la classique distance de Hausdorff [Hau14] pour comparer deux ensembles qui est basée sur une distance entre les éléments de ces ensembles. Comme les requêtes MDX sont modélisées comme des ensembles de références, comparer

deux requêtes MDX revient à comparer deux ensembles de références. De manière informelle, la distance de Hausdorff considère que deux ensembles sont proches si chaque élément de l'un des ensembles est proche des éléments de l'autre ensemble.

Définition 3.1.4 *(Distance de Hausdorff entre requêtes)*
Soit deux requêtes q_1, q_2, la distance de Hausdorff entre q_1 et q_2 est :

$$d_H(q_1, q_2) = \max\{ \max_{r_1 \in q_1} \min_{r_2 \in q_2} d_{references}(r_1, r_2), \\ \max_{r_2 \in q_2} \min_{r_1 \in q_1} d_{references}(r_1, r_2) \}$$

Par ailleurs, il est aussi possible de comparer deux requêtes en se basant uniquement sur le nombre de dimensions qui diffèrent entre les deux. Ainsi, nous proposons une distance basée sur les dimensions définie comme suit.

Définition 3.1.5 *(Distance entre requêtes basée sur les dimensions)*
Soit deux requêtes q_1, q_2, cette distance d_{dim} donne le nombre de dimensions où q_1 et q_2 diffèrent (si $q_1 = R_1^1 \times \ldots \times R_n^1$ et $q_2 = R_1^2 \times \ldots \times R_n^2$, D_i est une dimension où q_1 et q_2 diffèrent si $R_i^1 \neq R_i^2$).

La distance de Hausdorff entre requêtes $d_H(q_1, q_2)$ est combinée avec la distance entre requêtes basée sur les dimensions $d_{dim}(q_1, q_2)$ afin de prendre plus ou moins en compte soit la distance entre références, soit les dimensions de la requête. Ainsi, la distance entre requêtes est définie comme une fonction de ces deux distances :

Définition 3.1.6 *(Distance entre requêtes basée sur la combinaison des deux distances)*
Soit deux requêtes q_1, q_2, la distance entre q_1 et q_2 est :

$$d^{\gamma}_{queries}(q_1, q_2) = \gamma \times d_{dim}(q_1, q_2) + (1 - \gamma) \times d_H(q_1, q_2) \text{ où } \gamma \in [0, 1].$$

Exemple 3.1.4 *Considérons les requêtes q_3 et q_6 décrites Annexe A telles que :*
$q_3 = \{r_3^1 = \langle Montant, Rouge, Tours, AllT\rangle, r_3^2 = \langle Montant, Bleu, Tours, AllT\rangle\}$ *et*
$q_6 = \{r_6^1 = \langle Montant, AllV, Blois, 2008\rangle\}$

Leur distance est calculée comme suit :

$d^{\gamma}_{queries}(q_3, q_6)$	$d_{references} = d_h$	$d_{references} = d_{sp}$
	$= d_H(q_3, q_6)$ $= \max\{\max\{\min\{d_{references}(r_3^1, r_6^1)\}, \min\{d_{references}(r_3^2, r_6^1)\}\},$ $\max\{\min\{d_{references}(r_6^1, r_3^1), d_{references}(r_6^1, r_3^2)\}\}\}$	
$d^0_{queries}(q_3, q_6)$	$= \max\{\max\{\min\{3\}, \min\{3\}\},$ $\max\{\min\{3, 3\}\}\}$ $= \max\{\max\{3, 3\}, \max\{3\}\}$ $= \max\{3, 3\}$ $= 3$	$= \max\{\max\{\min\{6\}, \min\{6\}\},$ $\max\{\min\{6, 6\}\}\}$ $= \max\{\max\{6, 6\}, \max\{6\}\}$ $= \max\{6, 6\}$ $= 6$
$d^1_{queries}(q_3, q_6)$	$= d_{dim}(q_3, q_6)$ $= 3$	
$d^{0.5}_{queries}(q_3, q_6)$	$= 0.5 \times d_{dim}(q_3, q_6) + 0.5 \times d_H(q_3, q_6)$ $= 0.5 \times 3 + 0.5 \times 3$ $= 3$	$= 0.5 \times 3 + 0.5 \times 6$ $= 4.5$

La propriété suivante indique l'intervalle possible des valeurs de la distance $d^{\gamma}_{queries}$. La valeur maximale de cette distance est notée $d^{max}_{queries}$.

Propriété 3.1.1 *Dans le cas où $d_{references} = d_{sp}$, soit un cube N-dimensionnel C, la distance $d^{\gamma}_{queries}$ varie de 0 à $d^{max}_{queries} = \gamma \times N + (1-\gamma) \times 2 \times \sum_{i=1}^{N} h_i$ où h_i est la hauteur de la hiérarchie de la dimension i.*

Preuve 3.1.2 *(Propriété 3.1.1)*
Nous avons : $d^{\gamma}_{queries}(q_1, q_2) \leq \gamma \times d_{dim}(q_1, q_2) + (1-\gamma) \times d_H(q_1, q_2)$.
Or, par définition : $d_{dim}(q_1, q_2) \leq N$
et, par définition : $d_H(q_1, q_2) \leq max\{d_{sp}(r_1, r_2) | r_1 \in q_1, r_2 \in q_2\}$
De plus, $d_{sp}(r_1, r_2) \leq \sum_{i=1}^{N} max\{d_{members}(r_1^i, r_2^i)\}$
et $d_{members}(r_1^i, r_2^i)$ est bornée par le plus long chemin dans le graphe associé à la dimension i. Si le graphe est un arbre alors le plus long chemin est $2.h_i$ où h_i est la hauteur de l'arbre.
Par conséquent, $d^{\gamma}_{queries}(q_1, q_2) \leq \gamma \times N + (1-\gamma) \times 2 \times \sum_{i=1}^{N} h_i$.

Exemple 3.1.5 *Considérons les requêtes q_1 et q_2 sur le cube $MesVentes$ telles que :*
$q_1 = \{\langle Montant, 'ZZZ', 41000, '01/01/2005'\rangle\}$ et $q_2 = \{\langle Quantité, 'AAA', 33000, '31/12/2007'\rangle\}$, la distance entre requêtes $d^{\gamma}_{queries}$ dans le cas où $d_{references} = d_{sp}$ et $\gamma = 0$ est $d^{0}_{queries}(q_1, q_2) = 22$. Or, pour le cube $MesVentes$, lorsque $d_{references} = d_{sp}$ et $\gamma = 0$, $d^{max}_{queries} = 0 \times 4 + (1-0) \times 2 \times \sum_{i=1}^{4} h_i = 0 + 2 \times (1+4+2+4) = 22$.
$d^{max}_{queries}$ est donc une valeur atteignable.

Propriété 3.1.2 *Dans le cas où $d_{references} = d_h$, soit un cube N-dimensionnel C, la distance $d^{\gamma}_{queries}$ varie de 0 à N.*

Preuve 3.1.3 *(Propriété 3.1.2)*
Nous avons : $d^{\gamma}_{queries}(q_1, q_2) \leq \gamma \times d_{dim}(q_1, q_2) + (1-\gamma) \times d_H(q_1, q_2)$.
Or, par définition : $d_{dim}(q_1, q_2) \leq N$
et, par définition : $d_H(q_1, q_2) \leq max\{d_h(r_1, r_2) | r_1 \in q_1, r_2 \in q_2\}$
De plus, $d_h(r_1, r_2) \leq N$
Par conséquent, $d^{\gamma}_{queries}(q_1, q_2) \leq N$.

Exemple 3.1.6 *Considérons de nouveau les requêtes q_1 et q_2 sur le cube $MesVentes$ de l'Exemple 3.1.5, la distance entre requêtes $d^{\gamma}_{queries}$ dans le cas où $d_{references} = d_h$ et $\gamma = 0$ est $d^{0}_{queries}(q_1, q_2) = 4$. Or, pour le cube $MesVentes$, lorsque $d_{references} = d_h$ et $\gamma = 0$, $d^{max}_{queries} = 4$.*
$d^{max}_{queries}$ est donc une valeur atteignable.

3.1.3 Distance entre sessions

Nous avons besoin d'un mécanisme pour comparer deux sessions. Diverses méthodes existent mais notre choix s'est porté sur des techniques d'*Approximate String Matching* qui comparent les séquences de caractères qui sont des mots ou des textes, or, nos séquences de requêtes forment des sessions.

3.1.3.1 Distance de Levenshtein

Issue du domaine de l'Approximate String Matching ([Nav01] pour un aperçu), la distance de Levenshtein calcule la distance entre deux chaînes de caractères x et y qui est le nombre minimal d'opérations nécessaires pour transformer x en y.
L'*Approximate String Matching* est donc le problème qui consiste à trouver dans un texte où un mot donné apparaît en permettant un nombre limité d'erreurs de concordance. Dans notre cas, les sessions sont vues comme des séquences d'éléments de la même manière qu'un texte est vu comme une séquence de caractères.

La distance de Levenshtein (ou distance d'Édition) [Lev66] permet les suppressions, les insertions et les substitutions de caractères afin de transformer une chaîne de caractères x en y. Elle peut être utilisée dans notre contexte, comme d'autres distances classiques. Nous présentons donc notre approche pour comparer deux sessions qui sont deux séquences de requêtes MDX.

Exemple 3.1.7 *Soit les sessions de requêtes $s_c = \langle q_1, q_2, q_3 \rangle$ et $s_2 = \langle q_1, q_3^2, q_3, q_4 \rangle$ présentées Annexe A. Si les opérations autorisées sur les sessions de requêtes sont l'insertion, la suppression et la substitution, alors, une séquence d'opérations qui transforme s_c en s_2 est : substituer q_2 par q_3^2 et insérer q_4 à la fin. Si toutes les opérations ont le même coût : 1 alors, cette séquence vaut 2. Une autre séquence permettant de transformer s_c en s_2 est : supprimer toutes les requêtes de s_c et insérer une à une les requêtes q_1, q_3^2, q_3 puis q_4. Bien entendu, le coût de cette dernière séquence n'est pas minimal.*

Comme dans la distance d'édition, afin de comparer deux sessions de requêtes s_1 et s_2, les opérations autorisées sont :

- La substitution d'une requête q_1 par une requête q_2. Nous proposons que le coût de cette opération soit la distance entre q_1 et q_2 comme définie Définition 3.1.6, à savoir $d^{\gamma}_{queries}(q_1, q_2)$.
- L'insertion (resp. suppression) d'une requête dans une séquence. Le coût de cette opération est une constante α [Nav01].

Une raison intuitive pour fixer le coût d'insertion (ou de suppression) à une constante est la suivante. Supposons que nous voulons calculer une distance entre la session $\langle a \rangle$ et la session $\langle a, b \rangle$ dans un premier temps et entre la session $\langle a \rangle$ et la session $\langle a, b' \rangle$ dans un second temps. Si $\langle a, b \rangle$ et $\langle a, b' \rangle$ ont été enregistrées dans le log, cela signifie que b est aussi intéressant que b'. Il n'y a donc pas de raison de distinguer ou favoriser l'ajout de b à l'ajout de b'. Dans les deux cas, les sessions sont distantes de $\langle a \rangle$ uniquement par le fait qu'une requête a été ajoutée.

Concernant la valeur de α, elle peut varier de 0 à $d^{max}_{queries}$. Des valeurs basses pour, par exemple, l'insertion, ne permettent pas de distinguer des sessions de requêtes trop longues. D'autre part, la valeur devrait être assez haute car il devrait être plus cher de supprimer puis d'insérer que de substituer. L'ajustement de cette valeur est présenté dans les expérimentations décrites Chapitre 4.

Définition 3.1.7 *(Distance entre sessions de requêtes)*
La distance entre deux sessions de requêtes s et s' est le coût minimal de toutes les séquences d'opérations transformant s en s'. Elle est notée $d_{sessions}$.

La propriété suivante indique que $d_{sessions}$ est une distance au sens mathématique.

Propriété 3.1.3 *D'après [Lev66], $d_{sessions}$ est une distance dans le sens où elle satisfait les propriétés suivantes : positivité, symétrie, séparation, inégalité triangulaire.*

La propriété suivante indique l'intervalle possible des valeurs de la distance $d_{sessions}$. La valeur maximale de cette distance est notée $d^{max}_{sessions}$.

Propriété 3.1.4 *Soit un cube N-dimensionnel C et un log dont la session la plus courte contient l_1 requêtes et la session la plus longue contient l_2 requêtes ($l_1 < l_2$), la distance $d_{sessions}$ varie de 0 à $d^{max}_{sessions} = (d^{max}_{queries} \times l_1) + \alpha(l_2 - l_1)$ où $d^{max}_{queries}$ est la valeur maximale de la distance $d_{queries}$ et α est le coût d'insertion (ou de suppression) d'une requête dans une session.*

Preuve 3.1.4 *(Propriété 3.1.4)*
$d_{sessions}(s_1, s_2)$ calcule le coût minimal pour transformer s_1 en s_2, c'est-à-dire pour transformer une séquence de longueur l_1 en une séquence de longueur l_2.

- *Supposons $l_1 < l_2$, comme indiqué précédemment, la substitution coûte moins cher que la combinaison de l'ajout et de la suppression. Ainsi, les l_1 requêtes de s_1 seront, au pire, substituées par les l_1 premières requêtes de s_2, cela aura pour coût : $d^{max}_{queries} \times l_1$, $d^{max}_{queries}$ correspondant à la valeur maximale de la distance $d_{queries}$ utilisée pour calculer le coût de substitution. Il reste ensuite à ajouter les $l_2 - l_1$ requêtes restantes dans s_2, cela aura pour coût : $\alpha(l_2 - l_1)$, α étant le coût d'ajout ou de suppression d'une requête.*
- *Supposons $l_2 < l_1$, les l_2 premières requêtes de s_1 seront, au pire, substituées par les l_2 requêtes de s_2, cela aura pour coût : $d^{max}_{queries} \times l_2$. Il reste ensuite à supprimer les $l_1 - l_2$ requêtes restantes dans s_1, cela aura pour coût : $\alpha(l_1 - l_2)$.*

Par conséquent, $d_{sessions}(s_1, s_2) \leq (d^{max}_{queries} \times l_1) + \alpha(l_2 - l_1)$ avec $l_1 < l_2$.

Exemple 3.1.8 *Considérons les sessions de requêtes $s_c = \langle q_1, q_2, q_3 \rangle$, $s_1 = \langle q_1, q_2^2, q_3^2 \rangle$, $s_2 = \langle q_1, q_3^2, q_3, q_4 \rangle$ et $s_3 = \langle q_1, q_2^2, q_5, q_6 \rangle$ présentées Annexe A. Supposons $\gamma = 0$ et α (le coût d'insertion ou de suppression) $= \frac{d^{max}_{queries}}{2} = 13$ et $d_{references} = d_{sp}$.*

- *La séquence d'opérations qui a le coût minimal pour transformer s_c en s_1 est : substituer q_2 par q_2^2 puis substituer q_3 par q_3^2. Substituer q_2 par q_2^2 coûte $d^0_{queries}(q_2, q_2^2) = 2$ et substituer q_3 par q_3^2 coûte $d^0_{queries}(q_3, q_3^2) = 2$. Ainsi, $d_{sessions}(s_c, s_1) = 4$.*
- *La séquence d'opérations qui a le coût minimal pour transformer s_c en s_2 est : substituer q_2 par q_3^2 puis insérer q_4 à la fin. Son coût est : $d_{sessions}(s_c, s_2) = 4$ (pour substituer q_2 par q_3^2) $+13$ (pour insérer q_4) $= 17$.*
- *La séquence d'opérations qui a le coût minimal pour transformer s_c en s_3 est : substituer q_2 par q_2^2 puis substituer q_3 par q_5 puis insérer q_6. Son coût est : $d_{sessions}(s_c, s_3) = 2$ (pour substituer q_2 par q_2^2) $+6$ (pour substituer q_3 par q_5) $+13$ (pour insérer q_6) $= 21$.*

Par conséquent, la session de requêtes s_1 est plus proche de la session de requêtes s_c que s_2 ou s_3.

3.2 Le cadre générique de génération des recommandations

Notre problématique est : Comment aider l'utilisateur à avancer dans son exploration des données ?

Comme réponse, nous proposons d'exploiter ce que les autres utilisateurs ont fait durant leurs précédentes navigations dans le cube, et d'utiliser ces informations comme base pour recommander ce que pourrait être la prochaine requête. A cet effet, nous présentons un cadre générique pour recommander des requêtes MDX qui utilise à la fois le log du serveur, c'est-à-dire l'ensemble des sessions de requêtes déjà posées pour naviguer dans le cube, et la séquence de requêtes de la session courante.
La raison sous-jacente à cette proposition de cadre générique est que les recommandations dépendent beaucoup de l'utilisateur et des données sur lesquelles les recommandations sont calculées. Par exemple, [AT05] indique le besoin de systèmes de recommandations flexibles et adaptés à l'utilisateur.

Le cadre que nous proposons repose sur le processus suivant :

1. Prétraitement des logs
2. Filtre : Génération des recommandations candidates en cherchant d'abord les sessions qui coïncident le mieux avec la session courante puis en prédisant ce que pourrait être la prochaine requête
3. Guide : Ordonnancement des recommandations candidates afin de présenter les requêtes à l'utilisateur par ordre d'intérêt

Ce cadre est générique dans le sens où il n'impose pas une méthode spécifique de prétraitement des logs, de génération des recommandations candidates ou d'ordonnancement des candidats. Au lieu de cela, ces actions sont laissées comme paramètres du cadre qui, de fait, peut être instancié de différentes manières afin de changer la méthode de calcul des recommandations.

Dans cette section, nous détaillons le cadre de recommandations de requêtes MDX.

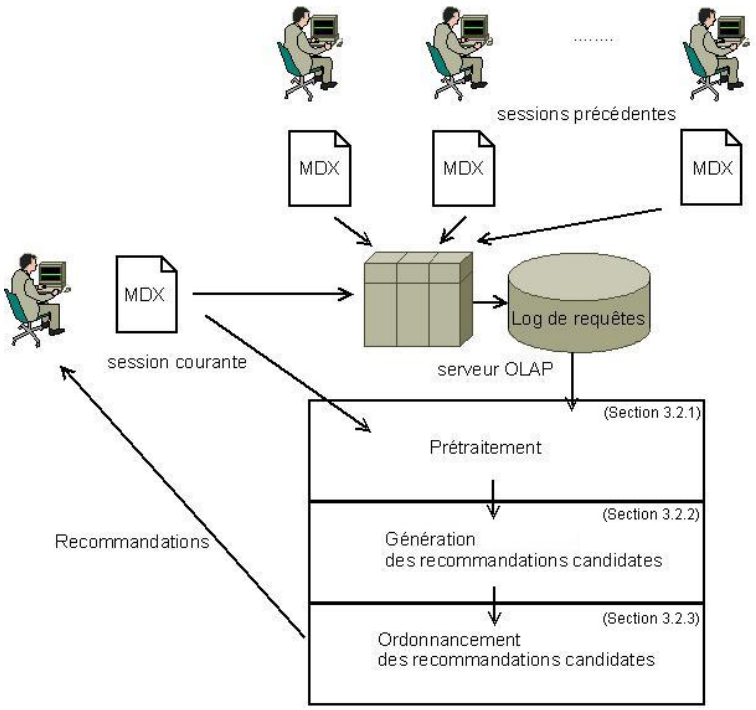


FIGURE 3.1 – Vue d'ensemble du cadre général

L'algorithme 1 global simplifié représente ce processus.

Chacune des étapes de ce processus est paramétrée avec une ou plusieurs fonctions. En changeant ces paramètres, la manière dont les recommandations sont calculées change (ce qui est illustré Section 3.3). Nous présentons donc chaque étape plus en détail dans cette section.

Remarque :

Comme définies Définition 2.4.1, nos recommandations sont obtenues à partir d'une matrice $Sessions \times Requêtes$, en mesurant l'utilité d'une requête pour une session donnée, et en retournant la (les) requête(s) qui maximise(nt) cette utilité. La matrice $Sessions \times Requêtes$ contient une valeur, qui dépend de l'instanciation du cadre, pour les requêtes déjà présentes dans la session. En

Algorithme 1 *Reco***OLAP**(S_c, $\mathscr{L}$, *Prétraitement*, g, *Match*, *Rep*, t, *Ordonnancement*, $\prec$, *Défaut*)

Entrées :
S_c : La session courante,
$\mathscr{L}$: Le log de sessions de requêtes,
Prétraitement : Une fonction de prétraitement de $\mathscr{L}$,
g : Une fonction de prétraitement de S_c,
Match : Une fonction de recherche de coïncidence entre deux sessions,
Rep : Un représentant de session,
t : Un seuil (entier),
Ordonnancement : Une fonction d'ordonnancement de requêtes,
$\prec$: Un ordre sur les requêtes,
Défaut : Une fonction qui renvoie une recommandation par défaut.
Sorties : Un ensemble ordonné de recommandations

1: $\mathcal{L} \leftarrow Prétraitement(\mathscr{L})$
2: $SetCandReq \leftarrow GénérationRequêtesCandidates(S_c, \mathcal{L}, g, Match, Rep, t)$
3: **Si** $SetCandReq \neq \emptyset$ **Alors**
4: **retourne** $Ordonnancement(SetCandReq, \prec)$
5: **Sinon**
6: **retourne** $Defaut(\mathcal{L})$
7: **Fin Si**

effet, si la requête appartient à la session, alors la requête est intéressante. De plus, si la requête n'a pas été lancée, il n'est pas possible de lui donner une 'note'. Enfin, arbitrairement, nous prenons pour cette valeur, une valeur maximale fonction de l'instanciation.

Exemple 3.2.1 *La matrice Sessions × Requêtes correspondant aux sessions s_1, s_2, s_3 et s_c du log de sessions de requêtes présenté Annexe A est :*

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	max		max		max			
s_2	max			max	max	max		
s_3	max		max				max	max
s_c	max	max		max				

3.2.1 Prétraitement des logs

Au vu du nombre d'utilisateurs et du nombre de sessions de requêtes possibles, le log peut être très volumineux. En outre, les divers utilisateurs peuvent avoir différents intérêts, et, vu la taille du cube, leurs requêtes peuvent naviguer dans des parties très différentes du cube. Le log de sessions de requêtes a donc besoin d'être prétraité pour pallier les problèmes engendrés par les différences d'intérêts des utilisateurs (Figure 3.1 : Prétraitement).

A partir du log de sessions de requêtes, la fonction de prétraitement *Prétraitement* : $\mathscr{L} \rightarrow$ *Prétraitement*($\mathscr{L}$) permet d'obtenir un log prétraité contenant toujours un ensemble de sessions. Notons que la fonction de prétraitement peut être l'identité, auquel cas, le log prétraité est le log initial de requêtes.

Un exemple de prétraitement est présenté Section 3.3.1.

3.2.2 Génération des recommandations candidates

A partir du log prétraité et de la session de requêtes courante, un ensemble de recommandations candidates est calculé grâce à l'Algorithme 2. Le principe de cet algorithme est le suivant. En plus des sessions du log prétraité et de la session de requêtes courante, l'algorithme utilise deux fonctions, $Match$ et Rep. $Match$ est utilisée pour trouver un ensemble de sessions qui coïncide avec une session donnée. Rep est utilisée pour obtenir le représentant d'une session donnée, c'est-à-dire une requête (Figure 3.1 : Prédiction / Génération). L'idée sous-jacente est de trouver parmi les sessions du log, celles qui ressemblent le plus possible à la session courante de l'utilisateur et de recommander l'une des requêtes de ces sessions à l'utilisateur.

L'algorithme utilise ces fonctions de la manière suivante. Premièrement, la session de requêtes courante est prétraitée avec la fonction de prétraitement $g : S_c, \mathcal{L} \rightarrow g(S_c, \mathcal{L})$ pour que les sessions du log prétraité $\mathcal{L}$ et la session courante S_c aient le même format (Ligne 2). Puis, $Match$ est utilisée pour chercher parmi l'ensemble des sessions celles qui coïncident avec la session courante (Ligne 3). Cette fonction renvoie un ensemble de sessions candidates ($CandSessions$) qui coïncident avec la session courante. Enfin, pour chaque session candidate, Rep est utilisée pour obtenir le représentant de la session candidate (Ligne 6-8). Cet ensemble de requêtes ($CandReq$) est retourné comme réponse (Ligne 10). Notons que dans certains cas, cet ensemble peut être vide.

Des exemples de fonctions $Match$ et Rep sont présentées Section 3.3.2.

Algorithme 2 GénérationRequêtesCandidates(S_c, $\mathcal{L}$, g, $Match$, Rep, t)

Entrées :
- S_c : La session de requêtes courante
- $\mathcal{L}$: Le log de sessions de requêtes prétraité obtenu à l'étape précédente
- g : Une fonction de prétraitement de la session courante
- $Match$: Une fonction générant des sessions candidates
- Rep : Une fonction renvoyant un représentant de session
- t : Un seuil (entier)

Sorties : Un ensemble de requêtes candidates

```
1:  𝒮 ← L'ensemble des sessions contenu dans ℒ
2:  PS ← g(S_c, ℒ)
3:  CandSessions ← Match(PS,𝒮,t)
4:  CandReq ← ∅
5:  Si CandSessions ≠ ∅ Alors
6:     Pour chaque s ∈ CandSessions Faire
7:        CandReq ← CandReq ∪ Rep(s)
8:     Fin Pour
9:  Fin Si
10: retourne CandReq
```

3.2.3 Ordonnancement des recommandations candidates

Dans l'étape précédente, un ensemble de requêtes candidates à la recommandation est calculé. L'objectif de cette étape suivante est d'ordonner les requêtes, étant donné un critère de satisfaction, ou relation d'ordre, exprimé par l'utilisateur (Figure 3.1 : Ordonnancement). A cet effet, un ordonnancement de requêtes est nécessaire, qui ordonne les recommandations candidates. Ainsi, à partir de l'ensemble de requêtes candidates $CandReq$ et d'une relation d'ordre $\prec$, la fonction d'ordonnancement $Ordonnancement : CandReq, \prec \rightarrow Ordonnancement(CandReq, \prec)$ permet d'obtenir l'ensemble ordonné des recommandations.

Des exemples de relations d'ordre sont présentées Section 3.3.3.

3.2.4 Recommandations par défaut

Comme remarqué précédemment, l'ensemble de recommandations candidates peut être vide. Dans ce cas, il pourra être utile de proposer à l'utilisateur une recommandation par défaut. Ainsi, à partir du log de sessions de requêtes prétraité, la fonction $Défaut : \mathcal{L} \rightarrow Défaut(\mathcal{L})$ permet d'obtenir une telle recommandation. Diverses recommandations par défaut peuvent être proposées à l'utilisateur.

Des exemples sont présentés Section 3.3.4.

3.3 Instanciations

Dans cette section, nous proposons différentes instanciations possibles de chacune des trois étapes de notre cadre générique de génération de recommandations afin d'illustrer l'applicabilité de nos algorithmes génériques. Les sessions de requêtes, sur le cube $MesVentes$ présenté Figure 2.3, utilisées dans les exemples de cette section sont annexées Annexe A.

3.3.1 Prétraitement

Nous présentons un paramètre possible de l'étape de prétraitement des logs : le partitionnement.

Définition 3.3.1 *(Classe de requêtes)*
Soit un cube C, une classe de requêtes est un ensemble $Q \subseteq query(C)$.

Définition 3.3.2 *(Partitionnement d'ensembles de requêtes)*
Soit un cube C et une distance entre requêtes, un partitionnement d'ensembles de requêtes est une fonction p de $2^{query(C)}$ vers $2^{2^{query(C)}}$ telle que, pour tout $Q \subset query(C)$, p calcule une partition de Q sous la forme d'un ensemble P de classes de requêtes.

En effet, au vu des différences d'intérêts des utilisateurs, le log peut être épars. Par conséquent, nous proposons le partitionnement du log de sessions de requêtes afin de grouper des requêtes similaires et ainsi pallier le problème de faible densité des logs. Pour cela, cette étape nécessite un partitionnement d'ensembles de requêtes. Ce partitionnement utilise une distance entre requêtes pour déterminer les partitions. Il renvoie un ensemble d'ensembles de requêtes que nous appelons un ensemble de classes de requêtes par la suite où chaque requête appartient à une seule classe.

Lorsque le log est partitionné en un ensemble de classes de requêtes, nous pouvons calculer l'ensemble des sessions généralisées. Le principe est le suivant : Pour chaque session de requêtes du log, remplacer, dans la session, chaque requête par la classe à laquelle elle appartient. Partitionner l'ensemble des requêtes peut être fait en utilisant un algorithme de clustering simple comme les K-médoïdes proposé par [KR87].

Ainsi, nous disposons de classes de requêtes et nous avons besoin d'une fonction permettant de classer une requête dans une des classes.

Définition 3.3.3 *(Classifieur de requêtes)*
Soit un cube C, un classifieur de requête cl est une fonction de $query(C) \times 2^{2^{query(C)}}$ vers $2^{query(C)}$ telle que, si $q \in query(C)$ est une requête et $P \subseteq 2^{query(C)}$ un ensemble de classes, alors $cl(q, P) \in P$. Nous considérons que $cl(q, P)$ est la classe de q.

Dans ce cas, le classifieur 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.

Définition 3.3.4 *(Session généralisée)*
Soit un cube C, un ensemble de classes de requêtes P, un classifieur de requête cl et $s = \langle q_1, \ldots, q_p \rangle$ une session de requêtes 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 $i \in [1, p]$.
Nous notons $gs[i]$ la $i^{ème}$ classe de la session généralisée gs.

Exemple 3.3.1 *Soit le log L présenté Annexe A contenant les trois sessions de requêtes s_1, s_2, s_3, et soit l'appel à la fonction de prétraitement $Prétraitement$ = k-medoides : $Prétraitement(L)$, nous obtenons la partition P constituée des cinq classes de requêtes suivantes :*

- $c_1 = \{q_1\}$,
- $c_2 = \{q_2^2\}$,
- $c_3 = \{q_3, q_3^2\}$,
- $c_4 = \{q_4\}$,
- $c_5 = \{q_5, q_6\}$

Nous remplaçons ensuite une requête dans une session par la classe à laquelle elle appartient. Nous obtenons ainsi les sessions généralisées g_1 de s_1, g_2 de s_2 et g_3 de s_3 :

- $g_1 = \langle c_1, c_2, c_3 \rangle$,
- $g_2 = \langle c_1, c_3, c_3, c_4 \rangle$,
- $g_3 = \langle c_1, c_2, c_5, c_5 \rangle$

Ces trois sessions généralisées constituent le log prétraité $\mathcal{L}$.

3.3.2 Génération

A partir des sessions du log prétraité et de la session de requêtes courante, nous voulons obtenir un ensemble de recommandations candidates calculé grâce à l'Algorithme 2 donné Section 3.2.2 qui utilise les fonctions, *Match* et *Rep*.

3.3.2.1 Match

La fonction *Match* utilisée dans l'algorithme de génération des requêtes candidates (Algorithme 2) permet de trouver un ensemble de sessions qui coïncident avec une session donnée (Figure 3.1 : Génération). Les exemples de fonction *Match* que nous considérons ici sont inspirés du domaine de l'Approximate String Matching [Nav01].

$Match_1$
Le problème consistant à trouver dans un ensemble de sessions, où apparaît la session courante, ou une légère modification de celle-ci, peut être traduit par un problème d'Approximate String Matching, pour lequel des algorithmes classiques peuvent être utilisés. Plus précisément, ces algorithmes prennent en entrée un texte t, une chaîne de caractères Ch, un ensemble d'opérations Op avec le coût de chaque opération *cost* et un seuil th et retournent un ensemble de paires $\langle p, c \rangle$ où p est la position et c est un coût. Le résultat indique les positions dans le texte où la chaîne de caractères coïncide, à th erreurs maximum près, avec le texte et, pour chaque position, le coût c de transformations au sens des opérations de Op nécessaires pour transformer la chaîne de caractères et obtenir la coïncidence.

Par exemple, si $approximateStringMatching(t, Ch, \{Op\}, \{cost\}, th)$ est une telle fonction, alors $approximateStringMatching(\text{"}hello\ word\text{"}, \text{"}world\text{"}, \{removeOneLetterToCh\}, \{1\}, 1)$ retournera $\langle 7, 1\rangle$ car supprimer une lettre de "*world*" consiste en une opération de coût 1 et entraînera la détection d'une coïncidence à la position 7 dans "*hello word*". Dans notre contexte, le texte est une des sessions du log $\mathscr{L}$, la chaîne de caractères est la session courante s_c, l'ensemble d'opérations est l'ensemble des opérations autorisées pour transformer une session en une autre.

Exemple 3.3.2 *(Recherche de sous-séquences)*
Dans le cas de la recherche de sous-séquences, la méthode d'Approximate String Matching utilisée se limite à supprimer le premier élément de la séquence recherchée si une correspondance exacte n'existe pas, et ainsi de suite jusqu'à ce que la séquence recherchée ne contienne plus d'élément. De sorte que l'ensemble des opérations peut être réduit à une seule opération : la suppression du premier élément de la chaîne de caractères recherchée : $\{Op\} = \{removeFirstElementToCh\}$*. L'objectif étant de voir si en enlevant des caractères à la chaîne recherchée, celle-ci est une sous-séquence du texte dans lequel elle est recherchée.*

Afin d'illustrer cette démarche, considérons les sessions de requêtes $s_c = \langle q_1, q_2, q_3\rangle$ *et* $s_2 = \langle q_1, q_3^2, q_3, q_4\rangle$ *présentées Annexe A.*
L'appel de la fonction $approximateStringMatching(s_2, s_c, \{removeFirstElementToSc\}, \{1\}, 3)$ *renvoie la paire* $\langle 3, 2\rangle$*. Ce qui signifie que la session de requêtes* s_2 *coïncide avec la session de requêtes* s_c *à la position* 3 *de* s_2 *et que le coût de la transformation vaut* 2*, à savoir supprimer* q_1 *puis* q_2 *de* s_c*. Ainsi,* s_c *privée de ses deux premiers éléments est une sous-séquence de* s_2*.*

La Fonction 3, basée sur l'Approximate String Matching, itère jusqu'à trouver au moins une concordance ou que le coût de transformation de la session courante n'excède pas le seuil (Ligne 3). A chaque itération, l'ensemble de sessions est scanné (Ligne 4) et la fonction *approximateString-Matching* est utilisée pour déterminer si appliquer des opérations de *Op* pour un coût *cost* sur la session courante permet d'obtenir des concordances (Ligne 5). Si cela est possible, alors le résultat de l'algorithme sera l'ensemble de paires $\langle CandSession, i\rangle$ où $CandSession$ est la session où la concordance est trouvée et i est la position de la concordance (Ligne 8). Sinon, *cost* est incrémenté (Ligne 10).

$Match_2$
La Fonction 4, quant à elle, calcule la distance $d_{sessions}$ entre la session courante et chaque session du log prétraité jusqu'à un seuil t en ne gardant que les sessions candidates qui obtiennent une distance minimale (Lignes 3 - 12). Puis la fonction retourne l'ensemble des sessions candidates (qui peut être vide) (Ligne 13).

3.3.2.2 Rep

La fonction $Match$ renvoie un ensemble de sessions candidates à la recommandation mais nous voulons recommander des requêtes à l'utilisateur pour l'aider à avancer dans son analyse. Nous avons donc besoin d'un mécanisme pour passer d'une session candidate à une requête candidate. Ainsi, la fonction *Rep*, utilisée dans l'algorithme de génération des requêtes candidates (Algorithme 2) permet d'obtenir le représentant d'une session donnée (Figure 3.1 : Génération) qui est la requête qui représente le mieux la session donnée au sens d'un des critères que nous présentons ci-après.

Fonction 3 $Match_1$ (PS, $\mathscr{S}$, t)

Entrées :
PS : La session courante prétraitée,
$\mathscr{S}$: L'ensemble des sessions du log prétraité,
t : Un seuil (entier).
Utilise :
Op : Un ensemble d'opérations autorisées sur une session pour la fonction *approximateStringMatching*.
Sorties : Un ensemble de paires $\langle CandSession, i\rangle$ où $CandSession$ est la session candidate et i est un entier

```
1: cost ← 0
2: Cand ← ∅
3: Tant que Cand = ∅ et cost ≤ t Faire
4:    Pour chaque CandSession ∈ 𝒮 Faire
5:       S=approximateStringMatching(CandSession, PS, Op, cost)
6:    Fin Pour
7:    Si S ≠ ∅ Alors
8:       Cand ← {⟨CandSession, i⟩|⟨i, cost⟩ ∈ S}
9:    Sinon
10:      cost ← cost + 1
11:   Fin Si
12: Fin Tant que
13: retourne Cand
```

Remarque

- Dans le cas des sessions généralisées, il s'agit de sessions de classes de requêtes. *Rep* est donc vu comme une fonction associant une requête à un ensemble de requêtes quelle que soit sa structuration.
- *Rep* peut prendre en paramètres soit une session, soit une session et un indice.

3.3.2.2.1 Successeur

La fonction $Match_1$ renvoie un ensemble de paires $\langle s, i\rangle$ où s est la session candidate et i est la position dans la session où la coïncidence a été trouvée. Une fonction *Rep* simple consiste à renvoyer, pour chaque paire, le successeur de $s[i]$, à savoir $s[i+1]$.

Définition 3.3.5 *(Successeur)*
Soit une session s et un indice i, le successeur du $i^{ème}$ élément de s est défini par : $Rep^{Succ}(s, i) = s[i+1]$.

3.3.2.2.2 Dernier

Par analogie avec la recherche sur le Web, où il a été montré que ce qui est vu à la fin d'une session peut être utilisé pour améliorer les recherches à venir [WBC07], un autre représentant de session peut être le dernier élément de la session. En effet, également dans notre cas, considérer que, si la session se termine, cela indique que l'utilisateur a trouvé quelque chose d'intéressant, a un sens. Nous adoptons ce point de vue et définissons le représentant d'une session : "dernier" comme étant le dernier élément d'une session candidate.

Définition 3.3.6 *(Dernier)*
Soit une session s, le dernier élément de s est défini par : $Rep^{Last}(s) = s[|s|]$ *où $|s|$ est la longueur de la session s.*

Fonction 4 $Match_2$ (PS, $\mathscr{S}$, t)

Entrées :
PS : La session courante prétraitée,
$\mathscr{S}$: L'ensemble des sessions du log prétraité,
t : Un seuil (entier).
Sorties : L'ensemble des sessions candidates

```
1: SetCandS ← ∅
2: MIN ← ∞
3: Pour chaque CandS ∈ 𝒮 Faire
4:    d_CandS=d_sessions(CandS, PS)
5:    Si (d_CandS ≤ t)&&(d_CandS = MIN) Alors
6:       SetCandS ← SetCandS ∪ CandS
7:    Fin Si
8:    Si (d_CandS ≤ t)&&(d_CandS < MIN) Alors
9:       MIN ← d_CandS
10:      SetCandS ← CandS
11:   Fin Si
12: Fin Pour
13: retourne SetCandS
```

3.3.2.2.3 Union ou Intersection

L'union de deux requêtes q_1 et q_2 : $q_1 \cup q_2$ consiste à faire l'union, par dimension i, des références de chaque requête $R_i^{q_1}$ et $R_i^{q_2}$ afin d'obtenir l'ensemble des références de chaque dimension $R_i = R_i^{q_1} \cup R_i^{q_2}$, puis de faire le produit cartésien de l'ensemble des références de chaque dimension : $\times_{i=1}^{N} R_i$. Plus formellement, soit C un cube N-dimensionnel, $q_1, q_2 \in query(C)$, $q_1 \cup q_2 = \times_{i=1}^{N} R_i^{q_1} \cup R_i^{q_2}$.

Définition 3.3.7 *(Union)*

Soit une session s, l'union des éléments de s est défini par : $Rep^{Union}(s) = \bigcup query(s)$ où $query(s)$ est l'ensemble des requêtes de la session s.

3.3.2.2.4 Médoïde

Le représentant peut aussi être la requête la plus représentative de la session au sens du médoïde de la session. Le médoïde d'un ensemble de requêtes appartient obligatoirement à cet ensemble de requêtes et correspond à la requête qui minimise les distances entre elle et les autres requêtes de l'ensemble.

Définition 3.3.8 *(Médoïde)*
Soit une session s, le médoïde des éléments de s est défini par : $Rep^{Médoïde}(s) = medoide(query(s))$ où $medoide(query(s))$ renvoie le médoïde des requêtes de la session s.

3.3.3 Ordonnancement

L'étape précédente nous fournit donc un ensemble de requêtes candidates à la recommandation. Encore une fois, il existe différentes méthodes d'ordonnancement des recommandations candidates, de la plus simple à la plus sophistiquée.

La fonction d'ordonnancement, $Ordonnancement : CandReq, \leq_{queries} \rightarrow Ordonnancement(CandReq, \leq_{queries})$, consiste donc à trier les requêtes candidates $CandReq$

selon un ordre entre requêtes $\leq_{queries}$ donné.

Citons par exemple :
- Ordonner les recommandations candidates en fonction de leur proximité avec la requête qui représente la session courante (Proximité),
- Ordonner les recommandations candidates en fonction de leur nombre d'occurrences dans l'ensemble des candidats,
- Ordonner les recommandations candidates en fonction du nombre de références qui n'ont pas encore été vues par l'utilisateur dans la session courante,
- Ordonner les recommandations candidates en fonction du profil de l'utilisateur (Personnalisation).

Deux d'entre elles sont présentées dans les sous-sections suivantes : Proximité et Personnalisation.

3.3.3.1 Proximité

Cette méthode permet d'ordonner les recommandations candidates en fonction de leur proximité avec le représentant de la session courante, c'est-à-dire la requête qui représente la session courante (successeur, dernier, médoïde du successeur, ...). Cette proximité est entendue au sens de la distance entre requêtes $d^{\gamma}_{queries}$ définie Definition 3.1.6.

Ainsi, soient deux requêtes candidates à la recommandation q_1, q_2 et soit $q_c = Rep(s_c)$ le représentant de la session courante, si $d^{\gamma}_{queries}(q_c, q1) < d^{\gamma}_{queries}(q_c, q2)$ alors l'ensemble ordonné des recommandations selon l'ordre $\leq_{s_c}$ sera $\{q_1, q_2\}$, sinon ce sera $\{q_2, q_1\}$.

3.3.3.2 Personnalisation

L'ensemble de requêtes candidates à la recommandation obtenues lors de l'étape de génération 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 utilisateur). Cette étape de personnalisation va permettre de proposer un ordre de requêtes candidates différent selon l'utilisateur.

Voici la définition du profil utilisateur selon [BGMM06] :

Définition 3.3.9 *(Profil utilisateur)*
Soit un cube $C = \langle D_1, ..., D_N, F\rangle$*, un profil utilisateur* Γ *sur* C *est un n-uplet* $\langle <_d, \{<_1, ..., <_N\}\rangle$ *où :*
- $<_d$ *est un ordre total sur les dimensions de* C*. Soient deux dimensions* D_i *et* D_j*,* D_i *est moins importante que* D_j*, si* $D_i <_d D_j$*.*
- *Pour chaque* $i \in [1, N]$*,* $<_i$ *est un ordre total sur les membres de la dimension* D_i*. Soient deux membres* m *et* m'*,* m *est moins intéressant que* m'*, si* $m <_i m'$*.*

Toujours d'après [BGMM06], le rôle du profil utilisateur est donc de fournir un ordre sur les références :

Définition 3.3.10 *(Ordre sur les références)*
Soit un profil utilisateur Γ *sur un cube* C*, soient* r *et* r' *deux références sur* C*, nous notons* $\Delta(r, r')$ *l'ensemble des dimensions où* r *et* r' *diffèrent, c'est-à-dire* $\Delta(r, r') = \{D_i \in C | r(D_i) \neq r'(D_i)\}$*.* r

est moins intéressante que r', noté $r \leq_\Gamma r'$, si $\forall D_i \in max_{<_d}(\Delta(r, r')), r(D_i) <_i r'(D_i)$.
Notons que l'ordre $\leq_\Gamma$ défini sur les références est un ordre lexicographique total.

Enfin, dans nos travaux, les requêtes MDX sont comparées selon leur ensemble de références. Ainsi, toujours inspirés par les travaux de [BGMM06], nous proposons un algorithme qui permet de comparer des requêtes par rapport au profil utilisateur.

Définition 3.3.11 *(Ordre sur les requêtes)*
Soit $\leq_\Gamma$ l'ordre total sur les références, comme défini Définition 3.3.10, et 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 $ref(q')$) représente l'ensemble des références de la requête q (respectivement q').

Une solution possible pour réaliser un tel ordonnancement est décrite par l'Algorithme 5 où, pour chaque couple de requêtes q, q' de l'ensemble $CandReq$ des requêtes candidates à la recommandation, si la référence préférée de q est rangée avant la référence préférée de q' alors q est placée avant q' dans l'ensemble ordonné $Reco$ sinon q' est placée avant q. Finalement, l'ensemble $Reco$ des requêtes candidates ordonnées en fonction de l'ordre sur les références $\leq_\Gamma$, c'est-à-dire l'ensemble des recommandations, est recommandé à l'utilisateur.

Algorithme 5 Perso($CandReq$,$\leq_\Gamma$)

Entrées :
$CandReq$: L'ensemble de requêtes candidates,
$\leq_\Gamma$: Un ordre sur les références.
Sorties : Un ensemble ordonné de recommandations

1: $Reco \leftarrow \emptyset$
2: **Pour** chaque $q, q' \in CandReq$ **Faire**
3: **Si** $max_{\leq_\Gamma}(ref(q)) \leq_\Gamma max_{\leq_\Gamma}(ref(q'))$ **Alors**
4: $Reco \leftarrow Reco \cup \{(q, q')\}$
5: **Sinon**
$Reco \leftarrow Reco \cup \{(q', q)\}$
6: **Fin Si**
7: **Fin Pour**
8: **retourne** $Reco$

Preuve 3.3.1 *(Algorithme 5)*

- *Montrons que si $q \preceq_\Gamma q'$ alors $max_{\leq_\Gamma}\{r \in q\} \leq_\Gamma max_{\leq_\Gamma}\{r' \in q'\}$.*
  Soit $MAX_q = max_{\leq_\Gamma}\{r \in q\}$, la référence de q telle que $\forall r \in q, MAX_q \geq_\Gamma r$.
  Comme $q \preceq_\Gamma q'$, par définition, $\exists r_x \in q' | r_x \geq_\Gamma MAX_q$.
  Soit $MAX_{q'} = max_{\leq_\Gamma}\{r' \in q'\}$, par définition, $MAX_{q'} \geq_\Gamma r_x$.
  Nous avons donc : $MAX_q \leq_\Gamma r_x \leq_\Gamma MAX_{q'}$.
- *Montrons que si $max_{\leq_\Gamma}\{r \in q\} \leq_\Gamma max_{\leq_\Gamma}\{r' \in q'\}$ alors $q \preceq_\Gamma q'$*
  Soit $MAX_q = max_{\leq_\Gamma}\{r \in q\}$, la référence de q telle que $\forall r \in q, MAX_q \geq_\Gamma r$.
  Soit $MAX_{q'} = max_{\leq_\Gamma}\{r' \in q'\}$, la référence de q' telle que $\forall r' \in q', MAX_{q'} \geq_\Gamma r'$.
  Comme $MAX_q \leq_\Gamma MAX_{q'}$, par définition, $\exists r_x \in q' | r_x \geq_\Gamma MAX_q$.
  Puisque $MAX_q \leq_\Gamma MAX_{q'}$, $\forall r \in q$, $r \leq_\Gamma MAX_{q'}$.
  Nous avons donc : $\forall r \in q$, $\exists r' \in q'$, tel que $q \preceq_\Gamma q'$.

3.3.4 Recommandation par défaut

Comme indiqué Section 3.2.4, l'ensemble de recommandations candidates peut être vide. Dans ce cas, nous proposons une recommandation par défaut à l'utilisateur.

Notre ensemble de sessions contenues dans le log peut être vu comme un graphe où chaque nœud est une requête ou une classe de requêtes, et les arcs sont les enchaînements de requêtes ou de classes. Ainsi, l'idée de Kleinberg, dans [Kle99a, Kle99b], qui considère le Web comme un graphe où les nœuds sont les pages web et les arcs sont les liens entre les pages, est applicable. En voulant montrer que le contenu des pages n'est pas le seul fournisseur d'information mais que la structure du graphe fournit aussi des informations utiles, il a introduit une procédure pour identifier les pages web qui sont de bons hubs (au sens, en français, de plate-forme de correspondance) ou de bonnes autorités, en réponse à un requête donnée où :

- *Autorités* sont les pages possédant un grand degré entrant, c'est-à-dire qu'elles sont pointées par un grand nombre de liens hypertexte,
- *Hubs* sont les pages qui ont des liens vers de multiples *autorités* pertinentes.

Ainsi, selon Kleinberg, un bon hub est une page qui référence de nombreuses pages qui sont des autorités. Une bonne autorité est une page qui contient des informations pertinentes pour l'utilisateur (Figure 3.2).

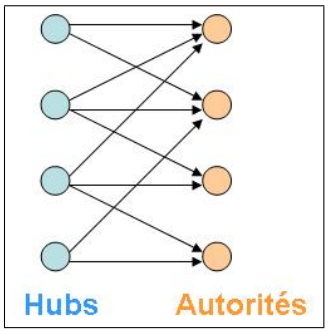


FIGURE 3.2 – Hubs et Autorités [Kle99a]

Exemple 3.3.3 *L'exemple suivant est souvent mentionné : Si nous considérons la requête "constructeurs automobiles", les pages d'accueil des sites de Peugeot, Ford, Toyota et autres constructeurs automobiles seront considérées comme de bonnes autorités tandis que les pages web qui listeront ces pages d'accueil seront de bons hubs.*

Dans notre contexte, un log étant vu comme un graphe, une autorité est un nœud du graphe pointé par de nombreux autres nœuds et un hub est un nœud qui pointe vers le maximum d'autorités.

Exemple 3.3.4 *Le log de sessions de requêtes présenté Annexe A peut être décrit par un graphe où les requêtes constitutives des sessions s_1, s_2 et s_3 sont les nœuds et les liens de séquencement entre les requêtes sont les arcs. Nous obtenons le graphe de la Figure 3.3 où q_1 et q_2^2 sont des hubs possibles car ils ont un maximum d'arcs sortants, q_3^2 est une autorité possible car elle a un maximum d'arcs entrants.*

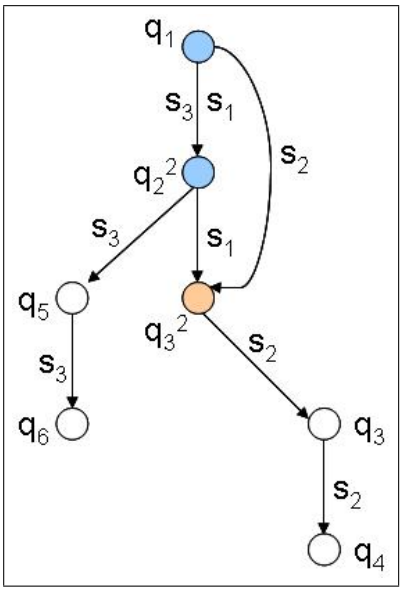


FIGURE 3.3 – Hubs et Autorités dans un log de sessions de requêtes

En se basant sur [Kle99a], nous proposons comme recommandation par défaut, le nœud du graphe faisant autorité (respectivement, le *hub*), c'est-à-dire, celui qui a le nombre le plus élevé de successeurs (respectivement, prédécesseurs).

Exemple 3.3.5 *Considérons le log de sessions de requêtes présenté Annexe A décrit par le graphe de la Figure 3.3 et supposons que la session courante soit telle qu'il soit impossible de proposer une recommandation. Si la recommandation par défaut choisie est "Autorité" alors* q_3^2 *sera proposée comme recommandation par défaut à la suite de la session courante.*

3.3.5 Exemples de combinaisons

Les sections précédentes nous ont permis d'entrevoir une liste non exhaustive des instanciations possibles de notre cadre générique de génération de recommandations. Dans cette section, nous présentons quatre combinaisons possibles ($ClusterH$, $ClusterSP$, EdH et $EdSP$) des différents paramètres de chacune des trois étapes du processus afin de proposer des recommandations à l'utilisateur.

Nous motivons notre choix de ces quatre combinaisons en indiquant qu'en se basant uniquement sur la liste non exhaustive des instanciations possibles, il existe plus de 80 [1] combinaisons possibles. Notre choix s'est donc porté sur des combinaisons qui nous paraissaient intéressantes et qui permettaient de comparer les deux distances entre références (d_h et d_{sp}).

1. Nous avons deux fonctions de prétraitement (k-médoïdes et identité), deux fonctions $Match$ ($Match_1$ et $Match_2$), cinq fonctions Rep (successeur, dernier, union, intersection, médoïde) et quatre fonctions d'ordonnancement (basée sur la proximité, la fréquence, les références et la personnalisation), soit : $2 \times 2 \times 5 \times 4 = 80$ combinaisons possibles.

3.3.5.1 ClusterH

Cette combinaison a été présentée dans [GMN08]. La distance entre références utilisée est la distance de Hamming d_h et la distance entre requêtes est $d^0_{queries}$.

1. Un prétraitement du log de sessions de requêtes est effectué grâce à l'algorithme de clustering des K-médoïdes, ce qui nous permet d'obtenir des classes de requêtes puis des sessions généralisées.
2. Puis la génération des recommandations se fait grâce à une fonction $Match_1$ recherchant les sous-séquences de la session généralisée courante qui coïncident avec les sessions généralisées du log et à la fonction $Rep^{Médoide}(Rep^{Succ})$ qui nous renvoie le médoïde de la classe succédant à la classe dont la position est renvoyée par $Match_1$. Nous obtenons ainsi un ensemble de requêtes candidates.
3. Enfin, ces requêtes sont ordonnées en fonction de leur proximité avec la dernière requête de la session courante.

Notons que la recommandation par défaut est le médoïde de la classe *hub*.

La matrice d'utilité $Sessions \times Requêtes$ assigne aux requêtes présentes dans la session, la valeur maximale max telle que $max = |s_c| + d^{max}_{queries}$ où $|s_c|$ est la longueur de la session courante s_c et $d^{max}_{queries}$ est la valeur maximale (démontrée Preuve 3.1.3) que peut atteindre la distance entre requêtes $d_{queries}$. Cette valeur maximale prend en compte le fait que la fonction $approximateStringMatching$, utilisée dans $Match_1$, aura pour coût maximum le nombre d'élément de la session courante. Elle prend aussi en compte le fait que l'ordonnancement est fait en fonction de la distance entre requêtes $d_{queries}$. La fonction d'utilité $u(s,q)$ calcule la différence entre la valeur maximale max et le coût de 'matching', $cost$, entre la session à laquelle appartient q, s' et la session s, additionné à la distance $d_{queries}$ entre le représentant de s, $Rep(s)$, et q, si q est le représentant de s'.

Formellement, $u(s,q) = \begin{cases} max - (cost(s,s') + d_{queries}(Rep(s),q)) & \text{si} \quad q = Rep(s') \\ 0 & \text{sinon} \end{cases}$

Exemple 3.3.6 *Nous illustrons cette combinaison par un exemple sur les données présentées Annexe A.*
Génération des recommandations :

1. *En appliquant la fonction de prétraitement : $Prétraitement(L)$ où $Prétraitement$ est l'algorithme des K-médoïdes, sur les sessions de requêtes s_1, s_2, s_3 contenues dans le log L, nous obtenons les quatre classes c_1, c_2, c_3, c_4 telles que $c_1 = \{q_1\}, c_2 = \{q^2_2, q_3, q^2_3\}, c_3 = \{q_4\}, c_4 = \{q_5, q_6\}$.*
   Nous remplaçons chaque requête des sessions par la classe à laquelle elle appartient et nous obtenons les trois sessions généralisées g_1, g_2, g_3 telles que $g_1 = \langle c_1, c_2, c_2\rangle, g_2 = \langle c_1, c_2, c_2, c_3\rangle, g_3 = \langle c_1, c_2, c_4, c_4\rangle$ où g_1 est la session généralisée de s_1, g_2 celle de s_2 et g_3 celle de s_3. Ces trois sessions généralisées forme l'ensemble $\mathscr{S}$ du log prétraité $\mathcal{L}$.
2. *Nous passons ensuite à l'étape de génération des requêtes candidates. Nous utilisons l'algorithme $GénérationRequêtesCandidates(s_c, \mathcal{L}$,k-medoides$, Match_1, Rep^{Médoide}(Rep^{Succ}), \infty)$.*
   Par conséquent, nous commençons par généraliser la session de requêtes courante. Le classifieur de requêtes nous indiquant que $q_2 \in c_2$, nous obtenons la session généralisée $PS = \langle c_1, c_2, c_2\rangle$ de s_c.
   La fonction $Match_1(PS, \mathscr{S}, \infty)$ recherchant les sous-séquences nous renvoie les trois paires suivantes : $\langle g_1, 3\rangle, \langle g_2, 3\rangle, \langle g_3, 2\rangle$ en tant que candidates.

Puis la fonction Rep recherchant le médoïde des successeurs nous renvoie le médoïde de $g_2[3] = c_3$ correspondant à la requête q_4 et le médoïde de $g_3[3] = c_4$ correspondant à la requête q_5.
Ainsi, l'ensemble non ordonné des requêtes candidates à la recommandation est : $\{q_4, q_5\}$.

3. *La dernière étape nous permet d'ordonner ces deux requêtes en fonction de leur proximité avec la dernière requête q_3 de la session courante s_c. Or, $d^0_{queries}(q_4, q_3)$ couplée avec la distance de Hamming entre références vaut 2 et $d^0_{queries}(q_5, q_3) = 3$. Nous obtenons l'ensemble ordonné de recommandations : $\{q_4, q_5\}$.*

Utilité :
La valeur assignée aux requêtes déjà présentes dans la session s_c est $max = |s_c| + d^{max}_{queries} = 3+4 = 7$. Nous obtenons la matrice :

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	7		7		7			
s_2	7			7	7	7		
s_3	7		7				7	7
s_c	7	7		7				

Nous calculons l'utilité des requêtes pour la session courante s_c : $u(s_c, q)$. L'étape 2 permet de donner aux requêtes q_2^2, q_3^2 et q_6, la valeur 0 car aucune de ces requêtes ne correspond au médoïde du successeur des sessions candidates. Par contre, $u(s_c, q_4)$ vaut $7-(cost(g_2, g_c)+d_{queries}(q_3, q_4)) = 7-(0+2) = 5$ sachant qu'aucun élément n'est enlevé à g_c pour coïncider avec g_2 et $u(s_c, q_5)$ vaut $7-(cost(g_3, g_c)+d_{queries}(q_3, q_5)) = 7-(2+3) = 2$ sachant qu'il est nécessaire d'enlever deux éléments à g_c pour coïncider avec g_3. Nous obtenons la matrice :

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	7		7		7			
s_2	7			7	7	7		
s_3	7		7				7	7
s_c	7	7	0	7	0	5	2	0

Nous retrouvons que l'ensemble ordonné des recommandations est $\{q_4, q_5\}$.
La Figure 3.4 illustre cet exemple.

3.3.5.2 ClusterSP

Cette combinaison a été présentée en partie dans [Neg09]. La distance entre références utilisée est la distance basée sur le plus court chemin d_{sp} et la distance entre requêtes est $d^0_{queries}$.
Les paramètres des étapes de prétraitement, génération et ordonnancement sont les mêmes que ceux de $ClusterH$.

La matrice d'utilité $Sessions \times Requêtes$ est obtenue de la même manière que celle de $ClusterH$.

Exemple 3.3.7 *Nous illustrons cette combinaison par un exemple sur les données présentées Annexe A.*
Génération des recommandations :

1. *En appliquant la fonction de prétraitement : Prétraitement(L) où Prétraitement est l'algorithme des K-médoïdes, sur les sessions de requêtes s_1, s_2, s_3 contenues dans le log L, nous*

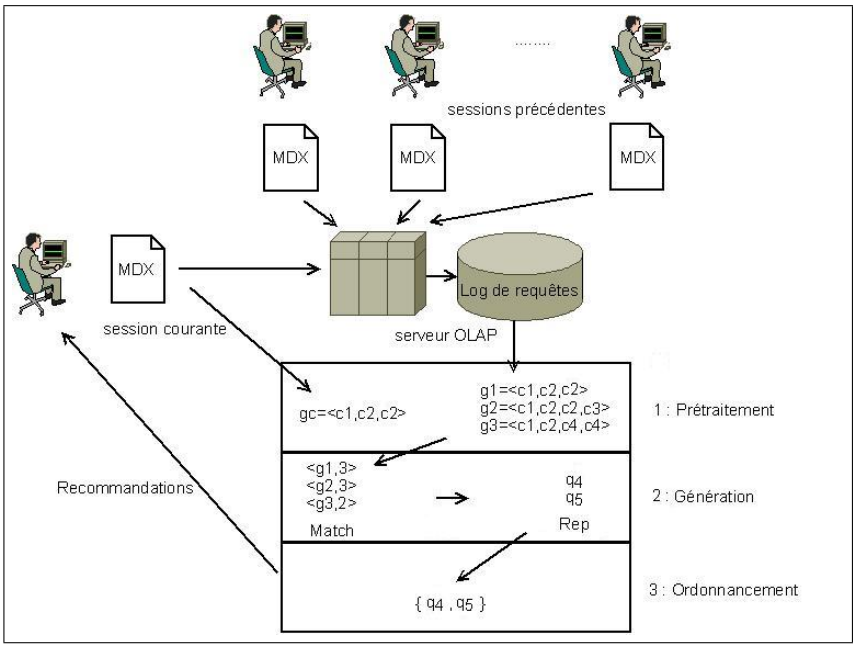


FIGURE 3.4 – Démarche de la combinaison *ClusterH*

obtenons les cinq classes c_1, c_2, c_3, c_4, c_5 *telles que* $c_1 = \{q_1\}, c_2 = \{q_2^2\}, c_3 = \{q_3, q_3^2\}, c_4 = \{q_4\}, c_5 = \{q_5, q_6\}$.
Nous remplaçons chaque requête des sessions par la classe à laquelle elle appartient et nous obtenons les trois sessions généralisées g_1, g_2, g_3 *telles que* $g_1 = \langle c_1, c_2, c_3\rangle, g_2 = \langle c_1, c_3, c_3, c_4\rangle, g_3 = \langle c_1, c_2, c_5, c_5\rangle$ *où* g_1 *est la session généralisée de* s_1, g_2 *celle de* s_2 *et* g_3 *celle de* s_3. *Ces trois sessions généralisées forme l'ensemble* $\mathscr{S}$ *du log prétraité* $\mathcal{L}$.

2. *Nous passons ensuite à l'étape de génération des requêtes candidates. Nous utilisons l'algorithme* $GénérationRequêtesCandidates(s_c, \mathcal{L}$,*k-medoides*$, Match_1, Rep^{Médoide}(Rep^{Succ}), \infty)$.
Par conséquent, nous commençons par généraliser la session de requêtes courante. Le classifieur de requêtes nous indiquant que $q_2 \in c_2$, *nous obtenons la session généralisée* $PS = \langle c_1, c_2, c_3\rangle$ *de* s_c.
La fonction $Match_1(PS, \mathscr{S}, \infty)$ *recherchant les sous-séquences nous renvoie les trois paires suivantes :* $\langle g_1, 3\rangle, \langle g_2, 2\rangle, \langle g_2, 3\rangle$ *en tant que candidates.*
Puis la fonction $Rep^{Médoide}(Rep^{Succ})$ *recherchant le médoïde des successeurs nous renvoie le médoïde de* $g_2[2] = c_3$ *correspondant à la requête* q_3^2 *et le médoïde de* $g_2[3] = c_4$ *correspondant à la requête* q_4.
Ainsi, l'ensemble non ordonné des requêtes candidates à la recommandation est : $\{q_3^2, q_4\}$.

3. *La dernière étape nous permet d'ordonner ces deux requêtes en fonction de leur proximité avec la dernière requête* q_3 *de la session courante* s_c. *Or,* $d^0_{queries}(q_3^2, q_3)$ *couplée avec la distance basée sur le plus court chemin vaut 2 et* $d^0_{queries}(q_4, q_3) = 4$.
Nous obtenons l'ensemble ordonné de recommandations : $\{q_3^2, q_4\}$.

Utilité :
La valeur assignée aux requêtes déjà présentes dans la session s_c *est* $max = |s_c| + d^{max}_{queries} = 3 + 26 = 29$. *Nous obtenons la matrice :*

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	29		29		29			
s_2	29			29	29	29		
s_3	29		29				29	29
s_c	29	29		29				

Nous calculons l'utilité des requêtes pour la session courante s_c *:* $u(s_c, q)$*. L'étape* 2 *permet de donner aux requêtes* q_2^2*,* q_5 *et* q_6*, la valeur* 0 *car aucune de ces requêtes ne correspond au médoïde du successeur des sessions candidates. Par contre,* $u(s_c, q_3^2)$ *vaut* $29 - (cost(g_2, g_c) + d_{queries}(q_3, q_3^2)) = 29 - (2+2) = 25$ *sachant qu'il est nécessaire d'enlever deux éléments à* g_c *pour coïncider avec* g_2 *et* $u(s_c, q_4)$ *vaut* $29 - (cost(g_2, g_c) + d_{queries}(q_3, q_4)) = 29 - (2+4) = 23$*. Nous obtenons la matrice :*

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	29		29		29			
s_2	29			29	29	29		
s_3	29		29				29	29
s_c	29	29	0	29	25	23	0	0

Nous retrouvons que l'ensemble ordonné des recommandations est $\{q_3^2, q_4\}$*.*
La Figure 3.5 illustre cet exemple.

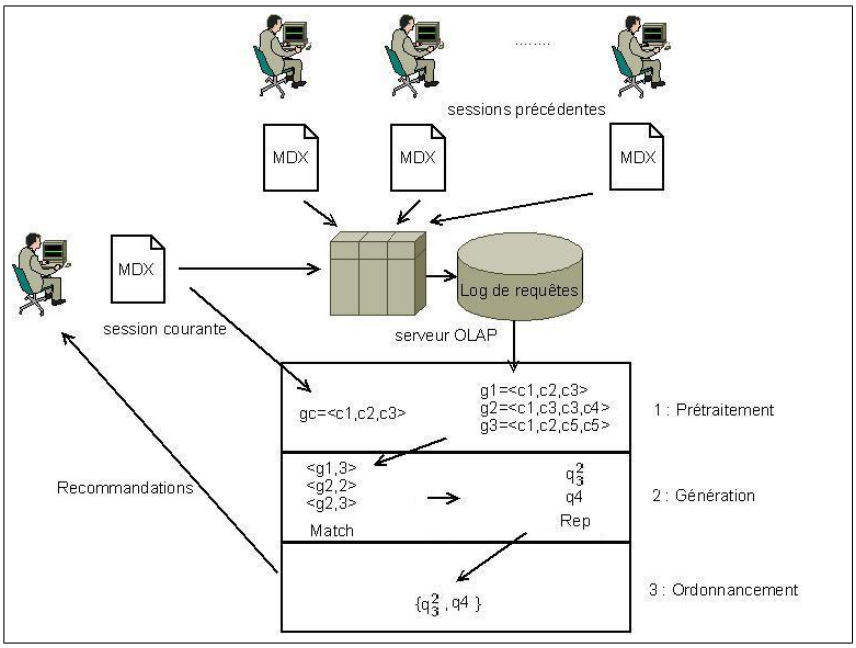


FIGURE 3.5 – Démarche de la combinaison *ClusterSP*

3.3.5.3 EdH

Dans cette combinaison, la distance entre références utilisée est la distance de Hamming d_h et la distance entre requêtes est $d^0_{queries}$.

1. Le log n'est pas prétraité : la fonction de prétraitement est donc l'identité ($Prétraitement = id$).
2. La génération des recommandations se fait grâce à la fonction $Match_2$ calculant la distance de Levenshtein entre les sessions de requêtes du log et la session de requêtes courante et à la fonction Rep^{Last} qui renvoie la dernière requête de chaque session de requêtes candidate renvoyée par $Match_2$. Nous obtenons ainsi un ensemble de requêtes candidates.
3. Enfin, ces requêtes sont ordonnées en fonction de leur proximité avec la dernière requête de la session courante.

La matrice d'utilité $Sessions \times Requêtes$ assigne aux requêtes présentes dans la session, la valeur maximale max telle que $max = d^{max}_{sessions} + d^{max}_{queries}$ où $d^{max}_{sessions}$ est la valeur maximale (démontrée Preuve 3.1.4) que peut atteindre la distance entre sessions $d_{sessions}$ et $d^{max}_{queries}$ est la valeur maximale (démontrée Preuve 3.1.3) que peut atteindre la distance entre requêtes $d_{queries}$. La fonction d'utilité $u(s, q)$ calcule la différence entre la valeur maximale max et la distance $d_{sessions}$ entre la session à laquelle appartient q, s' et la session s, additionné à la distance $d_{queries}$ entre le représentant de s, $Rep(s)$, et q, si q est le représentant de s'.

Formellement, $u(s,q) = \begin{cases} max - (d_{sessions}(s, s') + d_{queries}(Rep(s), q)) & \text{si} \quad q = Rep(s') \\ 0 & \text{sinon} \end{cases}$

Exemple 3.3.8 *Nous illustrons cette combinaison par un exemple sur les données présentées Annexe A.*
Génération des recommandations :

1. *La fonction de prétraitement étant l'identité, nous passons directement à la seconde étape.*
2. *Lors de l'étape de génération des requêtes candidates, nous utilisons l'algorithme* $GénérationRequêtesCandidates(s_c, \mathcal{L}, Id, Match_2, Rep^{Last}, \infty)$ *où* s_c *est la session de requêtes courante et* $\mathcal{S}$ *est l'ensemble de sessions de requêtes* s_1, s_2, s_3 *contenu dans le log.*
   La fonction $Match_2(s_c, \mathcal{S}, \infty)$ *recherchant les sessions de requêtes qui coïncident avec* s_c *via la distance de Levenshtein renvoie la session :* s_1 *en tant que candidate.*
   Puis la fonction Rep recherchant la dernière requête des sessions candidates renvoie la requête candidate q_4.
3. *La dernière étape d'ordonnancement n'est pas nécessaire ici puisqu'il n'y a qu'une seule requête candidate. Nous obtenons l'ensemble de recommandations :* $\{q^2_3\}$.

Utilité :
La valeur assignée aux requêtes déjà présentes dans la session s_c *est* $max = d^{max}_{sessions} + d^{max}_{queries} = ((d^{max}_{queries} \times |s_c|) + \alpha(|s_3| - |s_c|)) + d^{max}_{queries} = ((4 \times 3) + 2(4 - 3)) + 4 = 18$ *où* s_c *est la session la plus courte et* s_3 *la plus longue du log. Nous prenons* $\alpha = 2$ *(cette valeur de* α *sera justifiée dans le Chapitre 4). Nous obtenons la matrice :*

$u(s,q)$	q_1	q_2	q^2_2	q_3	q^2_3	q_4	q_5	q_6
s_1	18		18		18			
s_2	18			18	18	18		
s_3	18		18				18	18
s_c	18	18		18				

Nous calculons l'utilité des requêtes pour la session courante s_c : $u(s_c, q)$. L'étape 2 permet de donner aux requêtes q_2^2 et q_5, la valeur 0 car aucune de ces requêtes ne correspond à la dernière requête des sessions candidates. Par contre, $u(s_c, q_3^2)$ vaut $18 - (d_{sessions}(s_1, s_c) + d_{queries}(q_3, q_3^2)) = 18 - (2 + 1) = 15$, $u(s_c, q_4)$ vaut $18 - (d_{sessions}(s_2, s_c) + d_{queries}(q_3, q_4)) = 18 - (3 + 2) = 13$ et $u(s_c, q_6)$ vaut $18 - (d_{sessions}(s_3, s_c) + d_{queries}(q_3, q_6)) = 18 - (6 + 3) = 9$. Nous obtenons la matrice :

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	18		18		18			
s_2	18			18	18	18		
s_3	18		18				18	18
s_c	18	18	0	18	15	13	0	9

Nous retrouvons que l'ensemble ordonné des recommandations est $\{q_3^2\}$, voire $\{q_3^2, q_4\}$.
La Figure 3.6 illustre cet exemple.

3.3.5.4 EdSP

Cette combinaison a été présentée dans [GMN09]. La distance entre références utilisée est la distance basée sur le plus court chemin d_{sp} et la distance entre requêtes est $d^0_{queries}$.
Les paramètres des étapes de prétraitement, génération et ordonnancement sont les mêmes que ceux de *EdH*. La matrice d'utilité *Sessions* × *Requêtes* est obtenue de la même manière que celle de *EdH*.

Exemple 3.3.9 *Nous illustrons cette combinaison par un exemple sur les données présentées Annexe A.*
Génération des recommandations :

1. *La fonction de prétraitement étant l'identité, nous passons directement à la seconde étape.*
2. *Lors de l'étape de génération des requêtes candidates, nous utilisons l'algorithme $GénérationRequêtesCandidates(s_c, \mathcal{L}, Id, Match_2, Rep^{Last}, \infty)$ où s_c est la session de requêtes courante et $\mathcal{S}$ est l'ensemble de sessions de requêtes s_1, s_2, s_3 contenu dans le log.*
   La fonction $Match_2(s_c, \mathcal{S}, \infty)$ recherchant les sessions de requêtes qui coïncident avec s_c via la distance de Levenshtein renvoie la session : s_1 en tant que candidate.
   Puis la fonction Rep recherchant la dernière requête des sessions candidates renvoie la requête candidate q_3^2.
3. *La dernière étape d'ordonnancement n'est pas nécessaire ici puisqu'il n'y a qu'une seule requête candidate. Nous obtenons l'ensemble de recommandations : $\{q_3^2\}$.*

Utilité :
La valeur assignée aux requêtes déjà présentes dans la session s_c est $max = d^{max}_{sessions} + d^{max}_{queries} = ((d^{max}_{queries} \times |s_c|) + \alpha(|s_3| - |s_c|)) + d^{max}_{queries} = ((26 \times 3) + 3.2(4 - 3)) + 26 = 107.2$ où s_c est la session la plus courte et s_3 la plus longue du log. Nous prenons $\alpha = 3.2$ (cette valeur de α sera justifiée dans le Chapitre 4). Nous obtenons la matrice :

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	107.2		107.2		107.2			
s_2	107.2			107.2	107.2	107.2		
s_3	107.2		107.2				107.2	107.2
s_c	107.2	107.2		107.2				

Nous calculons l'utilité des requêtes pour la session courante s_c *:* $u(s_c, q)$*. L'étape* 2 *permet de donner aux requêtes* q_2^2 *et* q_5*, la valeur* 0 *car aucune de ces requêtes ne correspond à la dernière requête des sessions candidates. Par contre,* $u(s_c, q_3^2)$ *vaut* $107.2-(d_{sessions}(s_1, s_c)+d_{queries}(q_3, q_3^2)) = 107.2-(4+2) = 101.2$*,* $u(s_c, q_4)$ *vaut* $107.2-(d_{sessions}(s_2, s_c)+d_{queries}(q_3, q_4)) = 107.2-(7.2+4) = 96$ *et* $u(s_c, q_6)$ *vaut* $107.2-(d_{sessions}(s_3, s_c)+d_{queries}(q_3, q_6)) = 107.2-(11.2+6) = 90$*. Nous obtenons la matrice :*

$u(s,q)$	q_1	q_2	q_2^2	q_3	q_3^2	q_4	q_5	q_6
s_1	107.2		107.2		107.2			
s_2	107.2			107.2	107.2	107.2		
s_3	107.2		107.2				107.2	107.2
s_c	107.2	107.2	0	107.2	101.2	96	0	90

Nous retrouvons que l'ensemble ordonné des recommandations est $\{q_3^2\}$*, voire* $\{q_3^2, q_4\}$*. La Figure 3.6 illustre cet exemple.*

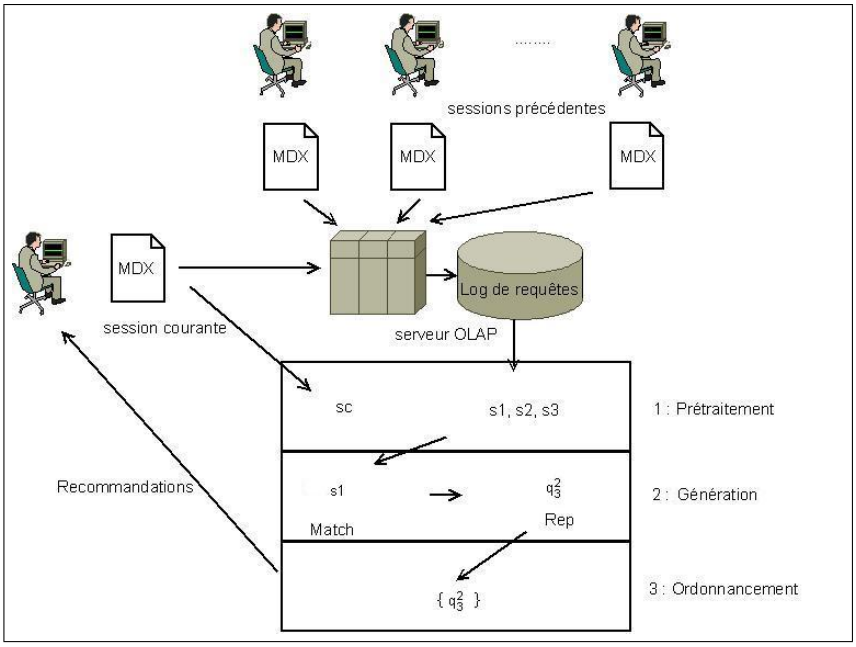


FIGURE 3.6 – Démarche des combinaisons *EdH* et *EdSP*

3.3.5.5 Remarques

La différence entre *ClusterH* et *ClusterSP* d'un coté et *EdH* et *EdSP* de l'autre, est uniquement due à la distance entre références utilisée, à savoir la distance de Hamming d_h pour *ClusterH* et *EdH* et la distance basée sur le plus court chemin d_{sp} pour *ClusterSP* et *EdSP*. De par la simplicité de calcul de d_h, *ClusterH* et *EdH* devraient renvoyer des résultats plus rapidement que *ClusterSP* et *EdSP*.

Par ailleurs, *ClusterH* et *ClusterSP* utilisent, comme fonction de prétraitement, l'algorithme des k-médoïdes alors que *EdH* et *EdSP* utilisent l'identité. De sorte que l'étape de prétraitement sera forcément beaucoup plus longue pour *ClusterH* et *ClusterSP*. De plus, *ClusterH* et

ClusterSP se limitent à une recherche de sous-séquences. Enfin, *ClusterH* et *ClusterSP* renvoient le médoïde du successeur des classes de requêtes qui coïncident avec la session courante alors que *EdH* et *EdSP* renvoient la dernière requête des sessions qui coïncident avec la session courante.

EdH et *EdSP* retournent le même ensemble ordonné de recommandations. Ceci est un cas particulier. En effet, l'exemple suivant montre que ces deux combinaisons ne renvoient pas forcément le même ensemble ordonné de recommandations.

Exemple 3.3.10 *Considérons les trois sessions s_c, s_1 et s_2 telles que $s_c = q_1$, $s_2 = q_1 \rightarrow q_4$ et $s_3 = q_1 \rightarrow q_2 \rightarrow q_3^2$ où les requêtes q_1, q_2, q_3^2 et q_4 sont celles présentées dans le log de l'Annexe A. Nous appliquons EdH et EdSP sur ces données :*

Combinaison	*EdH*	*EdSP*
Génération des recommandations		
$Match_2(s_c, \{s_1, s_2\}, \infty)$ $Reco^{Last}$	s_1 q_4	s_2 q_3^2
Utilité		
max	12	58.4
$u(s_c, q_2)$	*0*	*0*
$u(s_c, q_4)$	$= 12 - (d_{sessions}(s_1, s_c) + d_{queries}(q_1, q_4))$ $= 12 - (2 + 3)$ $= 7$	$= 58.4 - (d_{sessions}(s_1, s_c) + d_{queries}(q_1, q_4))$ $= 58.4 - (3.2 + 6)$ $= 49.2$
$u(s_c, q_3^2)$	$= 12 - (d_{sessions}(s_2, s_c) + d_{queries}(q_1, q_3^2))$ $= 12 - (4 + 2)$ $= 6$	$= 58.4 - (d_{sessions}(s_2, s_c) + d_{queries}(q_1, q_3^2))$ $= 58.4 - (6.4 + 2)$ $= 50$
Recommandation	$\{q_4\}$	$\{q_3^2\}$

Ainsi, EdH renvoie l'ensemble ordonné de recommandations $\{q_4\}$ alors que EdSP renvoie l'ensemble ordonné de recommandations $\{q_3^2\}$.

3.4 Synthèse

Dans ce chapitre, dans un premier temps, nous proposons :
- une distance entre références $d_{references}$ qui peut être soit la distance de Hamming d_h, soit une distance basée sur le plus court chemin entre deux membres d_{sp},
- une distance entre requêtes MDX $d_{queries}$ qui est la combinaison de deux distances entre requêtes, l'une basée sur les dimensions et l'autre étant la distance de Hausdorff,
- une distance entre sessions $d_{sessions}$ qui est la distance de Levenshtein.

Dans un second temps, nous proposons un cadre générique de génération de recommandations de requêtes MDX. Ce cadre est générique dans le sens où il peut être instancié de différentes manières afin de changer la méthode de calcul des recommandations. Ce cadre est constitué de trois étapes paramétrables :

1. Prétraitement de l'ensemble des sessions de requêtes du log,
2. Génération des recommandations candidates,
3. Ordonnancement des recommandations candidates.

Ainsi, notre cadre de génération de recommandations propose des requêtes MDX recommandées à l'utilisateur afin de l'aider dans son exploration du cube de données.

Dans un troisième temps, nous présentons différents paramètres pour chacune de ces trois étapes :

1. Nous proposons deux fonctions de prétraitement, l'une utilisant l'algorithme des k-médoïdes, l'autre étant l'identité.
2. L'étape de génération est décomposée en deux parties :
   (a) Nous proposons d'abord deux fonctions de *matching*, l'une permettant d'obtenir les sessions qui coïncident avec la session courante et à quelle position de la session du log a lieu cette coïncidence et l'autre renvoyant la (ou les) session(s) la (les) plus proche(s) de la session courante au sens de la distance entre sessions.
   (b) Nous proposons ensuite cinq représentants de session, à savoir successeur, dernier, union, intersection ou médoïde.
3. Nous proposons enfin deux méthodes d'ordonnancement des requêtes candidates, l'une basée sur la proximité entre la requête à recommander et la requête qui représente la session courante (au sens de la distance entre requêtes), l'autre basée sur le profil de l'utilisateur.

Enfin, nous présentons quatre combinaisons possibles de ces différents paramètres d'étapes dont le tableau suivant fait un récapitulatif.

Combinaison	Distances ($d_{references}$)	Prétraitement	Génération		Ordonnancement (ordre sur les requêtes)
			Match	Rep	
ClusterH	d_h	K-médoïdes	Sous Séquences	Médoïde du successeur	Proximité
ClusterSP	d_{sp}	K-médoïdes	Sous Séquences	Médoïde du successeur	Proximité
EdH	d_h	Identité	Distance de Levenshtein	Dernier	Proximité
EdSP	d_{sp}	Identité	Distance de Levenshtein	Dernier	Proximité

Notons que les performances de ces quatre combinaisons seront comparées dans le chapitre suivant.

En conclusion, nous proposons une méthode de recommandation qui calcule l'utilité d'une requête pour la session courante selon le principe suivant. A partir d'un log de sessions de requêtes MDX, du schéma et d'une instance de la base de données multidimensionnelle, il s'agit de trouver les sessions du log qui sont proches de la session courante (au sens d'une distance entre sessions), de garder pour chacune de ces sessions, la requête qui la représente le mieux et d'ordonner les requêtes obtenues selon un ordre défini.
Notre méthode de recommandation collaborative utilise soit un modèle, soit une heuristique pour l'extrapolation des scores. Elle a pour objectif de recommander les requêtes MDX qui peuvent aider au mieux l'utilisateur dans son exploration du cube de données.

Le tableau comparatif initié Section 2.4.4 du Chapitre 2 dont nous ne conservons que les travaux s'appliquant aux bases de données multidimensionnelles devient le tableau 3.7 suivant :

Référence			[JRTZ09]	[Sar99], [Sar00], [SS01]	[CCD+08]	[Sap99], [Sap00]	*Reco*OLAP [GMN08]	*Reco*OLAP [GMN09]
Entrée (de l'algorithme de recommandation)	Log					×	×	×
	Schéma de BD		×	×	×	×	×	×
	Instance de BD		×	×	×	×	×	×
	Session courante		×	×	×	×	×	×
	Profil		×					
Sortie (de l'algorithme de recommandation)			Requête	Ensemble de tuples	Ensemble de tuples	Requête	Ensemble de requêtes	Ensemble de requêtes
Algorithme	Approche	Contenu	×	×	×			
		Collaboratif				×	×	×
		Hybride						
	Filtre	Estimation des scores	Modèle		×		×	
			Heuristique	×		×		×
			Technique	Théorie des graphes	Voisinage	Test statistique	Modèle statistique	Voisinage
	Génération		Calcul	Sélection	Sélection	Sélection	Sélection	Sélection
Guide					×	×	×	×

FIGURE 3.7 – Comparatif des travaux sur l'aide à l'exploration dans les bases de données multidimensionnelles

Chapitre 4

Réalisations et Tests

Sommaire

Ce chapitre présente l'environnement *Reco*OLAP développé afin de valider et de rendre opérationnel le cadre général de recommandation de requêtes que nous avons présenté. L'objectif est d'offrir un environnement recommandant à un utilisateur donné, des requêtes MDX en fonction d'une session d'analyse courante et d'un log de sessions de requêtes MDX.

Ce chapitre est organisé comme suit. Nous décrivons, dans un premier temps, l'architecture du système développé dans la Section 4.1, puis les données utilisées lors des expérimentations dans la Section 4.2. Enfin, quelques-unes des expérimentations et leurs résultats sont présentés dans la Section 4.3.

4.1 Le système

Le système de recommandation fonctionne de la manière suivante : à partir d'un log de sessions de requêtes MDX et d'une session courante de requêtes MDX, en appliquant l'algorithme de recommandation selon certains paramètres définis, le système propose un ensemble ordonné de requêtes MDX pouvant être recommandées à la suite de la session courante.

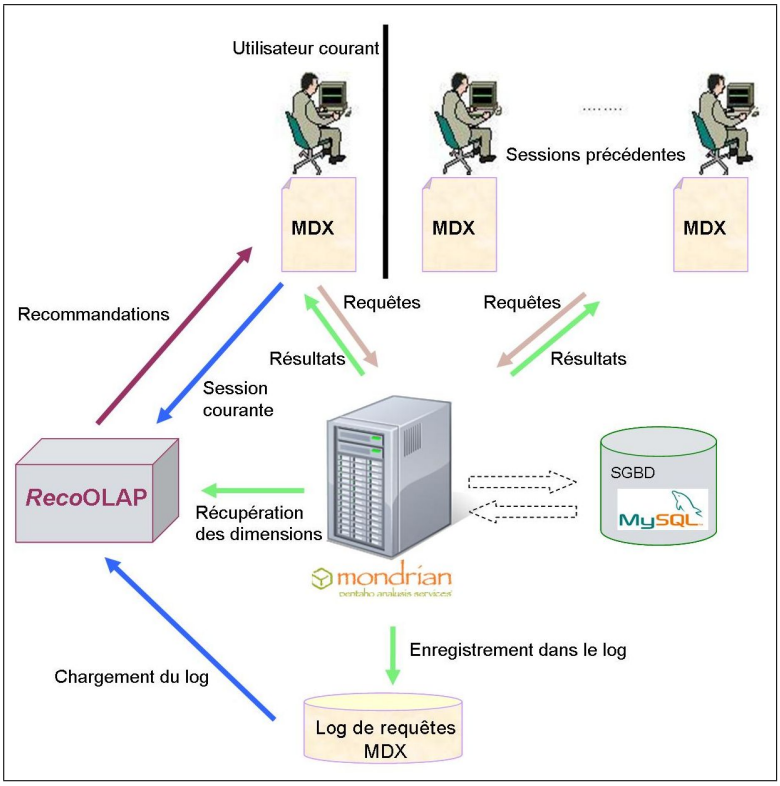


FIGURE 4.1 – Architecture du système *Reco*OLAP

La Figure 4.1 illustre l'architecture de notre système. Premièrement, chaque requête lancée par un utilisateur sur le cube de données, stocké dans le SGBD *MySQL* ([Sun08]), via le serveur OLAP *Mondrian* ([Pen09]) obtient une réponse. Deuxièmement, les sessions de requêtes des utilisateurs précédents ont été enregistrées dans un log de sessions de requêtes. La session de requêtes de l'utilisateur courant : la session courante, ainsi que les sessions de requêtes du log sont chargées dans l'application de génération de recommandations : *Reco*OLAP. Pendant le processus de recomman-

dation, *Reco*OLAP accède aux informations contenues dans le serveur *OLAP Mondrian*. Enfin, le système retourne à l'utilisateur l'ensemble ordonné des recommandations.

Notons que l'application pour la recommandation de requêtes MDX est développée en Java sous JRE 1.6.0_13 avec *MySQL* 5.1.28 et *Mondrian* 3.0.4.11371. Tous les tests ont été réalisés avec un Core 2 Duo - E4600 à 4GB de RAM utilisant Linux CentOS5.

4.2 Les données

Nous avons donc besoin de logs de requêtes. Par conséquent, nous utilisons des données synthétiques produites avec notre propre générateur de données, développé lui aussi en Java. Pour cela, nous générons des ensembles de sessions sur la base de données : FoodMart fournie avec le moteur OLAP Mondrian [Pen09].

Notre générateur utilise les paramètres suivants : un nombre (X) de sessions dans le log, un nombre maximal (Y) de requêtes par session, un nombre (Z) de dimensions dans le pool de départ [1] pour jouer sur la densité du log, un seuil (β) de comparaison de cellules.

Enfin, notre générateur simule une analyse pilotée par la découverte, comme présentée Section 2.1.3.1.

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 parmi le pool des Z dimensions de départ, 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 ($y < Y$) suivantes sont obtenues ainsi, s'il reste des dimensions à ajouter :
2. Soit q la requête générée à l'étape précédente, détection dans le résultat de q des paires de cellules ne variant que d'un membre dont la différence de valeurs dépasse le seuil β.
3. Obtention d'un ensemble de membres par application d'un opérateur, choisi aléatoirement parmi ceux proposés par [Sar99, Sar00, SS01] à savoir, $DIFF$, $EXCEP$ et $RELAX$ (présentés Section 2.4), sur l'une des paires de cellules détectées, elle aussi choisie aléatoirement.
4. Pour chaque membre de cet ensemble, choix aléatoire entre la sélection du membre lui-même ou la sélection de tous les membres situés au même niveau que lui dans la hiérarchie. Obtention d'un ensemble de membres à sélectionner par dimension.
5. Modification de q en substituant, dimension par dimension, les membres présents dans q par les membres à sélectionner, ou

6. Dans le cas où aucune explication n'a été trouvée, c'est-à-dire que l'ensemble de membres est vide, modification de q en tirant aléatoirement une dimension parmi celles qui ne sont

1. Le pool de départ est l'ensemble des Z premières dimensions du cube. Plus le nombre Z sera petit, plus la densité des logs sera importante, puisque le nombre de possibilités pour le choix des dimensions à faire apparaître dans la requête de départ sera restreint.

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 dont la différence de valeurs dépasse β. 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.

Exemple 4.2.1 *Afin d'illustrer cet algorithme de génération de sessions de requêtes MDX, voici un exemple de génération d'une session de trois requêtes parmi les 13 dimensions du cube Sales de la base FoodMart de Mondrian :*

1. *Création de la première requête :*
   - *Tirage aléatoire de deux dimensions : Obtention de 'Education Level' et 'Promotion Media'*
   - *Membre situé au niveau le plut haut de la hiérarchie de chaque dimension : 'All Education Level' pour 'Education Level' et 'All Media' pour 'Promotion Media'*
   - *Sélection des fils du membre situé au niveau le plus haut : '[Education Level].[All Education Level].Children' et '[Promotion Media].[All Media].Children'*

   Ainsi, la première requête est :

   SELECT {[Education Level].[All Education Level].Children} ON COLUMNS,
   {[Promotion Media].[All Media].Children} ON ROWS
   FROM [Sales]

   Le tableau croisé correspondant est :

	Education Level				
Promotion Media	Bachelors Degree	Graduate Degree	High School Degree	Partial College	Partial High School
Bulk Mail	1 333	266	1 061	326	1 334
Cash Register Handout	1 642	583	2 040	466	1 966
Daily Paper	1 920	553	2 372	621	2 272
Daily Paper, Radio	1 698	296	2 190	593	2 114
Daily Paper, Radio, TV	2 534	496	2 526	697	3 260
In-Store Coupon	925	247	1 064	377	1 185
No Media	50 292	11 409	58 129	18 543	57 075
Product Attachment	2 161	428	2 221	650	2 084
Radio	609	143	638	226	838
Street Handout	1 766	285	1 514	578	1 610
Sunday Paper	1 048	149	1 227	411	1 504
Sunday Paper, Radio	1 476	345	1 760	534	1 830
Sunday Paper, Radio, TV	630	172	876	230	818
TV	805	198	1 046	293	1 265

2. *Génération de la seconde requête :*
   - *Détection en ligne dans la requête précédente de paires de cellules 'anormales' et choix aléatoire de l'une d'elles : les cellules '58129' et '11409', correspondant respectivement au croisement des membres 'High School Degree'×'No Media' et 'Graduate Degree'×'No Media', contiennent des valeurs éloignées au sens du seuil β défini, par conséquent, elles constituent des cellules 'anormales'.*
   - *Tirage aléatoire d'une opération à appliquer : Obtention de $DIFF$*
   - *Obtention d'un ensemble de membres : le membre 'CA' de la dimension 'Store' constitue l'ensemble des membres.*
   - *Tirage aléatoire entre le membre ou les membres situés au même niveau que le membre : Obtention des membres situés au même niveau que le membre*
   - *Substitution des membres : '[Store].[All Stores]' est remplacé par '[Store].[All Stores].[USA].Children'*

Ainsi, la seconde requête est :

```
SELECT  Crossjoin({[Education Level].[All Education Levels].Children},
        {[Store].[All Stores].[USA].Children})                    ON COLUMNS,
        {[Promotion Media].[All Media].Children}                  ON ROWS
FROM    [Sales]
```

Le tableau croisé correspondant est :

	Education Level														
	Bachelors Degree			Graduate Degree			High School Degree			Partial College			Partial High School		
	Store			Store			Store			Store			Store		
Promotion Media	CA	OR	WA	CA	OR	WA	CA	OR	WA	CA	OR	WA	CA	OR	WA
Bulk Mail	1 040		293	196		70	826		235	175		151	896		438
Cash Register Handout	458	367	817	214	182	187	588	531	921	165	84	217	530	531	905
Daily Paper	927	126	867	202	54	297	1 035	189	1 148	300	4	317	1 203	87	982
Daily Paper, Radio	136	854	708	28	198	70	171	1 105	914	62	297	234	114	1 210	790
Daily Paper, Radio, TV	1 060	515	959	219	159	118	1 002	802	722	214	222	261	1 395	694	1 171
In-Store Coupon	9	279	637	21	47	179		335	729	32	116	229	37	406	742
No Media	13 651	12 149	24 492	2 845	3 409	5 155	14 698	15 913	27 518	4 961	5 176	8 406	15 316	15 822	25 937
Product Attachment	615	689	857	152	143	133	457	900	864	196	222	232	599	823	662
Radio	109		500	20		123	200		438	85		141	223		615
Street Handout	484		1 282	25		260	410		1 104	87		491	321		1 289
Sunday Paper	367	114	567	78		71	419	180	628	134	37	240	741	86	677
Sunday Paper, Radio	293	395	788	32	65	248	323	526	911	78	160	296	339	600	891
Sunday Paper, Radio, TV	161		469	16		156	192		684	40		190	107		711
TV	526	279		126	72		810	221	15	236	52	5	1 021	232	12

3. *Génération de la troisième requête :*
   - *Détection en ligne dans la requête précédente de paires de cellules 'anormales' et choix aléatoire de l'une d'elles : les cellules '27518' et '25937', correspondant respectivement au croisement des membres 'High School Degree, WA'× 'No Media' et 'Partial High School, WA'× 'No Media', contiennent des valeurs éloignées au sens du seuil β défini, par conséquent, elles constituent des cellules 'anormales'.*
   - *Tirage aléatoire d'une opération à appliquer : Obtention de $EXCEP$*
   - *Obtention d'un ensemble de membres : les membres '23688' de la dimension 'Store Size in SQFT', 'Gourmet Supermarket' de la dimension 'Store Type' et 'CA' de la dimension 'Store' constituent l'ensemble des membres.*
   - *Tirage aléatoire entre le membre ou les membres situés au même niveau que le membre : Obtention du membre*
   - *Substitution des membres : '[Store Size in SQFT].[All Store Size in SQFTs]' est remplacé par '[Store Size in SQFT].[All Store Size in SQFTs].[23688]', '[Store Type].[All Store Types]' est remplacé par '[Store Type].[All Store Types].[Gourmet Supermarket]' et '[Store].[All Stores].[USA].Children' est remplacé par '[Store].[All Stores].[USA].[CA]'.*

   Ainsi, la troisième requête est :

```
SELECT  Crossjoin({[Education Level].[All Education Levels].Children},
        Crossjoin({[Store].[All Stores].[USA].[CA]},
        Crossjoin({[Store Size in SQFT].[All Store Size in SQFTs].[23688]},
        {[Store Type].[All Store Types].[Gourmet Supermarket]})))  ON COLUMNS,
        {[Promotion Media].[All Media].Children}                  ON ROWS
FROM    [Sales]
```

Le tableau croisé correspondant est :

	Education Level				
	Bachelors Degree	Graduate Degree	High School Degree	Partial College	Partial High School
	Store	Store	Store	Store	Store
	+CA	+CA	+CA	+CA	+CA
	Store Size in SQFT	Store Size in SQFT	Store Size in SQFT	Store Size in SQFT	Store Size in SQFT
	23688	23688	23688	23688	23688
	Store Type	Store Type	Store Type	Store Type	Store Type
Promotion Media	Gourmet Supermarket	Gourmet Supermarket	Gourmet Supermarket	Gourmet Supermarket	Gourmet Supermarket
Bulk Mail	520	79	394	85	434
Cash Register Handout	132	75	149	74	84
Daily Paper	335	86	408	73	377
Daily Paper, Radio	136	28	171	62	114
Daily Paper, Radio, TV	284	70	129	40	376
In-Store Coupon					
No Media	3 596	619	4 026	1 508	4 302
Product Attachment	273	79	269	95	297
Radio					
Street Handout	87	19	67	44	41
Sunday Paper	147	18	170	87	431
Sunday Paper, Radio					
Sunday Paper, Radio, TV					
TV	146	17	112		168

4.3 Expérimentations et Résultats

Dans cette section, nous présentons les expérimentations menées sur les quatre instanciations de notre cadre générique de recommandation de requêtes MDX présentées Section 3.3.5, à savoir $ClusterH$, $ClusterSP$, EdH et $EdSP$.

Notre première expérience est une analyse de performance et consiste à évaluer l'efficacité de l'instanciation en terme de temps nécessaire à la proposition d'une recommandation. La seconde, quant à elle, est une analyse de rappel / précision et consiste à évaluer l'instanciation en terme de rappel, précision et F-mesure.

4.3.1 Analyse de performance

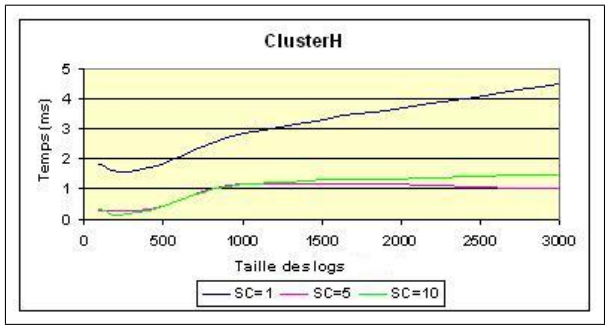
Notre première expérience évalue le temps nécessaire pour générer une recommandation et se décompose en deux tests.

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

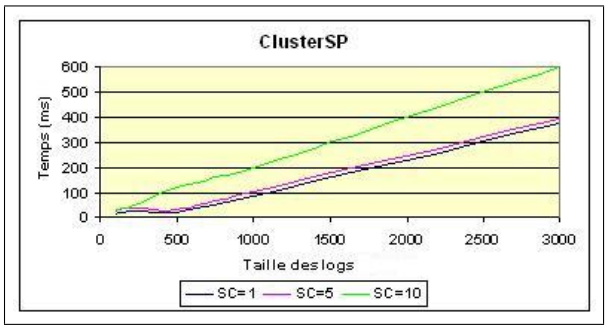
De même, pour EdH et $EdSP$ qui utilisent la distance de Levenshtein, pour recommander une requête pour une session s, le système mesure seulement le temps de comparaison de la dernière requête de la session s ($s[|s|]$) avec chaque requête du log. Cela est possible car les distances calculées précédemment pour $s \setminus \{s[|s|]\}$ ont été gardées en mémoire.

Différentes tailles de log

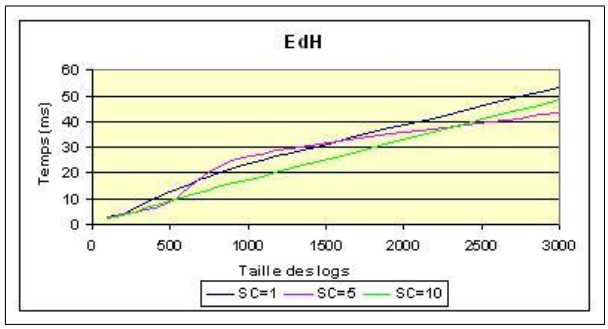
Le temps est calculé pour différentes tailles de log (en terme de nombre de requêtes). Les résultats sont présentés Figure 4.2 en fonction des différentes tailles de log. Les tailles de log sont les tailles réelles des logs, c'est-à-dire le nombre total de requêtes contenues dans le log. La recommandation est calculée pour chacun de ces logs, pour des sessions courantes SC de tailles variées (le nombre de requêtes par session courante est égal à 1, 5 ou 10), générées elles aussi par notre générateur de sessions.



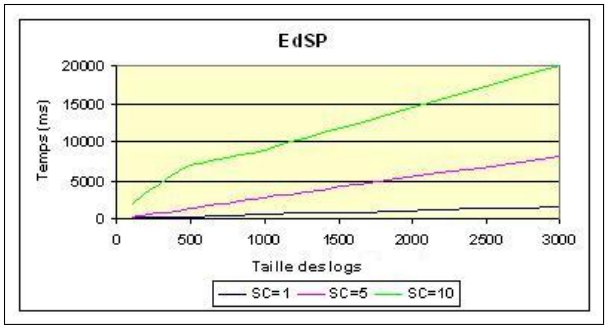
(a) Performance de ClusterH en fonction de la taille des logs



(b) Performance de ClusterSP en fonction de la taille des logs



(c) Performance de EdH en fonction de la taille des logs



(d) Performance de EdSP en fonction de la taille des logs

FIGURE 4.2 – Analyse de performance (temps de génération d'une recommandation) de ClusterH, ClusterSP, EdH et EdSP en fonction de la taille des logs

La Figure 4.2 montre que, quelle que soit la méthode utilisée, le temps nécessaire pour générer une recommandation augmente linéairement avec la taille des logs mais reste acceptable sauf pour *EdSP* qui atteint les 20 secondes et qui devra être optimisé. Les Figures 4.2(b) et 4.2(d) indiquent 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. De même, le temps de calcul de la distance basée sur le plus court chemin (d_{sp}) est toujours plus long que celui de la distance de Hamming (d_h). Enfin, *ClusterH* et *ClusterSP* sont plus rapides que *EdH* et *EdSP*. Remarquons tout de même que *ClusterH*, *ClusterSP* et *EdH* ne dépassent pas la seconde pour générer une recommandation à la suite d'une session courante dont la taille peut atteindre 10 requêtes.

Différentes densités de log

Le temps est calculé pour différentes densités de log. Les résultats sont présentés Figure 4.3 en fonction des différentes densités de logs. Les densités des logs sont obtenues en faisant varier le paramètre Z (nombre de dimensions dans le pool de départ). Z pouvant varier entre 2 et 13 (nombre total de dimensions du fait *Sales* de la base de données *FoodMart*) et prend les valeurs : 5 (forte densité), 9 (moyenne densité) ou 13 (faible densité). La recommandation est calculée pour chacun de ces logs, pour des sessions courantes SC de tailles variées (le nombre de requêtes par session

courante est égal à 1, 5 ou 10), générées elles aussi par notre générateur de sessions.

La Figure 4.3 montre que, quelle que soit la méthode utilisée, le temps nécessaire pour générer une recommandation est influencé par la densité des logs mais reste toujours acceptable sauf pour *EdSP*. Comme pour la Figure 4.2, nous retrouvons le fait que le temps d'exécution augmente lorsque la taille du log augmente et qu'il est légèrement influencé par la taille de la session courante pour *ClusterSP* et *EdSP*. De la même manière, nous remarquons de nouveau que *ClusterH*, *ClusterSP* et *EdH* ne dépassent pas la seconde pour générer une recommandation à la suite d'une session courante pouvant contenir jusqu'à 10 requêtes.

4.3.2 Rappel / Précision

Nous présentons dans cette section, la seconde expérimentation que nous avons menée, dans laquelle nous utilisons une validation croisée sur 10 échantillons (de l'anglais, *10-fold cross validation*) pour évaluer nos instanciations, dans le même esprit que la validation expérimentale réalisée dans [CEP09].

L'ensemble de sessions généré est partitionné en dix sous-ensembles de tailles égales 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* et est notée q_{at}.

Nous évaluons le rappel et la précision [BYRN99] des recommandations en utilisant les métriques suivantes :

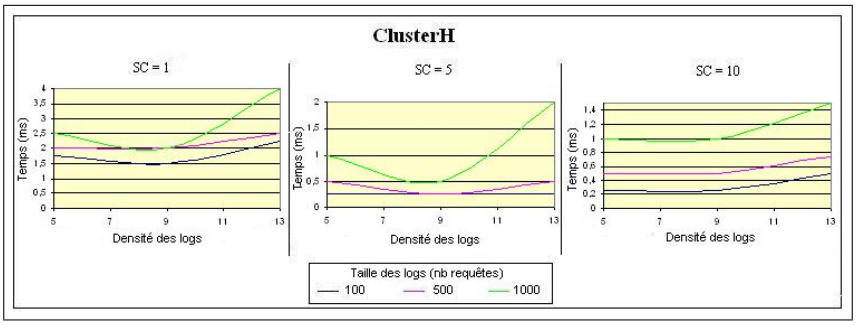
- $Précision = \frac{|membres(q_{at}) \cap membres(q_{rec})|}{|membres(q_{rec})|}$,
- $Rappel = \frac{|membres(q_{at}) \cap membres(q_{rec})|}{|membres(q_{at})|}$.

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 tailles : 936 requêtes (200 sessions) pour le log de forte densité, 902 requêtes (200 sessions) pour le log de densité moyenne et 979 requêtes (200 sessions) pour le log de densité faible. Nous prenons des logs de tailles légèrement plus petites car la *10-fold cross validation* prend beaucoup de temps.

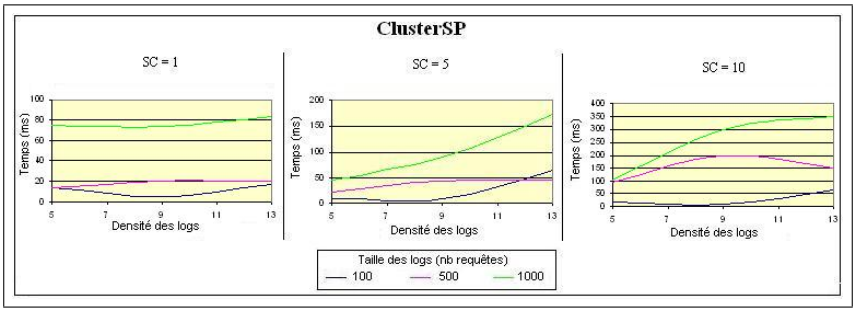
La première expérience nous permet de régler notre système en choisissant pour α et γ, les valeurs qui obtiennent les meilleurs rappel et précision, α étant le coût de l'opération d'ajout (ou de suppression) utilisé dans la distance d'édition (définie Section 3.1.3.1) pour les instanciations *EdH* et *EdSP* et γ étant utilisé dans la distance entre requêtes $d^{\gamma}_{queries}$.

Recherche de α pour *EdH* et *EdSP*

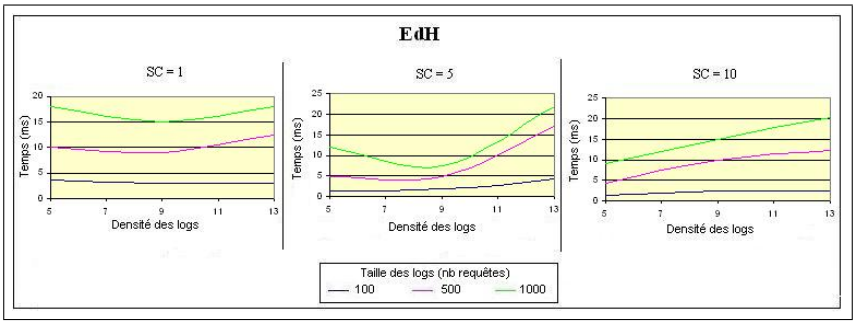
D'après la Propriété 3.1.1, α varie entre 0 et $d^{max}_{queries} = 74$, cependant, seules les valeurs de α permettant d'obtenir une précision supérieure à 0.7 sont affichées, par souci de lisibilité. Ainsi, la Figure 4.4 montre qu'une précision supérieure à 0.78 est obtenue pour $\alpha < 2$ pour *EdH* (Figure 4.4(a)) et pour $\alpha < 3.5$ pour *EdSP* (Figure 4.4(b)). Le meilleur rapport Rappel / Précision est obtenu lorsque les deux courbes sont le plus proche possible voire confondues, c'est le cas pour *EdH*



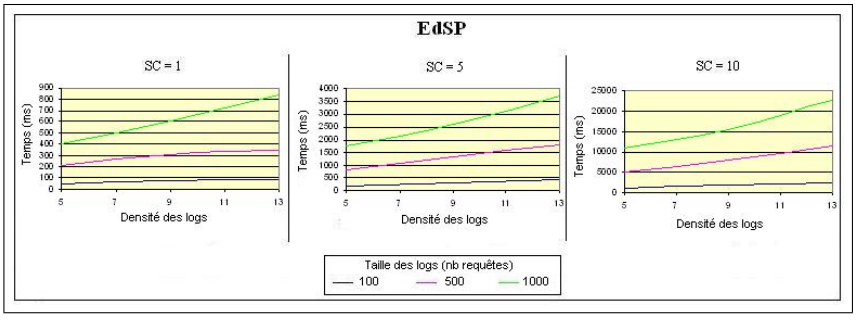
(a) Performance de ClusterH en fonction de la densité des logs



(b) Performance de ClusterSP en fonction de la densité des logs

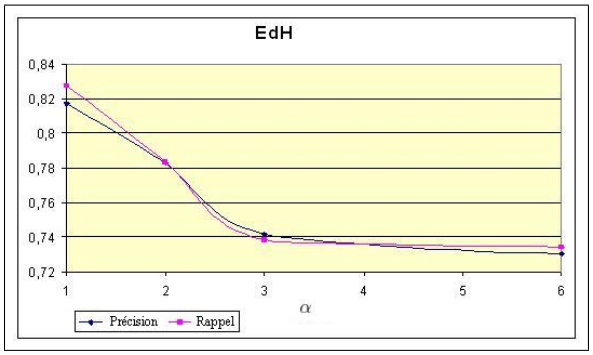


(c) Performance de EdH en fonction de la densité des logs

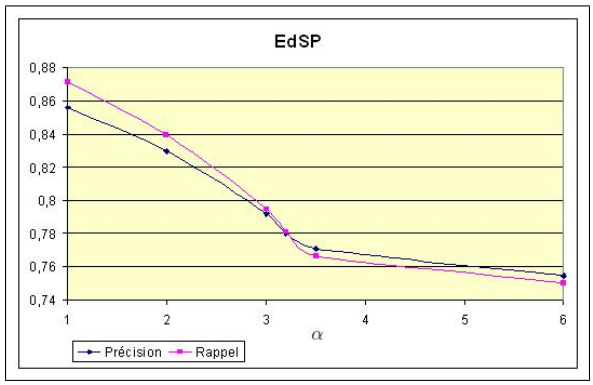


(d) Performance de EdSP en fonction de la densité des logs

FIGURE 4.3 – Analyse de performance (temps de génération d'une recommandation) de ClusterH, ClusterSP, EdH et EdSP en fonction de la densité des logs



(a) Précision et Rappel de EdH pour une densité de logs faible en fonction de α



(b) Précision et Rappel de EdSP pour une densité de logs faible en fonction de α

FIGURE 4.4 – Recherche de α pour EdH et EdSP

lorsque α vaut 2 et pour $EdSP$ lorsque α vaut 3.2. Ainsi, dans les expériences suivantes, la valeur de α est fixée à 2 pour EdH et à 3.2 pour $EdSP$.

Recherche de γ

Le rappel et la précision sont calculés avec $\gamma = 0, 0.5$ ou 1. La Figure 4.5 montre que les plus mauvais résultats sont obtenus pour $\gamma = 1$ c'est-à-dire quand la distance entre requêtes tient uniquement compte du nombre de dimensions qui diffèrent (cf. Section 3.1). Pour 0 et 0.5, les courbes sont confondues. Ainsi, cette figure montre que le fait de prendre en compte les dimensions qui diffèrent, contribue négativement à la distance entre requêtes.

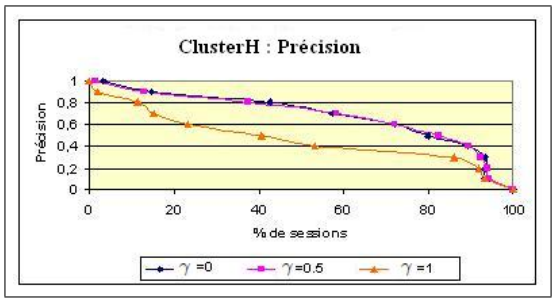
Rappel / Précision

Les Figures 4.6 et 4.7 montrent l'inverse de la distribution de fréquences cumulées[2] (inverse CFD) des valeurs enregistrées de rappel, précision et / ou F-mesure[3] 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 $\geq y$.

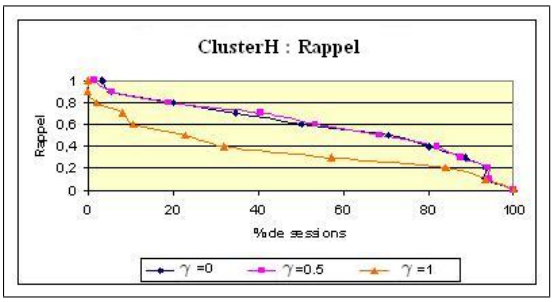
La Figure 4.6 montre l'inverse CFD de la précision et du rappel des recommandations calculée avec α et γ fixés par les tests précédents pour chaque instanciation. Les résultats démontrent l'efficacité de notre méthode puisque, quelle que soit la méthode utilisée, la précision atteint 0.8 pour plus de 80% des sessions pour des logs denses, et pour plus de 45% dans le cas de logs peu denses. Remarquons que la Figure 4.6(a) montrent que $ClusterH$ obtient une précision supérieure à 0.8

2. La distribution des fréquences cumulées est aussi connue sous le nom de fonction de répartition. En théorie des probabilités ou en statistiques, la fonction de répartition d'une variable aléatoire réelle caractérise la loi de probabilité de cette variable aléatoire réelle. La fonction de répartition de la variable aléatoire réelle x est la fonction F_X qui à tout réel x associe $F_X(x) = \mathbb{P}(X \leq x)$, où le membre de droite représente la probabilité que la variable aléatoire réelle X prenne une valeur inférieure ou égale à x. Son inverse, notée F^{-1}, est définie par $F^{-1}(p) = inf\{x \in \mathbb{R}, F(x) \geq p\}, p \in]0, 1]$.

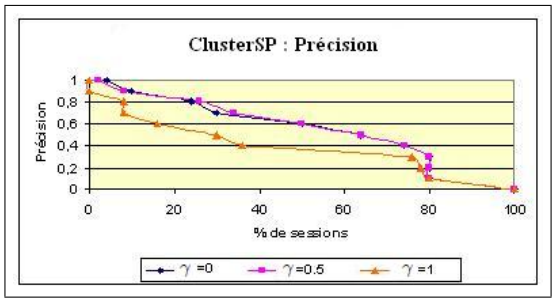
3. La F-mesure [vR79], $F = \frac{2.(\text{Précision} \cdot \text{Rappel})}{(\text{Précision} + \text{Rappel})}$, est une mesure de qualité.



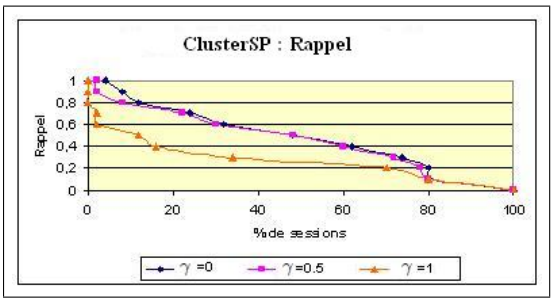
(a) Précision de ClusterH pour une densité de logs faible en fonction de γ



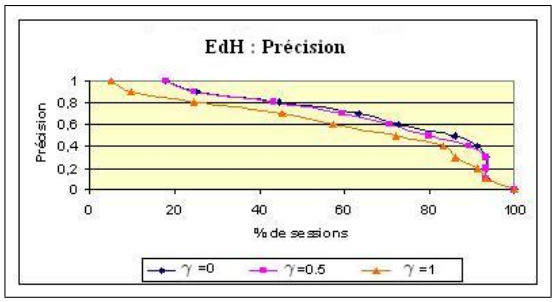
(b) Rappel de ClusterH pour une densité de logs faible en fonction de γ



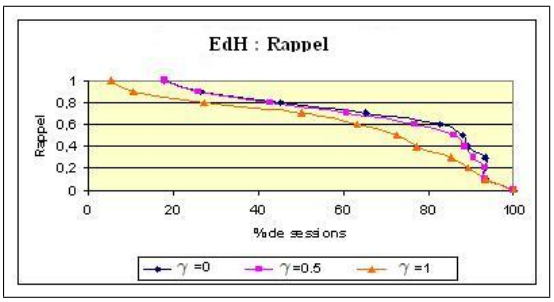
(c) Précision de ClusterSP pour une densité de logs faible en fonction de γ



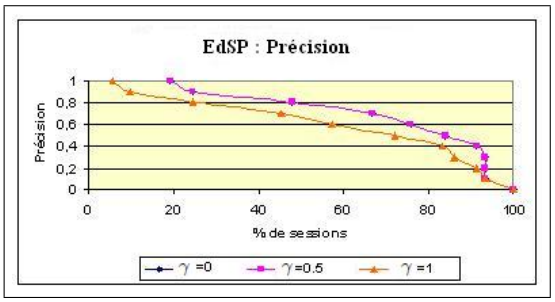
(d) Rappel de ClusterSP pour une densité de logs faible en fonction de γ



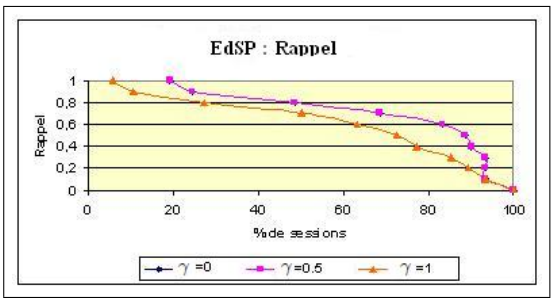
(e) Précision de EdH pour $\alpha = 2$ et une densité de logs faible en fonction de γ



(f) Rappel de EdH pour $\alpha = 2$ et une densité de logs faible en fonction de γ

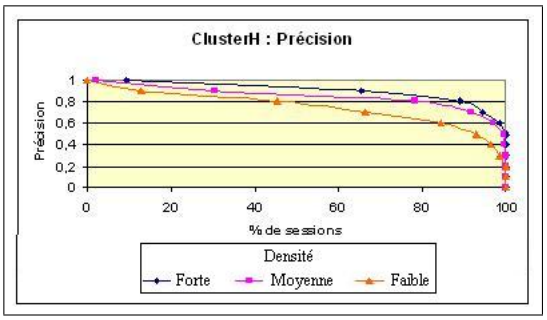


(g) Précision de EdSP pour $\alpha = 3.2$ et une densité de logs faible en fonction de γ

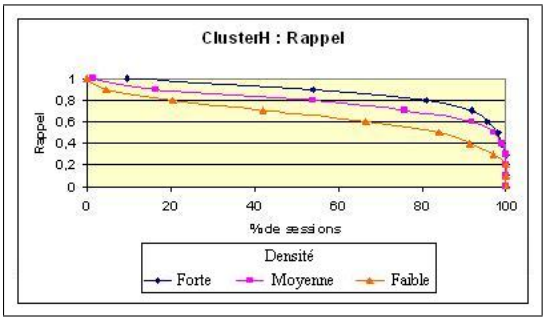


(h) Rappel de EdSP pour $\alpha = 3.2$ et une densité de logs faible en fonction de γ

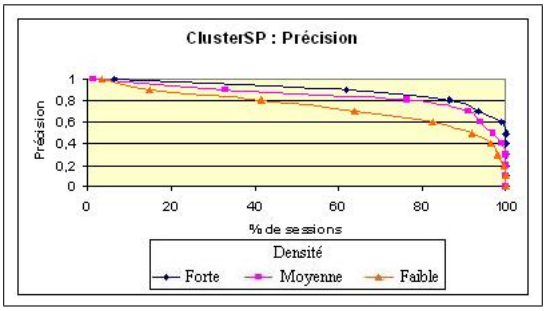
FIGURE 4.5 – Recherche de γ pour ClusterH, ClusterSP, EdH et EdSP



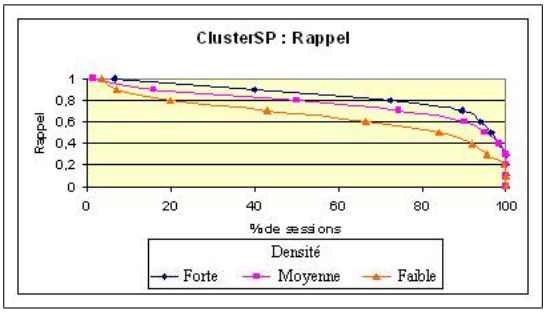
(a) Précision de ClusterH pour $\gamma = 0$ en fonction de la densité des logs



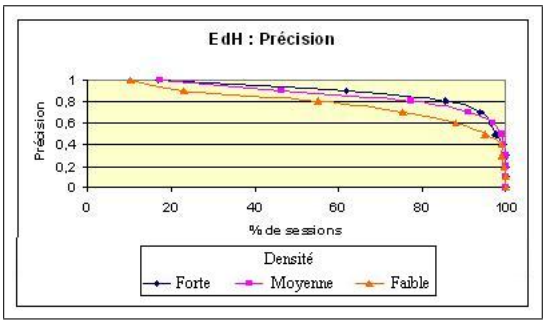
(b) Rappel de ClusterH pour $\gamma = 0$ en fonction de la densité des logs



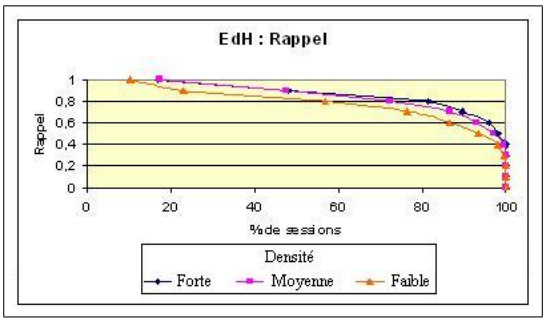
(c) Précision de ClusterSP pour $\gamma = 0$ en fonction de la densité des logs



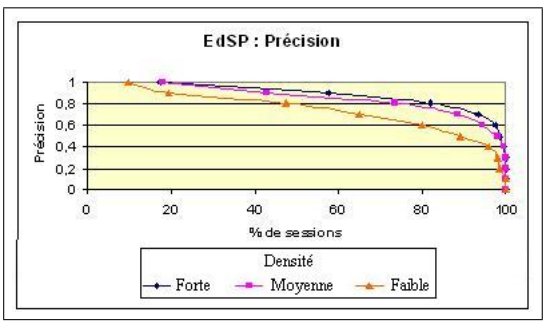
(d) Rappel de ClusterSP pour $\gamma = 0$ en fonction de la densité des logs



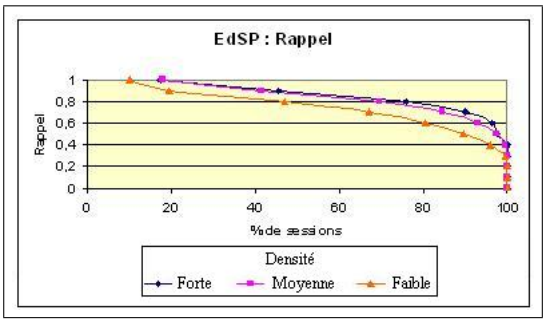
(e) Précision de EdH pour $\gamma = 0$ et $\alpha = 2$ en fonction de la densité des logs



(f) Rappel de EdH pour $\gamma = 0$ et $\alpha = 2$ en fonction de la densité des logs



(g) Précision de EdSP pour $\gamma = 0$ et $\alpha = 3.2$ en fonction de la densité des logs



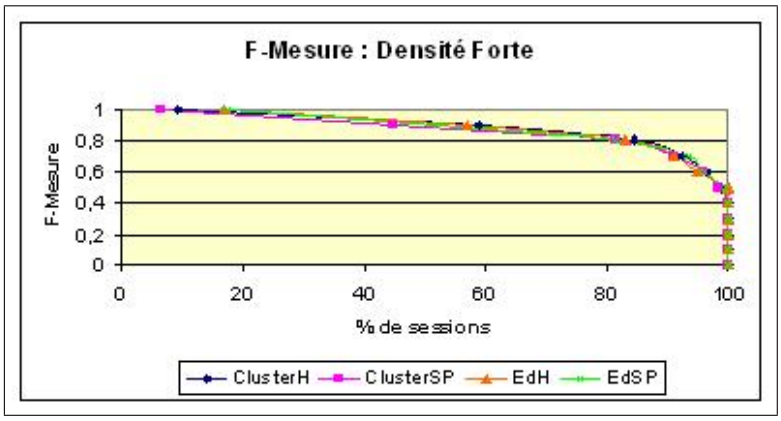
(h) Rappel de EdSP pour $\gamma = 0$ et $\alpha = 3.2$ en fonction de la densité des logs

FIGURE 4.6 – Précision et Rappel de ClusterH, ClusterSP, EdH et EdSP en fonction de la densité des logs

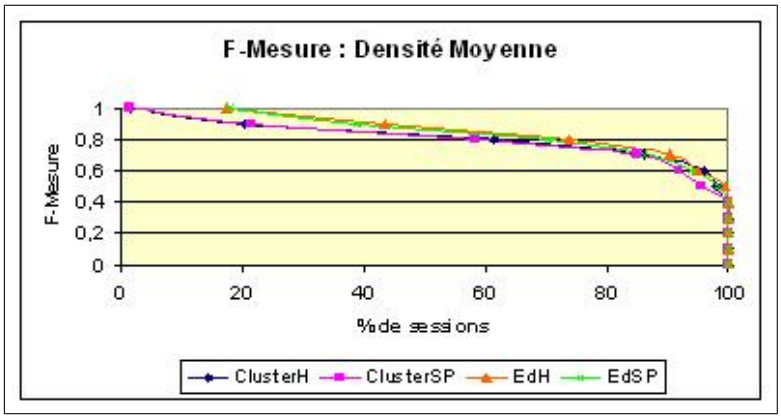
pour plus de 90% des sessions, lorsque les logs sont denses. Notons aussi que, hormis *ClusterH*, le rappel atteint 0.8 pour plus de 75% des sessions pour des logs denses, et pour plus de 45% dans le cas de logs de densité faible. Ainsi, la Figure 4.6 montre que la densité des logs a une influence sur la qualité des recommandations générées (plus les logs sont denses, plus les recommandations sont de bonne qualité) puisque les ensembles de références sont plus proches. Enfin, cette figure montre que *ClusterH* et *ClusterSP* obtiennent de moins bons résultats que *EdH* et *EdSP*, ce qui peut s'expliquer par le fait que *ClusterH* et *ClusterSP*, lors du prétraitement, généralise l'information en regroupant des requêtes qui peuvent être plus ou moins proches. Enfin, les méthodes *ClusterH* et *EdH* utilisant la distance de Hamming obtiennent de meilleurs résultats que les méthodes qui utilisent la distance basée sur le plus court chemin.

Différentes densités de logs

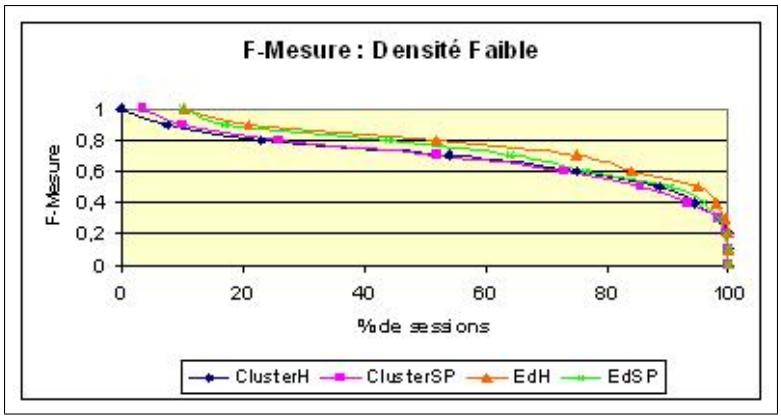
La Figure 4.7 montre l'inverse CFD de la F-mesure obtenue pour les sessions avec les quatre instanciations de recommandations de requêtes *MDX*. Dans un premier temps, nous constatons que les quatre instanciations obtiennent de bons résultats quelle que soit la densité des logs. En effet, la Figure 4.7(a) montrent que pour des logs denses, les quatre méthodes sont aussi performantes les unes que les autres. Cependant, les Figures 4.7(b) et 4.7(c), pour des logs moins denses, montrent que les méthodes utilisant un algorithme de classification, à savoir *ClusterH* et *ClusterSP* ont des performances légèrement plus mauvaises en comparaison aux deux autres méthodes. Il peut être aussi constaté que *EdSP* et *ClusterSP* utilisant la distance basée sur le plus court chemin et celles basées sur la distance de Hamming, *ClusterH* et *EdH* ont des performances assez proches, ce qui peut paraître surprenant dans un premier temps puisque la distance de Hamming pour comparer les références est plus 'grossière' que la distance basée sur le plus court chemin. Cependant, notons que la manière dont sont calculés la précision et le rappel (inspiré par [CEP09]) favorise les méthodes utilisant la distance de Hamming. En effet, si *ClusterH* ou *EdH* recommandent une requête proche (au sens de la distance de Hamming) de la requête attendue, elle aura un bon rappel et une bonne précision. Mais, si *ClusterSP* ou *EdSP* recommandent une requête proche (au sens de la distance basée sur le plus court chemin) de la requête attendue, elle peut avoir un mauvais rappel et/ ou une mauvaise précision. Par conséquent, nous envisageons de remettre en cause la méthode de calcul du rappel et de la précision.



(a) F-mesure de ClusterH, ClusterSP, EdH, EdSP pour une densité forte



(b) F-mesure de ClusterH, ClusterSP, EdH, EdSP pour une densité moyenne



(c) F-mesure de ClusterH, ClusterSP, EdH, EdSP pour une densité faible

FIGURE 4.7 – Comparatif des F-mesures de ClusterH, ClusterSP, EdH et EdSP selon la densité des logs

4.4 Synthèse

Nous avons présenté dans ce chapitre le système développé ainsi que son architecture. A partir d'un log de sessions de requêtes et d'une session courante de requêtes *MDX* sur un cube de données stocké dans le SGBD *MySQL* via le serveur OLAP *Mondrian*, le système propose un ensemble ordonné de requêtes *MDX* pouvant être recommandées à la suite de la session courante.

Ce système nous a permis de valider notre approche de génération de recommandations sur des données synthétiques produites avec notre propre générateur d'ensembles de sessions de requêtes *MDX* sur la base de données *FoodMart* fournie par *Mondrian*. Ce générateur qui simule une analyse pilotée par la découverte a pour paramètres le nombre de sessions dans le log, un nombre maximal de requêtes par session, un nombre de dimensions dans le pool de départ pour jouer sur la densité du log et un seuil de comparaison de cellules.

Nous avons expérimenté notre cadre générique de génération de recommandations et les tests que nous avons effectués ont révélé que notre algorithme de recommandation donne des résultats plutôt satisfaisants.
Les temps de génération d'une recommandation dépassent rarement la seconde sauf pour des fichiers logs de grande taille et de densité faible. Nous avons donc observé que les temps d'exécution augmentent lorsque la taille de la session courante augmente.
Nous avons également observé que les requêtes recommandées générées sont de très bonne qualité puisque la F-mesure atteint 0.8 pour plus de 45% des sessions même pour des fichiers logs de grande taille et de densité faible, cependant, nous avons observé que le pourcentage de sessions atteignant de très bonnes F-mesures augmente lorsque les logs sont denses.

Nos tests ont donc démontré que l'algorithme générique de génération de recommandations que nous avons établi permet de générer des requêtes *MDX* pertinentes en un temps d'exécution très convenable (à l'exception de *EdSP*).

Chapitre 5

Un cadre générique d'organisation des analyses

Sommaire

Dans le domaine de l'exploration OLAP d'entrepôts de données, il y a une nécessité d'organisation, de réutilisation et de partage de requêtes, dans le but de simplifier et d'accélérer le processus de requêtage [DKK05, GMN06]. De manière générale, nous pouvons affirmer que l'information contextuelle durant l'exploration de l'entrepôt de données doit être prise en compte.

Comme présentés Section 2.2 et d'après [KBG+09], les SGBDs fournissent des dispositifs sophistiqués pour aider les utilisateurs en organisant, stockant, contrôlant, et en recherchant des données dans une base de données. Cependant, ils proposent des possibilités limitées pour la réutilisation, l'annotation ou la recherche avancée de requêtes que les utilisateurs lancent sur les données. De plus, l'objectif des systèmes de recommandation comme présenté Section 2.3 et Chapitre 3, est d'aider l'utilisateur lors de son exploration des données, à lancer des requêtes "correctes" et pertinentes. Toujours d'après [KBG+09], cet objectif est proche de celui d'un *système collaboratif de gestion de données*[1]. En fait, un tel système facilite le développement de nouveaux outils d'aide dans le sens où il n'est plus utile de considérer la formulation des requêtes et la gestion des données mais simplement d'explorer l'ensemble des requêtes précédemment lancées.

Dans ce chapitre, afin d'englober notre cadre générique de génération de recommandations (Chapitre 3) dans un cadre plus général, nous proposons donc un cadre d'organisation des analyses ou système de gestion collaborative de requêtes pour les environnements de partage de données qui n'a pas encore été implémenté. Cependant, les requêtes recommandées obtenues grâce à notre cadre générique de génération de recommandations, présenté Chapitre 3, peuvent être intégrées dans un cadre / outil plus général qui est l'organisation et la gestion de requêtes d'analyses. En effet, un tel système permet aux utilisateurs d'effectuer des tâches simples comme naviguer au travers des requêtes des logs ou annoter / décrire les requêtes. De sorte que les utilisateurs peuvent rapidement trouver, éditer et réexécuter des requêtes précédentes. De plus, le système doit permettre des tâches plus sophistiquées comme l'identification de requêtes qui opèrent sur des données spécifiques, qui ont des propriétés particulières (taille de l'ensemble de résultat, temps d'exécution), ou qui produisent des résultats spécifiques. Le système doit également explorer les logs de requêtes et recommander des requêtes. Enfin, le système proposé inclut une interface intuitive et facile à utiliser.

5.1 Exemple intuitif d'organisation des analyses

Dans cette section, nous motivons à travers un exemple comment le cadre que nous proposons peut être utilisé pour exécuter des analyses OLAP. Cet exemple illustre une analyse simple réalisée par un utilisateur dans un environnement multi-utilisateurs en montrant les actions que l'utilisateur doit faire pour organiser, réutiliser, évaluer, observer et partager des requêtes MDX.

Dans ce qui suit, nous utilisons les termes *observer*, *évaluer*, *partager* avec les significations suivantes :

- Observer concerne la possibilité pour l'utilisateur de naviguer parmi des ensembles de requêtes, ainsi que parmi les requêtes dans l'ensemble de requêtes. Observer et naviguer sont interchangeables dans ce qui suit.
- Partager concerne la possibilité pour l'utilisateur d'observer des requêtes définies par d'autres utilisateurs.
- Évaluer concerne la possibilité pour l'utilisateur d'exécuter une requête sélectionnée et de visualiser la réponse à cette requête.

Considérons deux analystes Elsa et Patrick. Supposons que Patrick interroge le cube de données *MesVentes* (présenté Section 2.1.1). Patrick crée un espace de travail pour enregistrer ses requêtes.

1. de l'anglais, *Collaborative Query Management System*.

Dans ce qui suit, cet espace de travail est appelé une analyse. Cette analyse est décrite par le texte *Positionnement de Blois en région Centre pour les ventes de véhicules en 2008*. Sa première requête interroge le cube complet. Puis il lance la requête sur les ventes de véhicules en région Centre. Puis, il se focalise uniquement sur les ventes de véhicules en 2008 dans la ville de Blois. Enfin, il lance une requête pour spécifier qu'il est intéressé par toutes les villes du département du Loir-Et-Cher pour l'année 2008. Durant son analyse, pour se souvenir des différentes requêtes, Patrick assigne une description à chaque requête et organise les requêtes de sorte que la relation entre requêtes affichée à l'écran reflète le raffinement des requêtes. A tout moment, il peut visualiser diverses informations (les *descripteurs*) relatives aux requêtes, comme, par exemple, le résultat de la requête, le code MDX, le nombre de fois que la requête a été lancée, ...

Comme le système gère plus d'un utilisateur, différents ensembles de requêtes définis par des utilisateurs différents peuvent être échangés entre les utilisateurs. Elsa peut donc observer les analyses et les requêtes de Patrick. Pendant ce temps, le système peut compter combien de fois une requête est évaluée et / ou observée, ces informations apparaissant comme des descripteurs associés à la requête.

Supposons maintenant que Elsa interroge l'ensemble de toutes les analyses existantes pour trouver quelles sont celles qui traitent de Véhicules, Tours ou Centre. Le système retourne les trois analyses : *Positionnement de Blois en région Centre pour les ventes de véhicules en 2008* (l'analyse de Patrick), *Ventes de véhicules immatriculés 'AAA' en 2008 à Tours* et *Ventes de véhicules rouges et bleus en région Centre* (correspondants aux sessions s_1 et s_2 de l'Annexe A). Elsa choisit d'observer d'abord l'analyse *Positionnement de Blois en région Centre pour les ventes de véhicules en 2008*. Elle sélectionne la requête décrivant le cube complet *Tous les véhicules*, lance la requête, visualise son résultat et veut copier cette requête dans son analyse. Elle crée une nouvelle analyse contenant la copie de la requête et assigne sa propre description à la requête : *Ventes de tous véhicules*. De sorte que le système ajoute un lien (appelé *pointeur*) de la requête *Ventes de tous véhicules* de l'analyse d'Elsa vers la requête *Tous les véhicules* de l'analyse de Patrick. Ce lien rappelle que la requête contenue dans l'analyse d'Elsa provient de la requête de Patrick. Ce pointeur peut être utilisé comme une recommandation que le système peut proposer aux utilisateurs qui observent cette requête particulière de l'analyse d'Elsa.

Pour poursuivre son analyse, Elsa interroge les analyses *Ventes de véhicules rouges et bleus en région Centre* et *Ventes de véhicules immatriculés 'AAA' en 2008 à Tours*, observe les requêtes de ces analyses mais ne trouve pas de requête intéressante. Puis elle revient à sa propre analyse et lance sa propre requête sur les ventes de véhicules en Indre-Et-Loire puis interroge à nouveau *Ventes de véhicules immatriculés 'AAA' en 2008 à Tours* et copie la requête *Rouge, Bleu, Tours* puis se voit recommander par *Reco*OLAP (présenté Chapitre 3), la requête *Rouge, Bleu, Centre* de l'analyse *Ventes de véhicules rouges et bleus en région Centre* qu'elle décide d'évaluer : le système ajoute un pointeur de la requête d'Elsa vers la requête *Rouge, Bleu, Centre* de l'analyse *Ventes de véhicules rouges et bleus en région Centre* pour indiquer que la seconde à été recommandée à la suite de la première. Toutes les requêtes de l'analyse sont décrites par *Ventes de véhicules rouges et bleus à Tours*, ainsi, l'analyse est associée à ce descripteur.

La Figure 5.1 illustre l'état courant de l'analyse d'Elsa, en décrivant ce que pourrait être l'interface utilisateur du système qui implémente les opérations pour organiser, partager et observer des requêtes MDX. Notons que l'organisation des requêtes diffère d'une analyse à l'autre : dans *Ventes de véhicules en 2008 à Blois*, l'analyse de Patrick, l'organisation des requêtes reflète le raffinement de requêtes, tandis que dans *Ventes de véhicules rouges et bleus à Tours*, l'analyse d'Elsa, l'organisation des requêtes reflète l'ordre dans lequel les requêtes ont été importées ou créées, c'est-à-dire leur séquencement.

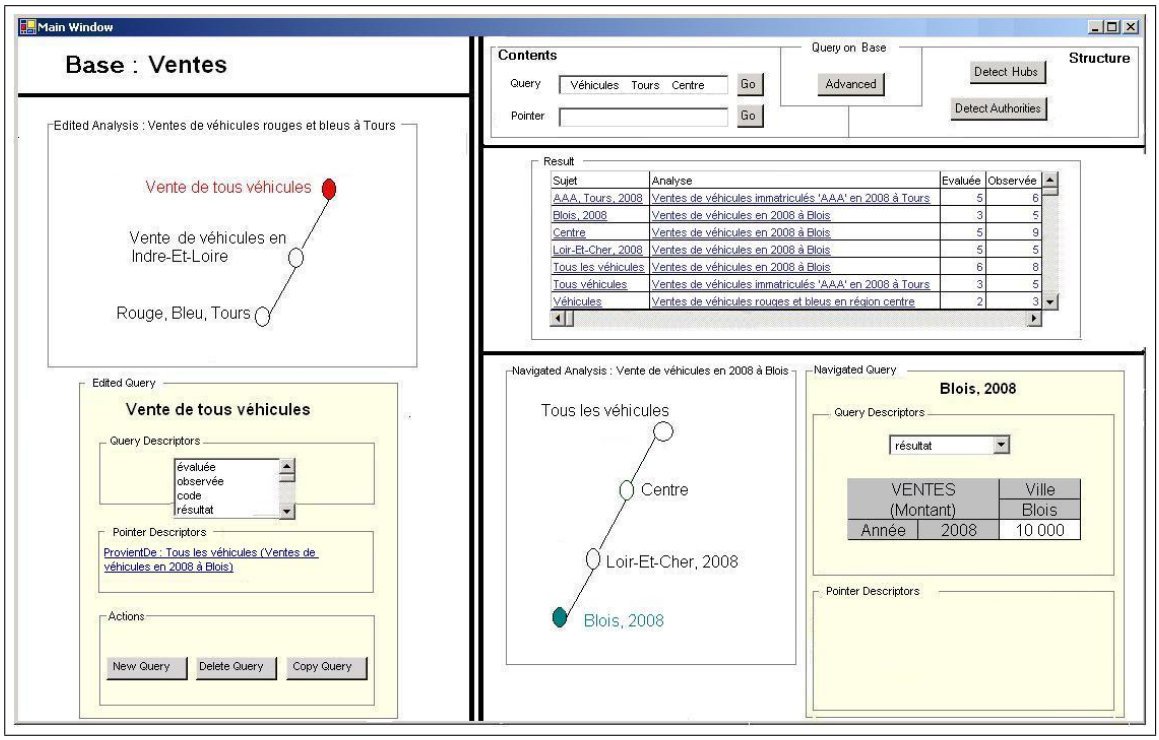


FIGURE 5.1 – Exemple d'analyse (Elsa)

5.2 Le cadre générique d'organisation des analyses

Dans [CGL+07], nous proposons un modèle qui répond à ces besoins, en adaptant le modèle proposé par [TACS98] dans le contexte du travail collaboratif [TACS02, AST03, AS04].

5.2.1 Les contextes et le modèle de [TACS98]

Afin d'améliorer l'organisation des données, l'information contextuelle pendant l'exploration des données du cube doit être prise en considération. La notion de contexte est une préoccupation importante en psychologie cognitive, linguistique, et en informatique. Les contextes ont été étudiés dans de nombreux secteurs de l'informatique, comme l'intelligence artificielle [McC96], les bases de données [KS96], et la représentation de connaissance [TACS98, TACS99]. Cependant, les formalisations sont très diverses et atteignent différents objectifs. Nous nous sommes essentiellement intéressés aux travaux de [TACS98, TACS99, TACS02, AST03, AS04] qui ont proposé un modèle pour utiliser cette information contextuelle.

5.2.1.1 Présentation informelle du modèle de [TACS98]

Le modèle de [TACS98] permet de représenter des contextes dans des bases d'informations.

D'après [TACS98], un contexte est un ensemble identifiable d'objets, chaque objet étant associé à un ensemble de descriptions et pouvant être relié à un autre objet via une relation entre objets.

Puis, une base d'informations est un ensemble de contextes pouvant être, eux aussi, reliés les uns aux autres par des relations entre contextes.

Exemple 5.2.1 *Considérons le log de sessions de requêtes présenté Annexe A, constitué de requêtes réparties dans les sessions s_1, s_2 et s_3. Le log peut être vu comme une base d'informations contenant trois sessions, où chaque session peut être vue comme un contexte qui décrit un groupe de requêtes du point de vue de l'utilisateur qui a lancé la session sur le cube de données où :*
$Log = \{\{s_1, s_2, s_3\}$, Nom :"Log annexe" $\}$

1. *$s_1 = \{\{q_{1_{s_1}}, q^2_{2_{s_1}}, q^2_{3_{s_1}}\}$, Nom : "Ventes de véhicules rouges et bleus en région Centre", AppartientA : Log$\}$*
   (a) *$q_{1_{s_1}} = \{$Nom : "Tous les véhicules", CodeMDX : "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].Members} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q^2_{2_{s_1}}$, AppartientA : s_1, CopiéeVers : $q_{1_{s_2}}\}$*
   (b) *$q^2_{2_{s_1}} = \{$Nom : "Ventes de véhicules en région Centre", CodeMDX : "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].[Centre]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : q^2_3, AppartientA : s_1, CopiéeVers : $q^2_{2_{s_3}}\}$*
   (c) *$q^2_{3_{s_1}} = \{$Nom :"Ventes de véhicules rouges et bleus en région Centre", CodeMDX : "SELECT {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS, {[Géographie].[AllG].[Centre]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", AppartientA : s_1, CopiéeVers : $q^2_{3_{s_2}}\}$*
2. *$s_2 = \{\{q_{1_{s_2}}, q^2_{3_{s_2}}, q_{3_{s_2}}, q_{4_{s_2}}\}$, Nom : "Ventes de véhicules immatriculés 'AAA' en 2008 à Tours", AppartientA : Log$\}$*
   (a) *$q_{1_{s_2}} = \{$Nom : "Tous les véhicules", CodeMDX : "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].Members} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q^2_{3_{s_2}}$, AppartientA : s_2, CopiéeVers : $q_{1_{s_3}}$, ProvientDe : $q_{1_{s_1}}\}$*
   (b) *$q^2_{3_{s_2}} = \{$Nom : "Ventes de véhicules rouges et bleus en région Centre", CodeMDX : "SELECT {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS, {[Géographie].[AllG].[Centre]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q_{3_{s_2}}$, AppartientA : s_2, ProvientDe : $q^2_{3_{s_1}}\}$*
   (c) *$q_{3_{s_2}} = \{$Nom : "Ventes de véhicules rouges et bleus à Tours", CodeMDX : "SELECT {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS, {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q_{4_{s_2}}$, AppartientA : $s_2\}$*
   (d) *$q_{4_{s_2}} = \{$Nom : "Ventes de véhicules immatriculés 'AAA' en 2008 à Tours", CodeMDX : "SELECT Crossjoin({[Véhicules].[AllV].[Bleu].[AAA]}, {[Temps].[AllT].[2008]}) ON COLUMNS, {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", AppartientA : $s_2\}$*
3. *$s_3 = \{\{q_{1_{s_3}}, q^2_{2_{s_3}}, q_{5_{s_3}}, q_{6_{s_3}}\}$, Nom : "Ventes de véhicules en 2008 à Blois", AppartientA : Log$\}$*
   (a) *$q_{1_{s_3}} = \{$Nom : "Tous les véhicules", CodeMDX : "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].Members} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q^2_{2_{s_3}}$, AppartientA : s_3, ProvientDe : $q_{1_{s_2}}\}$*
   (b) *$q^2_{2_{s_3}} = \{$Nom : "Ventes de véhicules en région Centre", CodeMDX : "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].[Centre]} ON ROWS FROM*

[Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q_{5_{s_3}}$, AppartientA : s_3, ProvientDe : $q^2_{2_{s_1}}$}

(c) *$q_{5_{s_3}}$ = {Nom : "Ventes de véhicules en Loir-Et-Cher en 2008", CodeMDX : "SELECT {[Géographie].[AllG].[Centre].[LoirEtCher].Children} ON COLUMNS, {[Temps].[AllT].[2008]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", ReqSuivante : $q_{6_{s_3}}$, AppartientA : s_3}*

(d) *$q_{6_{s_3}}$ = {Nom : "Ventes de véhicules en 2008 à Blois", CodeMDX : "SELECT {[Géographie].[AllG].[Centre].[LoirEtCher].[Blois]} ON COLUMNS, {[Temps].[AllT].[2008]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}", AppartientA : s_3}*

Notons que le descripteur Nom correspond à une description textuelle, codeMDX à la requête écrite en MDX, ReqSuivante indique que la requête est reliée à une autre requête via une relation de séquencement, CopiéeVers indique que la requête source de ce lien a été copiée par la requête cible, ProvientDe indique que la requête source de ce lien est une copie de la requête cible et AppartientA indique l'ensemble d'appartenance de la requête ou de la session.

Ainsi, une base d'informations peut être vue comme un graphe où les nœuds sont les objets et les arcs sont les liens entre les objets.

Exemple 5.2.2 *Reprenons l'exemple précédent où nous considérons le log comme une base d'informations, les sessions s_1, s_2 et s_3 comme des contextes de la base d'informations et les requêtes comme des objets des contextes. Le graphe correspondant est présenté Figure 5.2.*

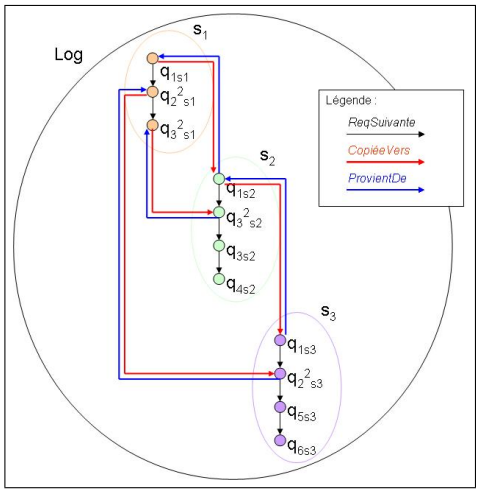


FIGURE 5.2 – Représentation contextuelle du log de sessions de requêtes présenté Annexe A

Ce modèle permet donc de représenter des contextes dans des bases d'informations. Il est constitué d'un ensemble d'opérations pour créer, mettre à jour, combiner (union, intersection, différence de contextes), et comparer des contextes. Les opérations fondamentales définies par les auteurs sont :

- Observation :
  - observer un ou plusieurs objets issu(s) d'un ou plusieurs contextes,
  - désigner le contexte courant.
- Modification :

  - créer un nouveau contexte,
  - insérer / supprimer un objet dans un contexte,
  - insérer / supprimer des descriptions d'objets.
- Copie :
  - copier / cloner un contexte,
- Combinaison :
  - faire l'union / l'intersection / la différence de deux contextes,

5.2.2 Présentation informelle de notre modèle

Comme pressenti Section 5.1, nous considérons que l'utilisateur définit et enregistre des requêtes MDX dans ce que nous appelons une *analyse*. Par analogie avec le modèle de [TACS98], les requêtes sont les objets et les analyses sont les contextes. Ainsi, dans une analyse, l'utilisateur peut organiser les requêtes de sorte qu'elles soient facilement parcourues dans l'avenir. Cette organisation peut refléter, par exemple, le séquencement classique des requêtes ou bien un ordre d'importance particulier, pertinent pour l'utilisateur. Dans un environnement multi-utilisateurs, les analyses peuvent être partagées entre les utilisateurs. L'ensemble des analyses peut être parcouru ou requêté. De plus, des requêtes MDX d'une analyse donnée peuvent être importées dans une autre analyse afin d'enrichir l'analyse courante de l'utilisateur.

Ainsi, le modèle de [TACS98], présenté de manière informelle dans la section précédente, peut être adapté à notre environnement. De sorte que, comme pressenti dans l'Exemple 5.2.1, les objets, contextes et bases d'informations peuvent être mis en parallèle avec nos requêtes, sessions et logs.

Définition 5.2.1 *(Descripteur)*
Un descripteur est un n-uplet $\langle attribut, valeur \rangle$ *qui permet de décrire une requête ou un pointeur.*

Requête :
Une requête est une requête MDX comme définie Définition 2.1.6.

Pointeur :
Un pointeur relie deux requêtes q_1 et q_2 telles que q_1 est la source et q_2 la cible du pointeur.

Analyse :
Une analyse est un ensemble de requêtes reliées les unes aux autres par des pointeurs.
Notons qu'une session de requêtes comme définie Définition 2.1.7 est un cas particulier d'analyse où les requêtes sont reliées les unes aux autres par des pointeurs représentant le séquencement des requêtes.

Base d'analyses
Une base d'analyses est un ensemble d'analyses.
Notons qu'un log de sessions de requêtes comme définie Définition 2.1.8 est un cas particulier de base d'analyses.

Intuitivement, notre modèle se décompose en deux niveaux :

- le niveau des données qui modélise les données qui peuvent être éditées, enregistrées, requêtées, ...
- le niveau du système qui modélise ce qui est présenté à l'utilisateur.

Le niveau des données

Les données peuvent être de trois types :

- une requête sur un cube de données comme définie Définition 2.1.6. Elle est identifiée par un identifiant de requête et est décrite par un ensemble de descripteurs.
  Exemple 5.2.3 *Par exemple, une requête peut être décrite par son code MDX et ainsi être associée au descripteur suivant :* $\langle$*code, "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].Members} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}"*$\rangle$.
  Elle peut aussi être décrite par :
  - $\langle$*résultat,* $CT_1\rangle$ *où* CT_1 *est le tableau croisé utilisé pour visualiser le résultat de la requête,*
  - $\langle$*évaluée, 7*$\rangle$ *qui indique que la requête a été évaluée sept fois.*
- une analyse dans laquelle l'utilisateur enregistre et organise les requêtes de sorte qu'elles soient facile à observer lors d'une prochaine utilisation,
- un pointeur qui est utilisé pour établir un lien entre requêtes. Il est associé à un ensemble de descripteurs. Les pointeurs sont utilisés pour décrire l'organisation des requêtes dans une base d'analyses. Ils peuvent aussi être utilisés pour indiquer la redondance de requêtes.
  Exemple 5.2.4 *Par exemple, considérons les requêtes* q_1 *associée au descripteur* $\langle$*code, "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].Members} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}"*$\rangle$ *et* q_2 *associée au descripteur* $\langle$*code, "SELECT {[Véhicules].[AllV].Members} ON COLUMNS, {[Géographie].[AllG].[Centre]} ON ROWS FROM [Ventes] WHERE {[NomMesures].[Montant]}"*$\rangle$*, elles peuvent être reliées par un pointeur dont le descripteur est* $\langle$*raffine,"Région"*$\rangle$ *pour indiquer que* q_2 *raffine* q_1 *et donc que* q_1 *est inclus dans* q_2 *au sens classique de l'inclusion de requêtes.*

Notons que dans notre modèle, les requêtes, les analyses et les pointeurs forment un graphe où les requêtes sont les nœuds et les pointeurs sont les arcs.

Le niveau du système

Ce niveau modélise ce qui est présenté à l'utilisateur, comme illustré Figure 5.1 :

- le nom de la base d'analyses dont l'instance est l'ensemble de toutes les analyses, requêtes et pointeurs qui peuvent être observés. Ce nom est indiqué dans la zone en haut à gauche de la Figure 5.1.
- l'interrogation de la base d'analyses qui permet de sélectionner les parties pertinentes de la base d'analyses. Cette interrogation peut se faire par l'intermédiaire de mots-clés en utilisant la zone en haut à droite de la Figure 5.1.
- la requête courante observée ainsi que son analyse est affichée dans la zone en bas à droite de la Figure 5.1.
- la requête courante éditée ainsi que son analyse est affichée dans la zone en bas à gauche de la Figure 5.1.

Interagir avec le système consiste à changer ce qui est affiché à l'écran. Cela peut être fait en :
- changeant la base d'analyses, soit en interrogeant à nouveau la base d'analyses, soit en visualisant une autre analyse parmi celles obtenues en réponse à l'interrogation de la base d'analyses,
- éditant la base d'analyses, c'est-à-dire en définissant, en modifiant ou en supprimant des analyses, des requêtes ou des pointeurs.

Par la suite, nous allons donc présenter :
- Un modèle général pour l'organisation de requêtes MDX, appelé *la Base d'analyses*, qui permet de partager et de réutiliser facilement des requêtes, et qui est composé de deux niveaux : le niveau des données présenté Section 5.2.3 et le niveau du système présenté Section 5.2.4,
- Les langages de définition, manipulation et parcours de cette base,
- L'exploitation de la structure de la base d'analyses pour fournir des recommandations utiles au parcours de l'utilisateur sera présentée, quant à elle, Section 5.3.

5.2.3 Le niveau des données

5.2.3.1 Le modèle de données

Pour des raisons de simplicité, notre modèle de données est décrit en utilisant le modèle relationnel [AHV95].

Notation 5.2.1 *(Dom) Soit Dom un ensemble infini de constantes dénombrables. Une constante spéciale NULL est utilisée pour indiquer le fait qu'aucune requête n'est observée. De plus, nous supposons que Dom est ordonné.*

5.2.3.1.1 Les relations

Nous considérons les trois relations suivantes (dans ce qui suit, q_{id}, a_{id},att, val, q_{id_1},$q_{id_2} \in Dom$) :
- requêtes : est une relation ternaire. Le n-uplet $requêtes(q_{id}, att, val)$ associe l'identifiant de requête q_{id} à un descripteur qui a pour attribut att et valeur val.
- analyses : est une relation binaire. Le n-uplet $analyses(a_{id}, q_{id})$ associe l'identifiant de requête q_{id} à l'identifiant d'analyse a_{id}.
- pointeurs : est une relation quaternaire. Le n-uplet $pointeurs(q_{id_1}, q_{id_2}, att, val)$ associe l'identifiant de requête q_{id_1} à l'identifiant de requête q_{id_2} avec un descripteur ayant pour attribut att et valeur val.

Généralement, pour chaque nom de relation R, une instance de relation sur R est un ensemble fini de tuples sur R. Notons que, dans notre modèle, requêtes et pointeurs forment un graphe où les requêtes sont les nœuds et les pointeurs les arcs.

5.2.3.1.2 La base d'analyses

Schéma et instance de la base d'analyses

Le schéma d'une base d'analyses consiste en un nom de base d'analyses et en l'ensemble des noms de relation $\{analyses, requêtes, pointeurs\}$. Une instance de base d'analyses est un ensemble fini de tuples qui est l'union des instances des relations sur R, pour $R \in \{analyses,\ requêtes,\ pointeurs\}$.

Instance bien formée de base d'analyses

Une instance de base d'analyses I est bien formée si chaque requête appartient à une et une seule analyse. Cela signifie que nous ne pouvons pas avoir, par exemple, $requêtes(q_1, att_1, val_1)$, $analyses(a_1, q_1)$ et $analyses(a_2, q_1)$. Le partage de requêtes entre analyses ne peut être fait qu'en dupliquant les requêtes et en connectant les copies grâce à des pointeurs. Cela permet de garder une trace de la duplication, une information qui peut être requêtée par la suite.

Cette condition est exprimée formellement ci-dessous :

- Si $requêtes(q_{id}, x, y) \in I$ alors il existe un seul $a_{id} \in Dom$ tel que $analyses(a_{id}, q_{id}) \in I$.
- Si $pointeurs(q_{id_1}, q_{id_2}, x, y) \in I$ alors il existe $a_{id_1}, a_{id_2} \in Dom$ tel que $analyses(a_{id_1}, q_{id_1}) \in I$ et $analyses(a_{id_2}, q_{id_2}) \in I$ et (q_{id_1}, q_{id_2}) est l'identifiant du pointeur.

5.2.3.2 Le langage de manipulation

Le langage que nous utilisons pour décrire la manipulation de la base d'analyses est Datalog$^\neg$ [AHV95] car nous avons besoin d'exprimer la récursivité (pour calculer la fermeture transitive du graphe des objets) et la division relationnelle (cf Section 5.3 pour des exemples de requêtes). Ce langage est utilisé pour calculer des sous-ensembles de la base d'analyses que l'utilisateur pourrait parcourir dans l'avenir.

Dans ce qui suit, soit une instance bien formée de la base d'analyses I, la sémantique d'un programme Datalog$^\neg$ P sur I est la sémantique stratifiée classique notée $P(I)$.

Programme bien formé

Un programme P sur une instance bien formée de base d'analyses I, est bien formé si les prédicats suivants appartiennent à $P(I)$:

- $analyses\_u$: une relation binaire qui permet d'identifier les analyses de I pertinentes pour l'utilisateur,
- $requêtes\_u$: une relation ternaire utilisée pour identifier les requêtes de I pertinentes pour l'utilisateur,
- $pointeurs\_u$: une relation quaternaire qui permet d'identifier les pointeurs de I pertinents pour l'utilisateur.

Ces prédicats sont utilisés pour identifier la partie pertinente de I que l'utilisateur souhaite explorer. Cela signifie que, si I est une instance bien formée de base d'analyses et P un programme bien formé, la base résultat est la réponse à P qui contient les relations $analyses\_u$, $requêtes\_u$ et $pointeurs\_u$. De plus, $\forall x, y, z \in Dom$, ce qui suit est nécessaire :

$$requêtes\_u(x, y, z) \in P(I) \Rightarrow requêtes(x, y, z) \in I$$
$$analyses\_u(x, y) \in P(I) \Rightarrow analyses(x, y) \in I$$
$$pointeurs\_u(x, y, z, t) \in P(I) \Rightarrow pointeurs(x, y, z, t) \in I.$$

Exemple 5.2.5 *Nous présentons deux exemples de programmes bien formés :*

1. *Le programme suivant cherche les requêtes dont le sujet traite de Véhicules ou de Tours mais pas de Blois :*

$$\begin{aligned}
requêtes_u(x, \text{"}sujet\text{"}, z) &\leftarrow requêtes(x, \text{"}sujet\text{"}, z), substring^2(z, \text{"}Véhicules\text{"}), \\
&\quad \neg substring(z, \text{"}Blois\text{"}) \\
requêtes_u(x, \text{"}sujet\text{"}, z) &\leftarrow requêtes(x, \text{"}sujet\text{"}, z), substring(z, \text{"}Tours\text{"}), \\
&\quad \neg substring(z, \text{"}Blois\text{"}) \\
requêtes_u(x, s, t) &\leftarrow requêtes(x, s, t), requêtes_u(x, \text{"}sujet\text{"}, z) \\
analyses_u(a, x) &\leftarrow requêtes_u(x, s, t), analyses(a, x) \\
pointeurs_u(x, x_1, y_1, z_1) &\leftarrow requêtes_u(x, s, t), pointeurs(x, x_1, y_1, z_1)
\end{aligned}$$

2. *Le programme suivant recherche les parties de la base d'analyses qui sont accessibles à partir d'une requête particulière* q_1 *:*

$$
\begin{array}{lcl}
ans(\text{"}q_1\text{"}, y) & \leftarrow & pointeurs(\text{"}q_1\text{"}, y, \_, \_) \\
ans(x, z) & \leftarrow & ans(x, y), pointeurs(y, z, \_, \_) \\
analyses\_u(a, x) & \leftarrow & ans(\text{"}q_1\text{"}, x), analyses(a, x) \\
requêtes\_u(q, att, val) & \leftarrow & ans(\text{"}q_1\text{"}, q), requêtes(q, att, val) \\
pointeurs\_u(q, q_2, att, val) & \leftarrow & ans(\text{"}q_1\text{"}, q), ans(\text{"}q_1\text{"}, q_2), pointeurs(q, q_2, att, val)
\end{array}
$$

5.2.4 Le niveau du système

5.2.4.1 Le modèle du système

Le système est une paire $\langle B, E\rangle$ où :
- B (pour *Base*) est une instance bien formée de base d'analyses,
- E (pour *Etat*) est un 3-uplet $\langle P, q_{nav}, q_{ed}\rangle$ qui représente ce qui est affiché à l'utilisateur :
  - P est un programme (exprimé comme un programme Datalog$^\neg$ bien formé) sur B,
  - $q_{nav} \in Dom$ est l'identifiant de la requête parcourue par l'utilisateur, appelée la requête naviguée,
  - $q_{ed} \in Dom$ est l'identifiant de la requête éditée par l'utilisateur, appelée la requête éditée.

Exemple 5.2.6 *Considérons le système dont l'interface est représentée Figure 5.1. L'interface est divisée en cinq zones :*
- *Puisque l'instance de la base d'analyses est souvent trop grande pour être affichée entièrement, seulement le nom de la base d'analyses est affiché en haut à gauche.*
- *La zone en bas à gauche est la zone d'édition. Dans cette zone, la requête éditée* q_{ed} *et son analyse sont affichées.*
- *La zone en haut à droite permet d'interroger la Base en définissant le programme* P*. Dans cette zone, il y a trois parties : la partie gauche permet de requêter le contenu de la base, c'est-à-dire de requêter les descripteurs de requêtes ou les descripteurs de pointeurs, la partie droite permet de requêter la structure avec certains programmes prédéfinis, et la partie centrale permet à l'utilisateur d'entrer directement un programme grâce à une interface ad-hoc (non détaillée).*
- *La zone centrale droite affiche le résultat du programme sur* B*, c'est-à-dire* $P(B)$ *: la liste des requêtes des analyses (avec une ligne par requête), son sujet, l'analyse à laquelle elle appartient et le nolbre de fois que la requête a été évaluée et observée.*
- *La zone en bas à droite est la zone de navigation. Dans cette zone, la requête naviguée* q_{nav} *et son analyse sont affichées.*

5.2.4.2 Le langage du système

Une opération sur le système, pour changer ce qui est affiché, peut être soit :
- Une opération de navigation, pour parcourir/observer les éléments de la base d'analyses. Les opérations de navigation modifient seulement l'état du système.
- Une opération d'édition, pour éditer les éléments de la base d'analyses. Les opérations d'édition modifient la base et peuvent éventuellement modifier l'état du système.

2. Substring est un prédicat intégré ayant la signification standard.

5.2.4.2.1 Les opérations de navigation

La navigation peut être faite en changeant la requête naviguée ou en changeant le programme sur la base d'analyses. Ces opérations permettent à l'utilisateur de parcourir les analyses ainsi que leur contenu.

Il existe deux méthodes permettant de changer la requête naviguée :

- *gotoQuery* qui accède à une requête via son identifiant.
  Exemple 5.2.7 *Considérons le système présenté Figure 5.1, l'utilisateur a interrogé B et a obtenu $P(B)$ dont les descripteurs de requêtes sont affichés dans la zone centrale droite. Il peut cliquer sur une requête particulière de cette zone. Cette action est associée à l'opération gotoQuery. De la même manière, lorsque l'utilisateur est en train de visualiser une requête particulière d'une analyse dans la zone en bas à droite, il peut cliquer sur une autre requête de cette zone. Cette action est aussi associée à l'opération gotoQuery.*

- *nextQuery* qui utilise les pointeurs ayant q_{nav} comme requête source.
  Exemple 5.2.8 *Considérons le système présenté Figure 5.1, l'utilisateur est en train de regarder les descripteurs de pointeurs de la requête naviguée dans la zone en bas à droite. Il peut accéder aux requêtes pointées par cette requête. Cette action est associée à l'opération nextQuery.*

Dans les exemples qui suivent, nous considérons le système S_1 présenté Figure 5.1 avec $Etat = \langle P, 4, 10\rangle$ où P renvoie les requêtes qui traitent de *Véhicules* ou $Tours$ ou $Centre$:

$$
\begin{array}{lcl}
requêtes\_u(x,"sujet",z) & \leftarrow & requêtes(x,"sujet",z), substring(z,"Véhicules") \\
requêtes\_u(x,"sujet",z) & \leftarrow & requêtes(x,"sujet",z), substring(z,"Tours") \\
requêtes\_u(x,"sujet",z) & \leftarrow & requêtes(x,"sujet",z), substring(z,"Centre") \\
requêtes\_u(x,s,t) & \leftarrow & requêtes(x,s,t), requêtes\_a(x,"sujet",z) \\
analyses\_u(c,x) & \leftarrow & requêtes\_u(x,s,t), analyses(c,x) \\
pointeurs\_u(x,x_1,y_1,z_1) & \leftarrow & requêtes\_u(x,s,t), pointeurs(x,x_1,y_1,z_1)
\end{array}
$$

Dans la suite, nous présentons les opérations nécessaires pour naviguer à travers les analyses et les requêtes.

gotoQuery

Pour une requête donnée que l'utilisateur est en train de regarder sur l'interface, cette opération permet de changer la requête naviguée, la requête naviguée cible étant le paramètre.

Définition 5.2.2 *Soit $Système = \langle Base, Etat\rangle$, $Etat = \langle P, q_{nav}, q_{ed}\rangle$, q_1 un identifiant de requête tel qu'il existe att_1, val_1 tel que $requêtes(q_1, att_1, val_1) \in P(Base)$.*
$gotoQuery(Système, q_1) = \langle Base, Etat'\rangle$ avec $Etat' = \langle P, q_1, q_{ed}\rangle$.

Exemple 5.2.9 *Considérons le système S_1 présenté Figure 5.1 avec $Etat = \langle P, 4, 10\rangle$. L'utilisateur décide de voir la requête décrite par $\langle sujet, "Centre"\rangle$ dans l'analyse qu'il est en train de regarder en cliquant sur son descripteur. Cette action est associée à l'opération : $gotoQuery(S_1, 2)$.*
Il obtient le nouveau système dont $Etat' = \langle P, 2, 10\rangle$ présenté Figure 5.3.

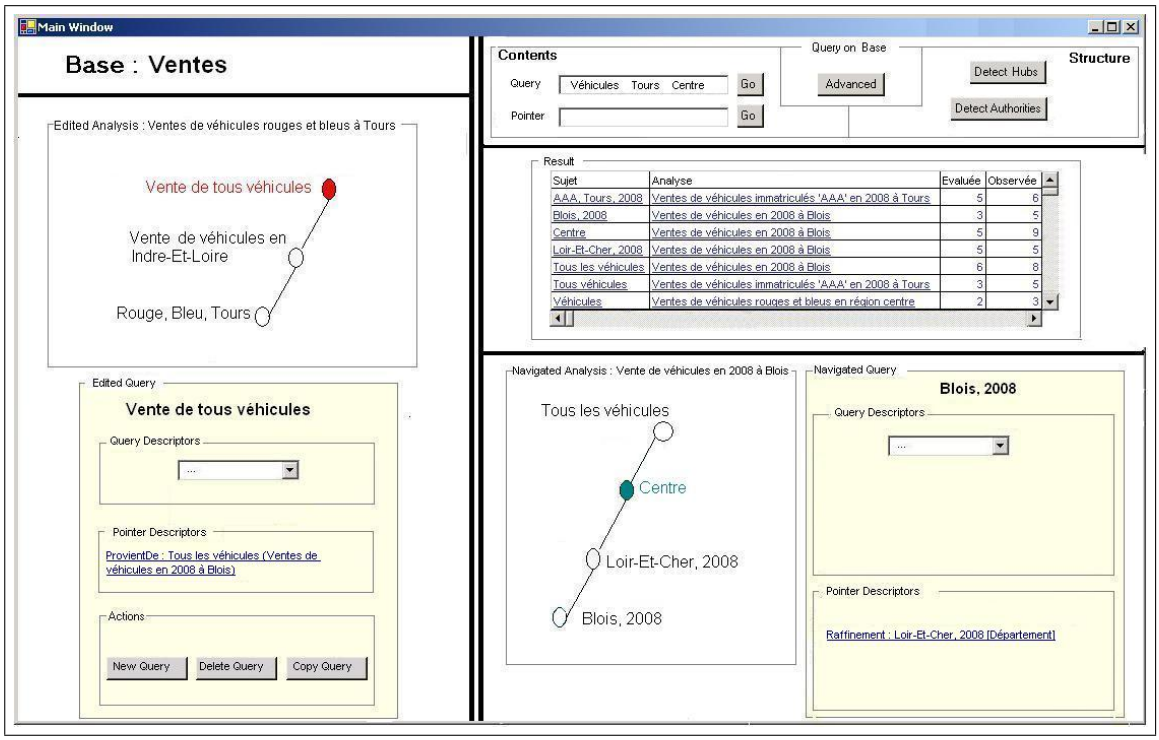


FIGURE 5.3 – Effet de l'opération de navigation gotoQuery

nextQuery

Pour un pointeur donné entre la requête naviguée et une autre requête que l'utilisateur ne peut pas voir sur l'interface mais dont les descripteurs de pointeurs sont affichés, cette opération permet de changer de requête naviguée en se déplaçant dans le graphe de requêtes.

Définition 5.2.3 *Soit* $Système = \langle Base, Etat\rangle$ *et* $Etat = \langle P, q_{nav}, q_{ed}\rangle$ *tel qu'il existe un n-uplet* $pointeurs(q_{nav}, q_1, att, val) \in Base$ *et* $requêtes(q_1, att_1, val_1) \in P(Base)$.
$nextQuery(Système, att, val) = \langle Base, Etat'\rangle$ *avec* $Etat' = \langle P, q_1, q_{ed}\rangle$.

newProg

Cette opération permet de lancer un nouveau programme et ainsi de changer l'ensemble des analyses dans lequel l'utilisateur est en train de naviguer.

Définition 5.2.4 *Soit Système* $= \langle Base, Etat\rangle$, $Etat = \langle P, q_{nav}, q_{ed}\rangle$ *et* P_1 *un programme Datalog*$^\neg$ *bien formé.*
$newProg(Système, P_1) = \langle Base, Etat'\rangle$ *où* $Etat' = \langle P_1, NULL, q_{ed}\rangle$.

Exemple 5.2.10 *Considérons le système* S_1 *où* $Etat = \langle P, 4, 10\rangle$, *l'utilisateur décide de changer* P, *c'est-à-dire de lancer une nouvelle recherche sur la base avec :*
$newProg(S_1, P_1)$ *où* P_1 *renvoie les requêtes qui traitent de Tours ou de* 2008 *:*

$$
\begin{array}{lcl}
requêtes\_u(x,"sujet",z) & \leftarrow & requêtes(x,"sujet",z), substring(z,"Tours") \\
requêtes\_u(x,"sujet",z) & \leftarrow & requêtes(x,"sujet",z), substring(z,"2008") \\
requêtes\_u(x,s,t) & \leftarrow & requêtes(x,s,t), requêtes\_u(x,"sujet",z) \\
analyses\_u(a,x) & \leftarrow & requêtes\_u(x,s,t), analyses(a,x) \\
pointeurs\_u(x,x_1,y_1,z_1) & \leftarrow & requêtes\_u(x,s,t), pointeurs(x,x_1,y_1,z_1)
\end{array}
$$

Il obtient le nouveau système où $Etat' = \langle q_1, NULL, 10\rangle$ *présenté Figure 5.4.*

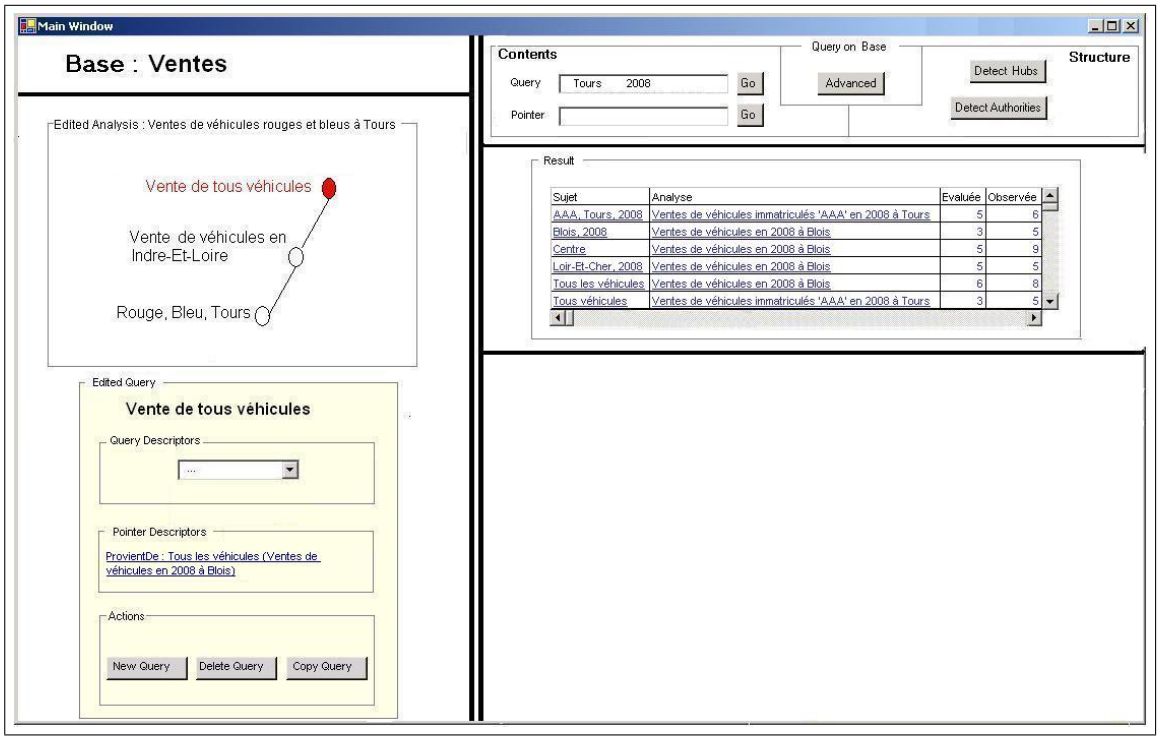


FIGURE 5.4 – Exemple d'opération de navigation : *newProg*

5.2.4.2.2 Les opérations d'édition

Dans ce qui suit, nous présentons les opérations nécessaires à la définition et à l'édition d'analyses et de requêtes. Les opérations d'édition modifient la base et peuvent éventuellement modifier l'état du système.
De la même manière que les primitives classiques d'un LDD[3], ce langage dispose d'opérations complexes comme l'opération *copyQuery* qui duplique la requête naviguée dans l'analyse éditée.

createQueryInNewAnalysis

Cette opération crée une nouvelle requête dans une nouvelle analyse.

Définition 5.2.5 *Soit* $Système = \langle Base, Etat\rangle$, $Etat = \langle P, q_{nav}, q_{ed}\rangle$, *soit* $a_1 \in Dom$ *un identifiant d'analyse qui n'apparaît pas dans* $Base$ *et soit* $q_1 \in Dom$ *un identifiant de requête qui n'apparaît pas non plus dans* $Base$.
$createQueryInNewAnalysis(Système, att, val) = \langle Base', Etat'\rangle$ *où :*

3. *Langage de Définition de Données*

- $Base' = Base \cup \{analyses(a_1, q_1), requêtes(q_1, att, val)\}$
- $Etat' = \langle P, q_{nav}, q_1 \rangle$

createQueryInEditedAnalysis

Cette opération ajoute une requête à l'analyse de la requête q_{ed}.

Définition 5.2.6 *Soit* $Système = \langle Base, Etat \rangle$, $Etat = \langle P, q_{nav}, q_{ed} \rangle$ *et* $q_1 \in Dom$ *un identifiant de requête qui n'apparaît pas dans* $Base$ *et* $analyses(a_{ed}, q_{ed}) \in Base$.
$createQueryInExistingAnalysis(Système, att, val) = \langle Base', Etat' \rangle$ *où :*

- $Base' = Base \cup \{requêtes(q_1, att, val), analyses(a_{ed}, q_1)\}$
- $Etat' = \langle P, q_{nav}, q_1 \rangle$.

copyQuery

Cette opération duplique la requête naviguée q_{nav} dans l'analyse éditée.

Définition 5.2.7 *Soit* $Système = \langle Base, Etat \rangle$, $Etat = \langle P, q_{nav}, q_{ed} \rangle$ *et* $q_1 \in Dom$ *un nouvel identifiant de requête qui n'apparaît pas dans* $Base$.
$copyQuery(Système) = \langle Base', Etat' \rangle$ *où :*

- $Base' = Base \cup I$ *et* $I = \{requêtes(q_1, x, y) | requêtes(q_{nav}, x, y) \in Base\}$
  $\cup \{pointeurs(q_1, q, s, t) | pointeurs(q_{nav}, q, s, t)\}$
  $\cup \{pointeurs(q_1, q_{nav}, inter-analyse, \text{"}provient\ de\text{"})\}$
  $\cup \{pointeurs(q_{nav}, q_1, inter-analyse, \text{"}copiée\ vers\text{"})\}$
- $Etat' = \langle P, q_{nav}, q_1 \rangle$.

Exemple 5.2.11 *Considérons le système* S_1 *présenté Figure 5.1, l'utilisateur décide de dupliquer la requête naviguée, c'est-à-dire la requête identifiée par* 4 *dans l'analyse éditée (qui contient* q_{ed}*), c'est-à-dire l'analyse identifiée par* 5*, avec* $copyQuery(S_1)$.
Il obtient le nouveau système présenté Figure 5.5 :

- $Base' = Base \cup \{requêtes(20, sujet, \text{"}Blois,\ 2008\text{"}),$
  $requêtes(20, code, \text{"}SELECT\ \{[Géographie].[AllG].[Centre].[LoirEtCher].[Blois]\}\ ON$
  $COLUMNS, \{[Temps].[AllT].[2008]\}\ ON\ ROWS$
  $FROM\ [Ventes]\ WHERE\ \{[NomMesures].[Montant]\}\text{"}),$
  $requêtes(20, évaluée, 6), requêtes(20, observée, 9), requêtes(20, résultat, CT_2),$
  $analyses(5, 20), pointeurs(20, 4, inter-analyse, \text{"}provient\ de\text{"}),$
  $pointeurs(4, 20, inter-analyse, \text{"}copiée\ vers\text{"})\}$
- $Etat' = \langle P, 4, 20 \rangle$

addDescToQuery

Cette opération ajoute un descripteur à la requête éditée q_{ed}.

Définition 5.2.8 *Soit* $\langle att, val \rangle$ *un descripteur,* $Système = \langle Base, Etat \rangle$ *et* $Etat = \langle P, q_{nav}, q_{ed} \rangle$.
$addDescToQuery(Système, att, val) = \langle Base', Etat \rangle$ *où* $Base' = Base \cup \{requêtes(q_{ed}, att, val)\}$.

addPointerBetweenQueries

Cette opération ajoute un pointeur entre la requête éditée q_{ed} et la requête naviguée q_{nav}.

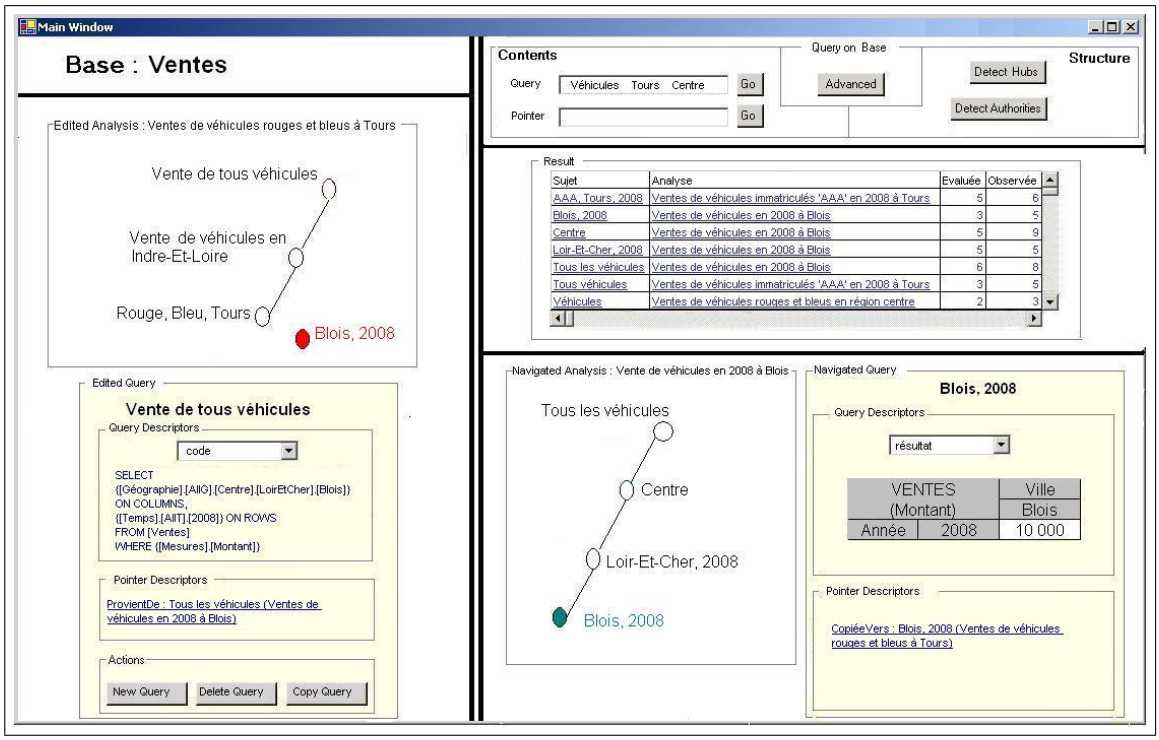


FIGURE 5.5 – Exemple d'opération d'édition : *copyQuery*

Définition 5.2.9 *Soit* $Système = \langle Base, Etat \rangle$, $Etat = \langle P, q_{nav}, q_{ed} \rangle$ *et* $\langle att, val \rangle$ *un descripteur de pointeur.*
$addPointerBetweenQueries(Système, att, val) = \langle Base', Etat \rangle$
avec $Base' = Base \cup \{pointeurs(q_{ed}, q_{nav}, att, val)\}$.

5.3 Exploitation du cadre

Dans cette section, nous présentons comment le modèle peut être utilisé pour fournir de l'information pertinente à un utilisateur qui parcoure une base d'analyses. Nous notons que la sémantique de l'organisation des données dans une base d'analyses est basée sur les descripteurs et les pointeurs. Les deux prochaines sous-sections montrent comment cela peut être exploité.

5.3.1 Exploitation des descripteurs

Il existe deux types de descripteurs : les descripteurs associés aux requêtes et les descripteurs associés aux pointeurs. Les descripteurs peuvent être ajoutés par l'utilisateur, quand il édite une requête ou un pointeur, ou par le système lui-même pour mettre à jour automatiquement les informations collectées sur les données.

5.3.1.1 Descripteurs associés aux requêtes

La plupart du temps, ces descripteurs sont ajoutés par l'utilisateur pour caractériser les requêtes. Voici quelques exemples de tels descripteurs :

- *sujet*, dont la valeur est une description textuelle de la requête,
- *analyse*, dont la valeur indique à quelle analyse appartient la requête,
- *code*, dont la valeur est le code MDX ou autre de la requête,
- *résultat*, dont la valeur est le résultat de la requête, habituellement affiché sous la forme d'un tableau croisé, comme dans la zone en bas à droite de la Figure 5.1.

Des exemples d'utilisation de ces descripteurs pour interroger la base d'analyses ont été donnés dans les sections précédentes.

Dans certains cas, les descripteurs sont ajoutés et mis à jour par le système afin de collecter des informations relatives à la navigation des utilisateurs.
Voici quelques exemples de tels descripteurs :

- *évaluée*, dont la valeur est un compteur qui indique combien de fois la requête a été évaluée. A chaque fois que le descripteur *résultat* est affiché, le compteur est incrémenté.
- *observée*, dont la valeur est un compteur qui indique combien de fois la requête a été observée. A chaque fois que les requêtes sont observées, le compteur est incrémenté.

Nous illustrons ici comment ces descripteurs peuvent être utilisés.

Exemple 5.3.1 *Supposons que l'ordonnancement des requêtes candidates à la recommandation se fasse en fonction du nombre d'occurrences de chacune d'elles dans l'ensemble des requêtes candidates. Il est possible de lancer un programme $Datalog^{\neg}$ qui détecte parmi les requêtes candidates, celles qui ont été lancées (évaluées) un certain nombre de fois ou qui sont les plus fréquentes.*
Par exemple, le programme suivant permet de détecter quelles requêtes ont été lancées (évaluées) plus de 10 fois :

$$
\begin{array}{lcl}
analyses\_u(a, q_1) & \leftarrow & requêtes(q_1, \text{"}évaluée\text{"}, x), x > 10, analyses(a, q_1) \\
requêtes\_u(q_1, att, val) & \leftarrow & requêtes(q_1, \text{"}évaluée\text{"}, x), x > 10, requêtes(q_1, att, val) \\
pointeurs\_u(q_1, q_2, att, val) & \leftarrow & requêtes(q_1, \text{"}évaluée\text{"}, x), x > 10, \\
 & & pointeurs(q_1, q_2, att, val)
\end{array}
$$

5.3.1.2 Descripteurs associés aux pointeurs

Ces descripteurs sont utilisés pour indiquer comment les requêtes sont organisées à l'intérieur d'une analyse (c'est le cas des pointeurs *intra-analyse*) ou bien sont relatifs à des requêtes provenant d'autres analyses (c'est le cas des pointeurs *inter-analyse*).

En ce qui concerne les pointeurs "intra-analyse", ces descripteurs peuvent être :

- *ordre d'importance*, dont la valeur reflète une relation d'ordre sur les requêtes d'une analyse qui est interprété comme un ordre d'importance pertinent pour l'utilisateur.
- *raffinement de requête*, dans ce cas, le pointeur connecte deux requêtes telles que l'une raffine l'autre,
- *séquencement*, dont la valeur reflète dans quel ordre les requêtes ont été définies.

En ce qui concerne les pointeurs "inter-analyse", en plus de la trace que l'utilisateur voudrait garder pour indiquer qu'une requête d'une autre analyse est intéressante, de tels descripteurs peuvent être ajoutés automatiquement par le système pour conserver des informations relatives aux opérations d'édition lancées par l'utilisateur. Ces descripteurs peuvent être :

- *provient de*, qui indique que la requête source est une copie d'une autre requête (c'est-à-dire la cible du pointeur), et donc a été empruntée à une autre analyse,
- *copiée vers*, qui indique que la requête source a été copiée dans une autre analyse.
- *reco*, qui indique que la requête cible a été recommandée à la suite de la requête source.

Exemple 5.3.2 *Considérons les quatre requêtes q_1, q_2^2, q_5 et q_6 de la session s_3 du log présenté Annexe A.*

Dans la Section 5.1, Patrick a lancé les requêtes dans l'ordre suivant : q_1, puis q_2^2, puis q_6, puis q_5. Ainsi, si nous décidons d'ordonner les requêtes selon le séquencement, nous aurions les trois pointeurs :

- $pointeurs(q_1, q_2^2, intra - analyse, "sequencement")$
- $pointeurs(q_2^2, q_6, intra - analyse, "sequencement")$
- $pointeurs(q_6, q_5, intra - analyse, "sequencement")$

et l'analyse correspondante serait : $q_1 \overset{seq}{\rightarrow} q_2^2 \overset{seq}{\rightarrow} q_6 \overset{seq}{\rightarrow} q_5$.

Toujours dans la Section 5.1, Patrick décide d'ordonner les requêtes en fonction du raffinement de requêtes, nous avons donc les trois pointeurs :

- $pointeurs(q_1, q_2^2, intra - analyse, "raffinement")$
- $pointeurs(q_2^2, q_5, intra - analyse, "raffinement")$
- $pointeurs(q_5, q_6, intra - analyse, "raffinement")$

et l'analyse correspondante est : $q_1 \overset{raff}{\rightarrow} q_2^2 \overset{raff}{\rightarrow} q_5 \overset{raff}{\rightarrow} q_6$.

Enfin, si nous décidons d'ordonner les requêtes selon un ordre d'importance, en considérant que q_5 est la plus importante (puisqu'elle permet de répondre à notre questionnement) et que q_1 est la moins importante, nous aurions les trois pointeurs :

- $pointeurs(q_5, q_6, intra - analyse, "importance")$
- $pointeurs(q_6, q_2^2, intra - analyse, "importance")$
- $pointeurs(q_2^2, q_1, intra - analyse, "importance")$

et l'analyse correspondante serait : $q_5 \overset{imp}{\rightarrow} q_6 \overset{imp}{\rightarrow} q_2^2 \overset{imp}{\rightarrow} q_1$.

Nous obtenons donc trois analyses différentes. De sorte que le système de recommandations ne proposera pas les mêmes recommandations selon le descripteur du pointeur. En effet, si nous considérons, par exemple, que la requête recommandée est la dernière requête de la session candidate, comme proposé Section 3.3.2 et que l'analyse a_1 contenant les requêtes q_1, q_2^2, q_5 et q_6 est une candidate, alors :

- *si les requêtes de a_1 sont reliées par des pointeurs "séquencement", la recommandation sera q_5,*
- *si les requêtes de a_1 sont reliées par des pointeurs "raffinement", la recommandation sera q_6,*
- *si les requêtes de a_1 sont reliées par des pointeurs "importance", la recommandation sera q_1.*

Les descripteurs de pointeurs vont donc influer sur les recommandations.

5.3.2 Exploitation des pointeurs

5.3.2.1 Analyses hubs et analyses autorités

Dans la Section 5.2.3.1.1, nous avons noté qu'une base d'analyses peut être décrite comme un graphe dont les nœuds sont les requêtes et les arcs sont les pointeurs entre les requêtes. Ici, nous proposons de voir une base d'analyses comme un graphe où les nœuds sont les analyses et les arcs sont les liens entre les analyses. En fait, un lien entre deux analyses a_1 et a_2 est créé en fonction des pointeurs existants entre les requêtes de a_1 et celles de a_2. En effet, s'il existe au moins un pointeur, quel qu'il soit, reliant l'une des requêtes de a_1 à l'une des requêtes de a_2, alors un lien est créé de a_1

vers a_2. De sorte que les notions de *hubs* et *autorités* de [Kle99a], présentées Section 3.3.4, peuvent être adaptées de la manière suivante :

- Une analyse *Autorité* est une analyse qui est pointée par toutes les autres analyses de la base d'analyses,
- Une analyse *Hub* est une analyse qui permet d'accéder à toutes les autres analyses.

De plus, les analyses Hub et les analyses Autorité, comme nous les avons définies, peuvent être trouvées en utilisant un programme Datalog$^\neg$ bien formé, avec une division (d'où la nécessité d'avoir la négation dans le langage) :

- Quels sont les analyses Hub ?

$$
\begin{array}{lcl}
link(a_1, a_2) & \leftarrow & analyses(a_1, q_1), analyses(a_2, q_2), \\
 & & pointeurs(q_1, q_2, \_, \_) \\
possiblelink(a_1, a_2) & \leftarrow & analyses(a_1, \_), analyses(a_2, \_) \\
nolink(a_1, a_2) & \leftarrow & possiblelink(a_1, a_2), \neg link(a_1, a_2) \\
analyses\_u(a_1, x) & \leftarrow & analyses(a_1, x), analyses(a_2, y), \neg nolink(a_1, a_2) \\
requêtes\_u(x, c, v) & \leftarrow & analyses\_u(a_1, x), requêtes(x, c, v) \\
pointeurs\_u(x_1, x_2, c, v,) & \leftarrow & analyses\_u(a_1, x_1), analyses\_u(a_2, x_2), \\
 & & pointeurs(x_1, x_2, c, v)
\end{array}
$$

- Quels sont les analyses Autorité ?

$$
\begin{array}{lcl}
link(a_2, a_1) & \leftarrow & analyses(a_1, q_1), analyses(a_2, q_2), \\
 & & pointeurs(q_2, q_1, \_, \_) \\
possiblelink(a_2, a_1) & \leftarrow & analyses(a_1, \_), analyses(a_2, \_) \\
nolink(a_2, a_1) & \leftarrow & possiblelink(a_2, a_1), \neg link(a_2, a_1) \\
analyses\_u(a_1, x) & \leftarrow & analyses(a_1, x), analyses(a_2, y), \neg nolink(a_2, a_1) \\
requêtes\__u(x, c, v) & \leftarrow & analyses\_u(a_1, x), requêtes(x, c, v) \\
pointeurs\_u(x_2, x_1, c, v,) & \leftarrow & analyses\_u(a_1, x_1), analyses\_u(a_2, x_2), \\
 & & pointeurs(x_2, x_1, c, v) \\
authority(a) & \leftarrow & analyses\_u(a, \_)
\end{array}
$$

Ainsi, par exemple, il est possible de restreindre la recherche des requêtes candidates à la recommandation à une base d'analyses constituée uniquement des analyses *Hubs* ou des analyses *Autorités*.

5.3.2.2 Recommandations suivies

Une autre utilisation possible d'un pointeur ayant pour descripteur *reco* est de l'associer à un pointeur ayant pour descripteur *copiée vers*. En effet, durant son analyse courante, l'utilisateur se voit proposer un ensemble de recommandations pouvant faire suite à sa requête courante. De ce fait, le système ajoute un pointeur *"reco"* entre la requête courante et chaque requête recommandée. Si l'utilisateur accepte l'une des recommandations et lance la requête recommandée choisie, alors le système ajoute un pointeur *"copiée vers"* entre la requête recommandée et la nouvelle requête lancée par l'utilisateur. De sorte que si une requête est pointée par un pointeur *"reco"* et qu'elle pointe vers une autre requête via un pointeur *"copiée vers"*, nous pouvons conclure que cette requête est une recommandation qui a été suivie.
Ainsi, il sera aisé, par exemple, de détecter par programme bien formé, si la méthode de recommandation est efficace ou non, en comparant les recommandations calculées à celles effectivement suivies.

Exemple 5.3.3 *Nous illustrons cette proposition par un exemple simple. Considérons le log de sessions de requêtes présenté Annexe A où, par souci de lisibilité, seuls les pointeur "séquencement"*

apparaissent. Supposons qu'un utilisateur lance sa première requête q_1 *et que le système de recommandations lui propose comme recommandations les requêtes* $q^2_{2_{s_1}}$ *et* $q^2_{3_{s_2}}$*. Le système d'organisation des analyses, quant à lui, va ajouter deux pointeurs "reco" :*

- $pointeurs(q_1, q^2_{2_{s_1}}, inter - analyse, "reco")$
- $pointeurs(q_1, q^2_{3_{s_2}}, inter - analyse, "reco")$

Aucune de ces requêtes recommandées n'intéresse l'utilisateur. Il décide de lancer sa propre requête q_2*. Le système de recommandation lui propose comme recommandations les requêtes* $q^2_{3_{s_1}}$ *et* $q_{3_{s_2}}$*. Le système d'organisation des analyses, quant à lui, va ajouter deux pointeurs "reco" :*

- $pointeurs(q_2, q^2_{3_{s_1}}, inter - analyse, "reco")$
- $pointeurs(q_2, q_{3_{s_2}}, inter - analyse, "reco")$

L'utilisateur accepte $q_{3_{s_2}}$ *comme recommandation, la copie dans son analyse et la lance. Le pointeur :* $pointeurs(q_1, q_{3_{s_2}}, inter - analyse, "copiée\ vers")$ *est alors ajouté.*
Seule la requête $q_{3_{s_2}}$ *est une recommandation suivie à la suite de la requête* q_1*.*
La Figure 5.6 illustre cet exemple.

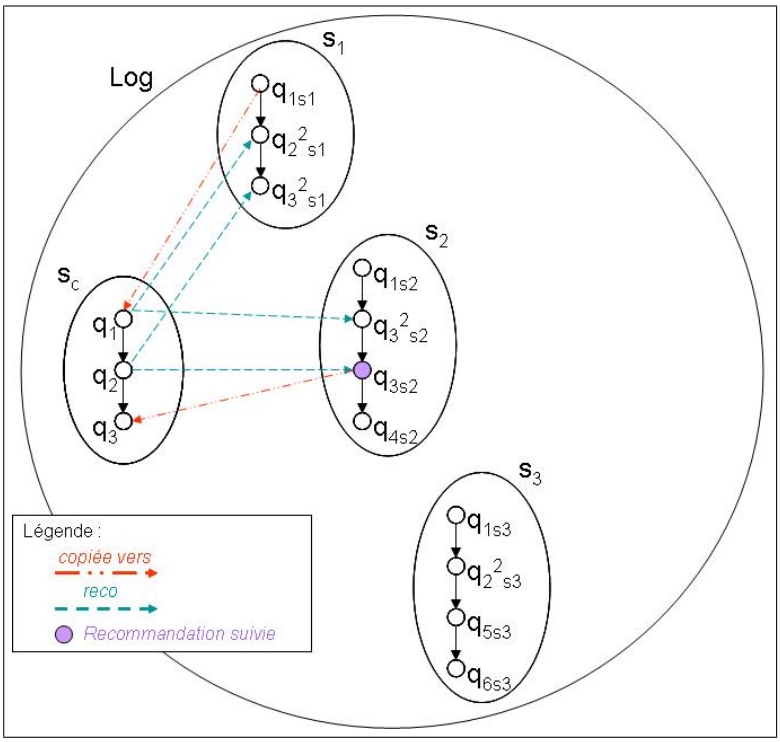


FIGURE 5.6 – Exemple de recommandation suivie

5.4 Synthèse

Dans ce chapitre, nous avons proposé un cadre générique d'organisation des analyses dont le modèle : la *Base d'analyses*, se décompose en deux niveaux :

- le niveau des données qui peuvent être des requêtes sur un cube de données, des analyses (c'est-à-dire des séquences de requêtes) et des pointeurs (c'est-à-dire des liens entre requêtes),

- le niveau du système qui modélise ce qui est présenté à l'utilisateur, à savoir, la *Base* utilisée, l'interrogation de la *Base*, l'analyse et la requête éditées, l'analyse et la requête naviguées,

et qui possède un langage de définition, de manipulation et de parcours de cette *Base* constitué :

- d'opérations de navigation pour parcourir/observer les éléments constitutifs de la *Base*,
- d'opérations d'édition pour éditer les éléments constitutifs de la *Base*.

Puis nous proposons une exploitation particulière de ce cadre générique adaptée aux recommandations de requêtes *MDX* et consistant en l'exploitation :

- des descripteurs des éléments constitutifs de la *Base*, à savoir, les analyses, les requêtes et les pointeurs, permettant, par exemple, d'obtenir les requêtes les plus fréquemment lancées sur un cube de données ou d'obtenir des recommandations différentes selon le séquencement de requêtes choisi dans une analyse.
- des pointeurs entre les requêtes de la *Base*, permettant, par exemple, de restreindre la recherche de requêtes candidates à la recommandation à une *Base* constituée d'éléments sélectionnés ou de connaître quelles requêtes recommandées ont été proposées à la suite d'une requête donnée et quelles recommandations ont réellement été suivies.

Chapitre 6

Conclusion et Travaux futurs

Sommaire

Dans ce chapitre consacré à la conclusion générale de la thèse, nous faisons tout d'abord une synthèse et un bilan de nos travaux pour, ensuite, donner quelques-unes de nos perspectives de recherche.

6.1 Synthèse des travaux

Dans un premier temps, nous avons présenté une vue d'ensemble des bases de données multidimensionnelles, puis un aperçu des outils existants de traitement de logs de requêtes et une vue d'ensemble des systèmes de recommandations. Puis nous avons présenté un état de l'art sur les méthodes d'aide à l'exploration dans le domaine des bases de données (relationnelles ou multidimensionnelles). Enfin, au cours de notre étude bibliographique, nous avons vu que le lien entre les recommandations et les bases de données n'est pas très développé. Cependant, il apparaît que :

- La recommandation consiste à guider l'utilisateur lors de son exploration de la base de données,
- Un utilisateur explore un cube de données en lançant une séquence de requêtes,
- Plusieurs utilisateurs différents peuvent explorer des parties proches du cube de données.

Sur la base de ces constatations, et sachant qu'à notre connaissance, il n'existe pas de travaux sur la recommandation tenant compte des particularités de l'interrogation d'un cube de données par des sessions de requêtes MDX dans un environnement multi-utilisateurs, nous avons, dans un second temps, proposé une méthode de recommandation collaborative tenant compte de ces particularités.

Plus précisément, notre contribution comprend :

1. La proposition d'un cadre générique de génération de recommandations de requêtes MDX ainsi que quatre instanciations de ce cadre. En effet, ce cadre est générique dans le sens où il peut être instancié de différentes manières afin de changer la méthode de calcul des recommandations. Ce cadre est constitué de trois étapes :
   (a) Prétraitement de l'ensemble des sessions de requêtes enregistrées dans un log de sessions de requêtes,
   (b) Génération des recommandations candidates,
   (c) Ordonnancement des recommandations candidates.
2. La réalisation d'un prototype intégrant et validant notre technique de recommandation. Ce prototype, nommé *RecoOLAP* consiste en une application où l'utilisateur explorant un cube de données en lançant des requêtes MDX se voit recommander des requêtes MDX pouvant faire suite à sa session d'analyse courante. Ce prototype nous a permis, via une batterie de tests, de vérifier l'efficacité de notre technique de recommandation et de comparer différentes instanciations possibles de notre cadre générique de génération de recommandations.
3. Un cadre générique d'organisation des analyses afin de pouvoir, notamment, partager des requêtes, naviguer au travers des requêtes du log et / ou des requêtes recommandées obtenues grâce à notre cadre générique de génération de recommandations ou encore de les annoter / décrire. Ce cadre possède un modèle : la *Base d'analyses* ainsi qu'un langage de définition, de manipulation et de parcours de cette *Base*. Notons qu'une exploitation de ce cadre générique adaptée aux recommandations de requêtes MDX a aussi été proposée.

Dans la section suivante, nous présentons dans quelles directions ce travail pourra être poursuivi.

6.2 Perspectives envisagées

L'objectif général de notre travail étant la proposition d'un cadre générique de génération de recommandations dans le cadre de l'exploration collaborative de cubes de données, nous continuerons à développer et enrichir *Reco*OLAP. Ce développement passera par des méthodes d'optimisation (par exemple, des méthodes d'indexation du log de sessions de requêtes) pour améliorer les performances du système. Par ailleurs, plusieurs autres perspectives s'offrent à nous. Nous en faisons une ébauche ci-dessous.

Aller plus loin dans la recommandation
Le travail présenté Chapitre 3 est une base prometteuse pour proposer des requêtes recommandées à l'utilisateur en étant au plus près de ses attentes. Cependant, certaines recommandations ne sont pas encore possibles, comme, par exemple, lorsque deux sessions s_1 et s_2 ne diffèrent que d'une sélection. En effet, dans ce cas, nous voudrions proposer comme recommandation à la suite de s_2, une des requêtes de s_1 où seule la sélection est modifiée.

Exemple 6.2.1 *Considérons les deux utilisateurs A et B interrogeant le même cube de données et supposons que le log de sessions de requêtes correspondant à l'exploration de ce cube contienne la séquence de requêtes s_1 lancées par B : q_1, puis q_2 puis q_3 telles que :*
- *q_1 : Quantité de véhicules vendus en France en 2007*
- *q_2 : Quantité de véhicules vendus en Loir-et-Cher en 2007*
- *q_3 : Quantité de véhicules vendus à Blois en 2007*

Supposons que l'utilisateur A, quant à lui, commence à interroger le cube via la séquence s_2 de requêtes q_4 puis q_5 telles que :
- *q_4 : Quantité de véhicules vendus en France toutes années confondues*
- *q_5 : Quantité de véhicules vendus en Loir-et-Cher en 2008*

Les utilisateurs A et B semblent intéressés par les mêmes données. Par conséquent, les requêtes de l'utilisateur B peuvent servir de guide à l'utilisateur A. Intuitivement, la requête q_6 pourrait être proposée à l'utilisateur A telle que :
- *q_6 : Quantité de véhicules vendus dans les villes du Loir-et-Cher en 2008*

A l'heure actuelle, notre système de génération de recommandations ne permet pas de repérer de telles sessions, ni de proposer une telle recommandation.

De manière plus générale, il s'agira de recommander des requêtes qui ne sont pas copiées ou construites à partir des requêtes du log.

Expérimentations sur des données réelles
Les expériences menées et dont les résultats sont présentés Chapitre 4 ont été réalisées sur des données synthétiques. Or, afin d'améliorer la qualité des requêtes recommandées et surtout l'évaluation des diverses instanciations de notre cadre générique de génération de recommandations, il faudrait pouvoir mener les expérimentations sur des données réelles.
Dans cette optique, un partenariat avec l'*IRSA* (**I**nstitut inter **R**égional pour la **SA**nté) a été mis en

place afin d'analyser environ 500 000 questionnaires d'examens de santé. La constitution d'un entrepôt des données des questionnaires est en cours. Enfin, cet entrepôt servira à valider des approches sur des données réelles.

Contribution à un système collaboratif de gestion de requêtes
A long terme, il s'agira de 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 le cadre présenté dans cette thèse et devrait également inclure une nouvelle méthode, prenant en compte les valeurs des mesures et pas seulement les références des cellules, proposée dans [GMNS09] ainsi que des techniques basées sur le contenu [CEP09] aussi bien que les méthodes combinant le contexte et le profil de l'utilisateur [JRTZ09, BGM+05, GR09].

Remarquons enfin que le travail proposé Chapitre 5 est une contribution au problème plus général évoqué par [KBG+09]. En effet, les auteurs indiquent le besoin de systèmes offrant des possibilités sophistiquées de gestion de requêtes, par exemple, en tirant profit des sessions précédentes, afin d'aider les utilisateurs de bases de données qui analysent de grandes quantités de données.

Bibliographie

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling Multidimensional Databases. In *ICDE*, pages 232–243, 1997.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AS04] Mina Akaishi and Nicolas Spyratos. Discovering Implicit Relationships in a Web of Contexts. In *Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets*, pages 175–188, 2004.

[AST03] Mina Akaishi, Nicolas Spyratos, and Yuzuru Tanaka. Contextual Search in Large Collections of Information Resources. In *EJC : European-Japanese Conference on Information Modelling and Knowledge Bases*, pages 295–302, 2003.

[AT05] G. Adomavicius and A. Tuzhilin. 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*, 17(6) :734–749, 2005.

[Bel47] Richard Bellman. *Dynamic programming.* Princeton, New Jersey : Princeton University Press. XXV, 342 p., 1947.

[BGM+05] Ladjel Bellatreche, Arnaud Giacometti, Patrick Marcel, Hassina Mouloudi, and Dominique Laurent. A personalization framework for OLAP queries. In *DOLAP*, pages 9–18, 2005.

[BGMM06] Ladjel Bellatreche, Arnaud Giacometti, Patrick Marcel, and Hassina Mouloudi. Personalization of MDX Queries. In *BDA*, 2006.

[BH90] Fred Buckley and Frank Harary. *Distance in Graphs.* Addison-Wesley, Reading, MA, USA, 1990.

[BHK98] John S. Breese, David Heckerman, and Carl Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, 1998. Morgan Kaufmann.

[BP98] Daniel Billsus and Michael J. Pazzani. Learning Collaborative Information Filters. In *ICML '98 : Proceedings of the Fifteenth International Conference on Machine Learning*, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[BS97] Marko Balabanovic and Yoav Shoham. Fab : content-based, collaborative recommendation. *Communications of the ACM*, 40(3) :66–72, 1997.

[BYHM04] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Query Recommendation Using Query Logs in Search Engines. In Wolfgang Lindner, Marco Mesiti, Can Türker, Yannis Tzitzikas, and Athena Vakali, editors, *EDBT Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 588–596. Springer, 2004.

[BYHMD05] Ricardo Baeza-Yates, Carlos Hurtado, Marcelo Mendoza, and Georges Dupret. Modeling User Search Behavior. In *LA-WEB '05 : Proceedings of the Third Latin American Web Congress*, page 242, Washington, DC, USA, 2005. IEEE Computer Society.

[BYRN99] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[CCD+08] Véronique Cariou, Jérôme Cubillé, Christian Derquenne, Sabine Goutier, Françoise Guisnel, and Henri Klajnmic. Built-In Indicators to Discover Interesting Drill Paths in a Cube. In *DaWaK '08 : Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*, pages 33–44, Berlin, Heidelberg, 2008. Springer-Verlag.

[CCS93] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to User-Analysis : An IT Mandate, 1993.

[CD91] F. Cuppens and R. Demolombe. Extending answers to neighbour entities in a cooperative answering context. *Decis. Support Syst.*, 7(1) :1–11, 1991.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1) :65–74, 1997.

[CEP09] Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Query Recommendations for Interactive Database Exploration. In *SSDBM*, pages 3–18, 2009.

[CGL+07] Yeow-Wei Choong, Arnaud Giacometti, Dominique Laurent, Patrick Marcel, Elsa Negre, and Nicolas Spyratos. Context-based exploitation of data warehouses. In *3èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2007), Poitiers*, volume B-3 of *RNTI*, pages 131–146. Cépaduès, Juin 2007.

[CT98] Luca Cabibbo and Riccardo Torlone. From a Procedural to a Visual Query Language for OLAP. In Maurizio Rafanelli and Matthias Jarke, editors, *10th International Conference on Scientific and Statistical Database Management, Proceedings, Capri, Italy, July 1-3*, pages 74–83. IEEE Computer Society, 1998.

[DB209a] DB2. DB2 Query Management Facility. Available at http://www.ibm.com/software/data/qmf, 2009.

[DB209b] DB2. DB2 Query Patroller. Available at http://www.ibm.com/software/data/db2/querypatroller, 2009.

[Dij71] Edsger W. Dijkstra. A short introduction to the art of programming. circulated privately, August 1971.

[DKK05] Jens-Peter Dittrich, Donald Kossmann, and Alexander Kreutz. Bridging the gap between OLAP and SQL. In *VLDB '05 : Proceedings of the 31st international conference on Very large data bases*, pages 1031–1042. VLDB Endowment, 2005.

[EMS09] EMS. EMS SQL Management Studio for ORACLE. Available at http://www.sqlmanager.net/fr/products/studio/oracle, 2009.

[FK04] Enrico Franconi and Anand Kamble. The GMD Data Model and Algebra for Multidimensional Information. In *CAiSE*, pages 446–462, 2004.

[Flo62] Robert W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345, 1962.

[FS02] Yongjian Fu and Ming-Yi Shih. A Framework for Personal Web Usage Mining. In *International Conference on Internet Computing*, pages 595–600, 2002.

[GL97] Marc Gyssens and Laks V. S. Lakshmanan. A Foundation for Multi-dimensional Databases. In *VLDB*, pages 106–115, 1997.

[GMN06] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. OLAP : un pas vers la navigation. In *2ème journée francophone sur les Entrepôts de Données et l'Analyse en ligne (EDA 2006), Versailles*, volume B-2 of *RNTI*, pages 105–118. Cépaduès, Juin 2006.

[GMN08] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. A framework for recommending OLAP queries. In *DOLAP*, pages 73–80, 2008.

[GMN09] Arnaud Giacometti, Patrick Marcel, and Elsa Negre. Recommending Multidimensional Queries. In *DaWaK*, pages 453–466, 2009.

[GMNS09] Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query recommendations for OLAP discovery driven analysis. In *DOLAP*, 2009.

[GMR98] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model : A conceptual model for data warehouses. *Int. J. Cooperative Inf. Syst.*, 7(2-3) :215–247, 1998.

[GR09] Matteo Golfarelli and Stefano Rizzi. Expressing OLAP Preferences. In *SSDBM 2009 : Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, pages 83–91, Berlin, Heidelberg, 2009. Springer-Verlag.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *Syst. Tech. J.*, 29 :147–160, 1950.

[Hau14] Felix Hausdorff. *Grundzüge der Mengenlehre*. von Veit, 1914.

[HG08] Felix Hernández and Elena Gaudioso. Evaluation of recommender systems : A new approach. *Expert Systems with Applications*, 35(3) :790–804, October 2008.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4 :100–107, 1968.

[HSRF95] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *CHI '95 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[Inm92] W. H. Inmon. *Building the data warehouse*. QED Information Sciences, Inc., Wellesley, MA, USA, 1992.

[JLS99] H. V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava. What can Hierarchies do for Data Warehouses ? In *VLDB*, pages 530–541, 1999.

[JLVV01] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[JRTZ09] Houssem Jerbi, Franck Ravat, Olivier Teste, and Gilles Zurfluh. Applying Recommendation Technology in OLAP Systems. In *International Conference on Enterprise Information Systems (ICEIS), Milan (Italie), 06/05/2009-10/05/2009*, number 24 in LNBIP, pages 220–233, http ://www.springerlink.com, 2009. Springer.

[KBG+09] Nodira Khoussainova, Magdalena Balazinska, Wolfgang Gatterbauer, YongChul Kwon, and Dan Suciu. A Case for A Collaborative Query Management System. In *CIDR*. www.crdrdb.org, 2009.

[Ken71] Allen Kent. *Information Analysis and Retrieval.* John Wiley and sons, Inc., 1971.

[KI04] Georgia Koutrika and Yannis E. Ioannidis. Personalization of Queries in Database Systems. In *ICDE*, pages 597–608, 2004.

[Kim96] Ralph Kimball. *The Data Warehouse Toolkit : Practical Techniques for Building Dimensional Data Warehouses.* John Wiley, 1996.

[KK06] Przemyslaw Kazienko and Pawel Kolodziejski. Personalized Integration of Recommendation Methods for e-commerce. *IJCSA*, 3(3) :12–26, 2006.

[Kle99a] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, 46(5) :604–632, 1999.

[Kle99b] Jon M. Kleinberg. Hubs, authorities, and communities. *ACM Comput. Surv.*, 31(4es) :5, 1999.

[KR87] L. Kaufmann and P. J. Rousseeuw. Clustering by means of medoids. In Y. Dodge, editor, *Statistical Data Analysis based on the L1 Norm*, pages 405–416. Elsevier/North Holland, Amsterdam, 1987.

[KR02] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling.* John Wiley & Sons, Inc., New York, USA, 2002.

[KS96] Vipul Kashyap and Amit P. Sheth. Semantic and Schematic Similarities Between Database Objects : A Context-B ased Approach. *VLDB Journal : Very Large Data Bases*, 5(4) :276–304, 1996.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[LVC+03] Peter Lyman, Hal R. Varian, Peter Charles, Nathan Good, Laheem L. Jordan, and Joyojeet Pal. How much information ? Available at http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/, 2003.

[McC96] J. McCarthy. A logical AI approach to context, 1996. Technical note, Computer Science Department, Stanford University, Stanford, CA, 1996.

[Mic98] Microsoft Corporation. Multidimensional Expressions (MDX) Reference. Available at http://msdn.microsoft.com/en-us/library/ms145506.aspx, 1998.

[MVSV03] Andreas S. Maniatis, Panos Vassiliadis, Spiros Skiadopoulos, and Yannis Vassiliou. CPM : A Cube Presentation Model for OLAP. In *DaWaK*, pages 4–13, 2003.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1) :31–88, 2001.

[Neg09] Elsa Negre. Recommandations personnalisées de requêtes MDX. In *EDA 2009 : Journée francophone sur les Entrepôts de Données et l'Analyse en ligne, Montpellier*, Juin 2009.

[ORA09] ORACLE. Oracle OLAP 11g. Available at http://www.oracle.com/technology/products/bi/olap, 2009.

[Pen09] Pentaho Corporation. Mondrian open source OLAP engine. Available at http://mondrian.pentaho.org, 2009.

[PPPS03] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, and Constantine D. Spyropoulos. Web Usage Mining as a Tool for Personalization : A Survey. *User Modeling and User-Adapted Interaction*, 13(4) :311–372, November 2003.

[RA07] Oscar Romero and Alberto Abelló. On the Need of a Reference Algebra for OLAP. In *DaWaK*, pages 99–110, 2007.

[RTTZ08] Franck Ravat, Olivier Teste, Ronan Tournier, and Gilles Zurfluh. Algebraic and graphic languages for olap manipulations. *IJDWM*, 4(1) :17–46, 2008.

[Sal83] Gerard Salton. *Introduction to Modern Information Retrieval (McGraw-Hill Computer Science Series)*. McGraw-Hill Companies, September 1983.

[Sap99] Carsten Sapia. On Modeling and Predicting Query Behavior in OLAP Systems. In *DMDW*, pages 2.1–2.10, 1999.

[Sap00] Carsten Sapia. PROMISE : Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems. In *DaWaK*, pages 224–233, 2000.

[Sar99] Sunita Sarawagi. Explaining Differences in Multidimensional Aggregates. In *VLDB*, pages 42–53, 1999.

[Sar00] Sunita Sarawagi. User-Adaptive Exploration of Multidimensional Data. In *VLDB*, pages 307–316, 2000.

[SAS09] SAS. SAS OLAP Server. Available at http://www.sas.com/technologies/dw/storage/mddb, 2009.

[SCDT00] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web Usage Mining : Discovery and Applications of Usage Patterns from Web data. *SIGKDD Explorations*, 1(2) :12–23, 2000.

[SKR01] J. Ben Schafer, Joseph A. Konstan, and John Riedl. E-commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, 5(1/2) :115–153, 2001.

[SS01] Gayatri Sathe and Sunita Sarawagi. Intelligent Rollups in Multidimensional OLAP Data. In *VLDB*, pages 531–540, 2001.

[Sun08] Sun microsystems. MySQL. Available at http://www.mysql.fr, 2008.

[TACS98] M. Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Context in Information Bases. In *CoopIS*, pages 260–270, 1998.

[TACS99] M. Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. Contextualization as an Abstraction Mechanism for Conceptual modelling. In *ER*, pages 475–489, 1999.

[TACS02] M. Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nicolas Spyratos. A theory of contexts in information bases. *Inf. Syst.*, 27(3) :151–191, 2002.

[TSM01] Thomas Thalhammer, Michael Schrefl, and Mukesh K. Mohania. Active data warehouses : complementing OLAP with analysis rules. *Data Knowl. Eng.*, 39(3) :241–269, 2001.

[UFA+98] Lyle Ungar, Dean Foster, Ellen Andre, Star Wars, Fred Star Wars, Dean Star Wars, and Jason Hiver Whispers. Clustering Methods for Collaborative Filtering. AAAI Press, 1998.

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[VS00] Panos Vassiliadis and Spiros Skiadopoulos. Modelling and Optimisation Issues for Multidimensional Databases. In *CAiSE*, pages 482–497, 2000.

[WBC07] Ryen W. White, Mikhail Bilenko, and Silviu Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *SIGIR*, pages 159–166, 2007.

[YPS09] Xiaoyan Yang, Cecilia M. Procopiuc, and Divesh Srivastava. Recommending Join Queries via Query Log Analysis. In *ICDE*, pages 964–975, 2009.

Annexe A

Exemple de données

Ces requêtes sont posées sur le cube $MesVentes$ présentés Section 2.1.1.

Session $s_c = \langle q_1, q_2, q_3 \rangle$: Ventes de véhicules rouges et bleus à Tours

```
SELECT  {[Véhicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].Members} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].[Centre].[IndreEtLoire]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS,
        {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}
```

Session $s_1 = \langle q_1, q_2^2, q_3^2 \rangle$: Ventes de véhicules rouges et bleus en région Centre

```
SELECT  {[Véhicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].Members} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].[Centre]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS,
        {[Géographie].[AllG].[Centre]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}
```

Session $s_2 = \langle q_1, q_3^2, q_3, q_4 \rangle$: Ventes de véhicules immatriculés 'AAA' en 2008 à Tours

```
SELECT  {[Vehicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].Members} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS,
        {[Géographie].[AllG].[Centre]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].[Rouge], [Véhicules].[AllV].[Bleu]} ON COLUMNS,
        {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  Crossjoin({[Véhicules].[AllV].[Bleu].[AAA]}, {[Temps].[AllT].[2008]}) ON COLUMNS,
        {[Géographie].[AllG].[Centre].[IndreEtLoire].[Tours]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}
```

Session $s_3 = \langle q_1, q_2^2, q_5, q_6 \rangle$: Ventes de véhicules en 2008 à Blois

```
SELECT  {[Vehicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].Members} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Véhicules].[AllV].Members} ON COLUMNS,
        {[Géographie].[AllG].[Centre]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Géographie].[AllG].[Centre].[LoirEtCher].Children} ON COLUMNS,
        {[Temps].[AllT].[2008]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}

SELECT  {[Géographie].[AllG].[Centre].[LoirEtCher].[Blois]} ON COLUMNS,
        {[Temps].[AllT].[2008]} ON ROWS
FROM    [Ventes]
WHERE   {[NomMesures].[Montant]}
```

Annexe B

Base de données *Foodmart* et Cube *Sales*

Voici le schéma de la base de données *FoodMart* (Figure B.1) fournie par le moteur OLAP Mondrian [Pen09] ainsi que le Cube *Sales* (Figure B.2) correspondant à la table de fait $Sales\_fact\_1997$.

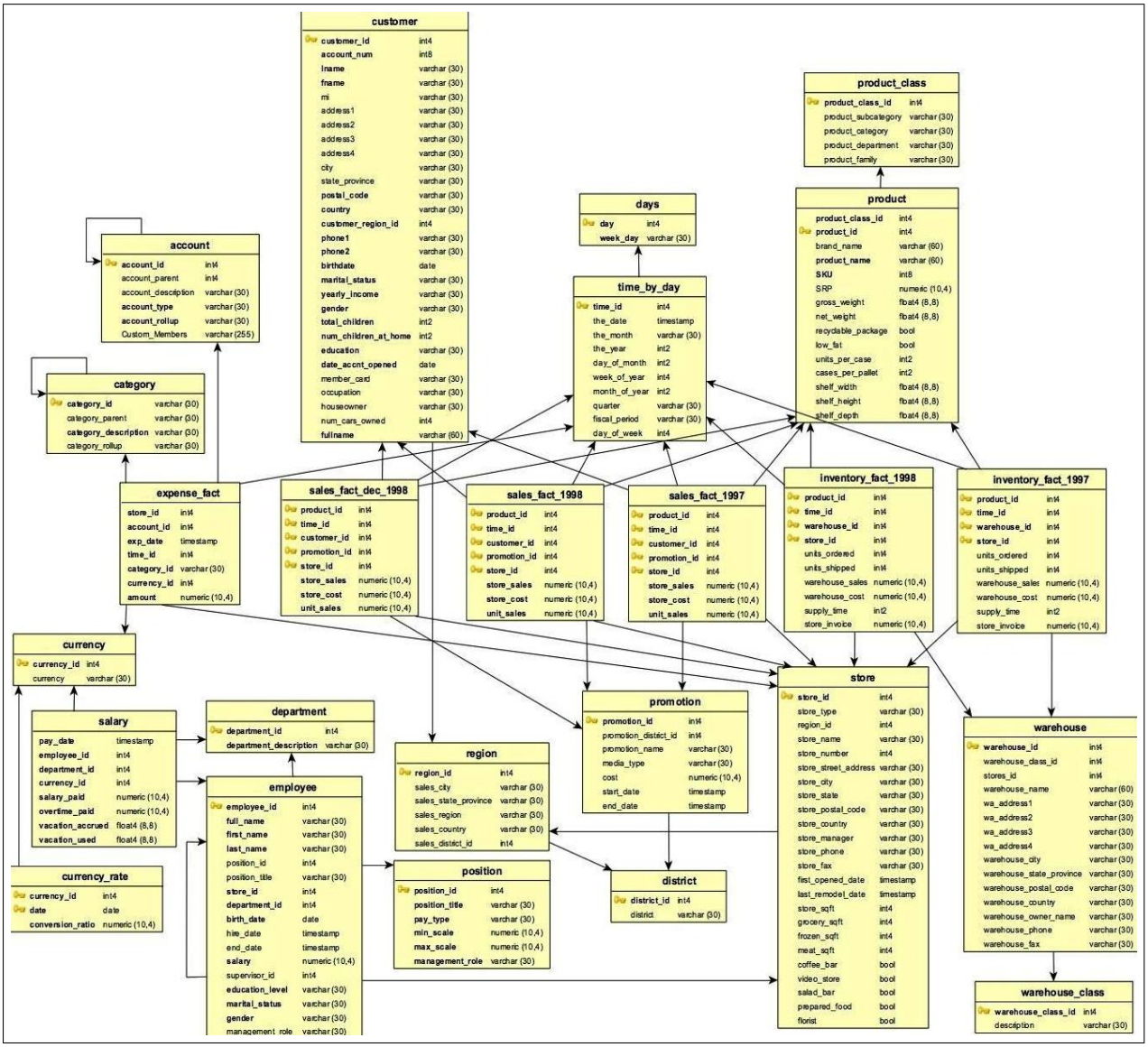


FIGURE B.1 – Schéma de la base de données *FoodMart*

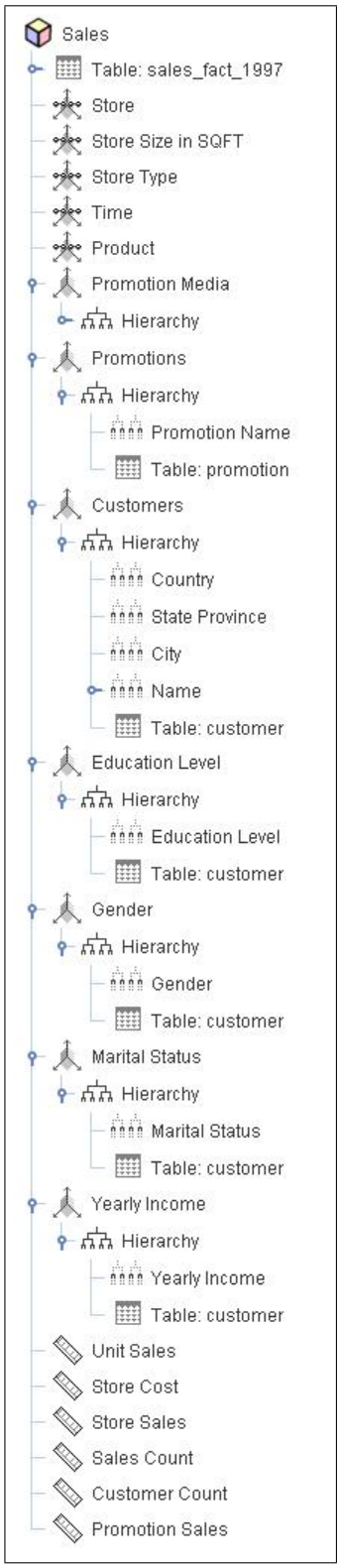


FIGURE B.2 – Schéma de la table de fait *Sales*

Elsa NEGRE

Exploration collaborative de cubes de données

Résumé

Les entrepôts de données stockent de gros volumes de données multidimensionnelles, consolidées et historisées dans le but d'être explorées et analysées par différents utilisateurs. L'exploration de données est un processus de recherche d'informations pertinentes au sein d'un ensemble de données. Dans le cadre de nos travaux, l'ensemble de données à explorer est un cube de données qui est un extrait de l'entrepôt de données que les utilisateurs interrogent en lançant des séquences de requêtes *OLAP* (*On-Line Analytical Processing*). Cependant, cette masse d'informations à explorer peut être très importante et variée, il est donc nécessaire d'aider l'utilisateur à y faire face en le guidant dans son exploration du cube de données afin qu'il trouve des informations pertinentes.
Le travail présenté dans cette thèse a pour objectif de proposer des recommandations, sous forme de requêtes *OLAP*, à un utilisateur interrogeant un cube de données. Cette proposition tire parti de ce qu'ont fait les autres utilisateurs lors de leurs précédentes explorations du même cube de données.
Nous commençons par présenter un aperçu du cadre et des techniques utilisés en Recherche d'Informations, Exploration des Usages du Web ou e-commerce. Puis, en nous inspirant de ce cadre, nous présentons un état de l'art sur l'aide à l'exploration des bases de données (relationnelles et multidimensionnelles). Cela nous permet de dégager des axes de travail dans le contexte des bases de données multidimensionnelles.
Par la suite, nous proposons donc un cadre générique de génération de recommandations, générique dans le sens où les trois étapes du processus sont paramétrables. Ainsi, à partir d'un ensemble de séquences de requêtes, correspondant aux explorations du cube de données faites précédemment par différents utilisateurs, et de la séquence de requêtes de l'utilisateur courant, notre cadre propose un ensemble de requêtes pouvant faire suite à la séquence de requêtes courante. Puis, diverses instanciations de ce cadre sont proposées.
Nous présentons ensuite un prototype écrit en *Java*. Il permet à un utilisateur de spécifier sa séquence de requêtes courante et lui renvoie un ensemble de recommandations. Ce prototype nous permet de valider notre approche et d'en vérifier l'efficacité avec un série d'expérimentations.
Finalement, afin d'améliorer cette aide collaborative à l'exploration de cubes de données et de permettre, notamment, le partage de requêtes, la navigation au sein des requêtes posées sur le cube de données, ou encore de les annoter, nous proposons un cadre d'organisation de requêtes. Ainsi, une instanciation adaptée à la gestion des recommandations est présentée.

Mots clés : Cubes de données, Exploration collaborative, Recommandations, Requêtes *OLAP*.

Abstract

Data warehouses store large volumes of consolidated and historized multidimensional data to be explored and analysed by various users. The data exploration is a process of searching relevant information in a dataset.

In this thesis, the dataset to explore is a data cube which is an extract of the data warehouse that users query by launching sequences of *OLAP* (*On-Line Analytical Processing*) queries. However, this volume of information can be very large and diversified, it is thus necessary to help the user to face this problem by guiding him/her in his/her data cube exploration in order to find relevant information.

The present work aims to propose recommendations, as *OLAP* queries, to a user querying a data cube. This proposal benefits from what the other users did during their previous explorations of the same data cube.
We start by presenting an overview of the used framework and techniques in Information Retrieval, Web Usage Mining or e-commerce. Then, inspired by this framework, we present a state of the art on collaborative assistance for data exploration in (relationnal and multidimensional) databases. It enables us to release work axes in the context of multidimensional databases.
Thereafter, we propose thus a generic framework to generate recommendations, generic in the sense that the three steps of the process are customizable. Thus, given a set of sequences of queries, corresponding to the previous explorations of various users, and given the sequence of queries of the current user, our framework proposes a set of queries as recommendations following his/her sequence. Then, various instantiations of our framework are proposed.
Then, we present a *Java* prototype allowing a user to specify his/her current sequence of queries and it returns a set of recommendations. This prototype validates our approach and its effectiveness thanks to an experimentations collection.
Finally, in order to improve this data cube exploration collaborative assistance and, in particular, to share, navigate or annotate the launched queries, we propose a framework to manage queries. Thus, an instantiation to manage recommendations is presented. **Keywords :** Data Cubes, Collaborative exploration, Recommendations, *OLAP* queries.