Organisation de log de requêtes OLAP sous forme de site web

Sonia Colas, Patrick Marcel, Elsa Negre

Université François Rabelais Tours, Laboratoire d'Informatique, France
{prénom.nom}@univ-tours.fr

Résumé. Vu comme une simple collection de requêtes, un log de requêtes d'un serveur OLAP est une structure peu exploitable. Dans cet article, nous proposons d'organiser un log de requêtes d'un serveur OLAP sous la forme d'un site web. Cela a plusieurs avantages, comme la compréhension rapide de ce qui a été fait lors des sessions d'analyse précédentes ou comme l'aide aux futures sessions d'analyse. La technique utilisée a fait ses preuves pour la réorganisation de sites web. Le log est transformé en un graphe pondéré de requêtes, les poids reflétant une similarité entre requêtes. Cette similarité est calculée en tenant compte à la fois des faits interrogés et de la présence ou non des deux requêtes au sein de la même session. Ces poids sont ensuite ajustés par une heuristique utilisant des fourmis artificielles pour trouver un équilibre entre ces deux facteurs. Enfin le log est présenté sous la forme d'un site web dont la structure est l'arbre couvrant minimal de ce graphe. Cette technique est implémentée en java pour le traitement de logs de requêtes MDX. Des expériences ont été conduites sur des logs synthétiques pour tester la qualité des solutions obtenues selon des critères utilisés dans le domaine du web et des interfaces homme-machine.

1 Introduction

Le travail de Khoussainova et al. (2009) a récemment mis en avant une lacune importante des SGBD actuels, à savoir le besoin d'outil de gestion de requêtes. De tels outils permettraient notamment de faciliter l'exploration et l'analyse de gros volumes de données dans un environnement multi-utilisateurs, par exemple en organisant les logs de requêtes pour mettre rapidement en évidence les analyses déjà conduites, de manière à aider les explorations ultérieures.

Le présent travail constitue un pas dans cette direction, dans le contexte de l'analyse OLAP d'entrepôts de données. Nous proposons ainsi d'organiser les logs de requêtes d'un serveur OLAP sous la forme de sites web. Nous adaptons une technique qui a fait ses preuves pour la réorganisation de sites web (Colas et al. (2008)). Cette adaptation repose sur une similarité entre requêtes basée sur le TF/IDF. Plus précisément, le log est d'abord vu comme un graphe pondéré de requêtes, les poids reflétant la similarité entre requêtes. Cette similarité est calculée en tenant compte à la fois des faits interrogés et du fait que les requêtes se trouvent au sein de la même session. Ces poids sont ensuite ajustés par une heuristique utilisant des fourmis artificielles pour trouver un équilibre entre ces deux facteurs. Enfin le log est présenté sous la forme d'un site web dont la structure est l'arbre couvrant minimal de ce graphe.

En résumé, nos contributions sont les suivantes :

– une méthode pour l'organisation de requêtes OLAP,

– l'implémentation de cette méthode pour organiser des logs de requêtes MDX (Microsoft Corporation (1998)),
– des expériences conduites sur des logs synthétiques pour tester la qualité des solutions obtenues, selon des critères utilisés dans le domaine du web et des interfaces homme-machine.

Le reste de cet article est organisé comme suit. La section suivante discute des travaux connexes et motive notre approche par l'exemple. Les Sections 3 et 4 présentent le cadre formel. La Section 5 présente l'implémentation et les tests réalisés. Enfin, la Section 6 conclut l'article et situe le travail dans la perspective plus large des outils de gestion de requêtes.

2 Motivations

Dans cette section, nous motivons notre approche en utilisant un exemple qui sera récurrent dans la suite de l'article. Nous commençons par évoquer les travaux connexes.

2.1 Travaux connexes

2.1.1 Structures de sites web

La qualité d'un site web se définit par son ergonomie et par sa capacité à répondre aux attentes et besoins réels des internautes à travers un contenu pertinent et de qualité. Les critères ergonomiques proviennent d'études empiriques ou de pratiques courantes (Leulier et al. (1998)), auxquels peuvent s'ajouter d'autres règles générales comme réduire le temps de chargement de la page ou proposer un moteur de recherche performant. Les mesures associées sont évaluées grâce à des modèles qui simulent la navigation d'un internaute (Miller (2004)) ou avec des tests d'usages (Bobillier-Chaumon et Sandoz-Guermond (2006)).

L'architecture d'un site web joue un rôle décisif pour son utilisabilité. Des indicateurs (Nogier (2005)) permettent d'éviter que les internautes se trouvent désorientés. Ils se basent sur ceux concernant les menus dans les applications (Landauer et Nachbar (1985)). Il s'agit, par exemple, d'optimiser le nombre de sous-chemins à chaque niveau (10 pour les novices et jusqu'à 20 items conseillés pour les menus, et 8 sous-rubriques maximum conseillées pour les sites), de limiter la profondeur à 3 ou 4 niveaux ou encore de proposer une structure sous forme d'arbre plutôt que de graphe pour la structure d'un site.

Pour aider le guidage, le plan du site peut être affiché sur une page réservée du site, mais encore beaucoup de sites n'en disposent pas. Des outils de génération de plan de site, reproduisant la structure réelle du site sont proposés aux webmestres. Dans Colas et al. (2008) a été proposé un générateur de plan, avec une structure thématique, qui s'adapte aux capacités de mémorisation des internautes non-voyants en leur proposant de limiter le nombre de sous-arbres à chaque niveau du plan. Ce générateur de plan utilise un algorithme de fourmis artificielles (Reimann et Laumanns (2004)), pour résoudre ce problème de transformation de graphe en arbre. Les fourmis artificielles ont pour rôle de déterminer des regroupements optimums de nœuds du graphe, puis le sous-problème restant consiste à trouver l'arbre de poids minimum, ce qui peut être réalisé par les algorithmes classiques de Kruskal (1956) ou de Prim (1957).

2.1.2 Outil de gestion de requêtes de bases de données

Ainsi que le rapporte Khoussainova et al. (2009), il n'existe pas véritablement d'outil de gestion de requêtes dans les SGBD actuels. Au mieux, des outils existants permettent de collecter

et d'analyser la charge du SGBD (DB2 query patroller, Oracle SQL management studio) mais ces fonctionnalités sont essentiellement orientées vers l'optimisation de requêtes ou le tuning physique.

A notre connaissance, notre travail est le premier à proposer une méthode d'organisation de requêtes OLAP sous la forme d'un site web. Il peut être vu comme la suite de Choong et al. (2007) où est proposée une organisation des requêtes OLAP sous forme de graphe, sans toutefois indiquer comment cette organisation était obtenue. Dans ce présent papier, nous proposons une méthode pour obtenir une organisation particulière de requêtes OLAP sous forme de site web.

La fouille de logs de requêtes dans les bases de données est un domaine en plein essor. Nous citerons, par exemple, le travail de Aouiche et al. (2006) pour traiter le problème du choix des vues à matérialiser, le travail de Sapia (2000) pour le préchargement de données, ou encore celui de Huang et al. (2006) pour retrouver les sessions utilisateur. Dans le domaine de l'aide à l'exploration d'entrepôts de données, Negre (2009) et Giacometti et al. (2009) utilisent les logs pour recommander des requêtes par des techniques issues du filtrage collaboratif. Ainsi, les travaux existants proposent une approche automatique (préchargement, recommandations, ...) tandis que notre approche vise à organiser ce qui a déjà été fait afin que l'utilisateur navigue sans peine.

2.2 Le problème

Considérons un log L de requêtes OLAP où chaque session est une simple séquence de requêtes commençant à l'indice 0. Ce log contient trois sessions de 2, 5 et 3 requêtes qui, par la suite sont nommées de q_1 à q_{10}. Ce log est le résultat des sessions conduites par différents utilisateurs pour analyser le cube de données $Sales$ de la base de données $Foodmart$, fournie avec le moteur OLAP Mondrian (Pentaho Corporation (2009)).

Supposons maintenant que d'autres utilisateurs aient les besoins suivants :

- rendre compte de ce qui a été fait avec le cube au cours des sessions antérieures,
- trouver une analyse sur un sujet particulier, cette analyse pouvant avoir été conduite sur plusieurs sessions,
- savoir si une analyse particulière n'a pas déjà été conduite,
- savoir si des sessions peuvent être connectées entre elles parce qu'elles portent sur des sujets proches.

A partir du seul log, satisfaire ces besoins est fastidieux, le log étant au mieux une suite de séquences de texte de requêtes, sans d'autres liens entre elles que le fait d'appartenir à la même session.

Considérons maintenant la Figure 4. Le log est présenté comme un site web de requêtes dont la Figure 4 présente la structure et où la Figure 1 montre la page web correspondant à la requête q_1.

Ainsi, l'utilisateur peut naviguer ce site et voir par exemple que :

- suivre un des chemins correspond à une analyse des ventes en Californie via les requêtes q_2 ou q_6,
- une analyse portant sur l'année 1998 n'a pas déjà été conduite, par exemple, la suite de requêtes q_3, q_9, q_{10} qui traite des ventes de boissons en fonction du genre en 1997, pourrait être réutilisée pour l'année 1998.

FIG. 1 – *Exemple de page web pour la requête* q_1.

2.3 Présentation intuitive de l'approche

Etant donné un log de requêtes, notre méthode repose sur les trois étapes suivantes : 1) Construction du graphe, 2) Construction de l'arbre et 3) Construction du site web.

Nous illustrons ci-dessous chacune de ces étapes à partir du log L. Avant cela, nous notons que si le log est trop volumineux, sa taille pourra être réduite en effectuant un clustering, comme cela est par exemple fait dans Negre (2009), et en construisant un site à partir uniquement des requêtes représentant les clusters.

Construction du graphe de requêtes A partir du log de requêtes, un graphe pondéré de requêtes, éventuellement non-connexe, est construit. Plusieurs méthodes de construction sont envisagées selon la prise en compte des sessions et du séquencement des requêtes.

Construction de l'arbre de recouvrement Une heuristique à base de fourmis artificielles est alors utilisée pour ajuster ces poids afin de rapprocher les requêtes de sessions différentes. A partir de ce graphe, un arbre de recouvrement minimal est calculé. La racine de cet arbre est la requête en moyenne la plus proche des autres requêtes du log.

Création des pages et des hyperliens L'arbre de recouvrement donne la structure du site, à savoir un ensemble de pages reliées entre elles à raison d'une requête par page (nœuds de l'arbre). Les hyperliens entre requêtes adjacentes q et q' sont ajoutés de manière à représenter les membres à ajouter et à supprimer pour passer de la requête q à la requête q'. A chaque nœud est également associé une description du sous-arbre dont ce nœud est la racine contenant la liste des membres apparaissant dans le sous-arbre.

Les deux sections suivantes détaillent les deux étapes de construction de l'arbre.

3 Construction du graphe de requêtes

Dans cette section, nous détaillons comment construire un graphe à partir d'un log de requêtes. En effet, l'ensemble des requêtes du fichier log peut être représenté sous la forme d'un

graphe où chaque nœud représente une requête et où les arêtes du graphe sont pondérées selon la distance entre les nœuds, i.e. les requêtes. Dans la Section 3.1, nous introduisons les concepts utilisés dans cet article. La Section 3.2 explique comment relier les nœuds du graphe et la Section 3.3, comment pondérer ces arcs.

3.1 Requête, session et log

Un cube n-dimensionnel $C = \langle D_1, ..., D_n, F \rangle$ est défini comme les $n + 1$ instances de relation d'un schéma en étoile, avec une instance de relation pour chacune des n dimensions et une instance de relation pour la table de fait.

Comme dans Negre (2009), nous considérons des requêtes MDX dont nous extrayons les membres de la manière suivante : Soit un cube n-dimensionnel $C = \langle D_1, ..., D_n, F \rangle$ et soit M_i un ensemble de membres pris dans la dimension D_i, $\forall i \in [1, n]$, nous considérons l'ensemble des membres d'une requête sur C comme étant l'ensemble de membres $M_1 \cup ... \cup M_n$.

Exemple 3.1 *Considérons la requête q_4 du log L présenté Section 2.2, sur le cube $Sales$ contenant 13 dimensions, son expression MDX est :*

```
SELECT  {[Store].[All Stores].Children}                    ON ROWS,
        Crossjoin({[Yearly Income].[All Yearly Incomes]},
        {[Gender].[All Gender].Children})                  ON COLUMNS
FROM    [Sales]
WHERE   [Measures].[Unit Sales] ;
```

Sachant que, toute dimension non détaillée correspond à la sélection du membre par défaut de la dimension, par exemple 1997 *pour la dimension $Time$ et $AllProduct$ pour la dimension $Product$ et sachant que les membres du niveau $[Store].[AllStores].Children \in \{Canada, USA, Mexico\}$ et les membres du niveau $[Gender].[AllGender].Children \in \{F, M\}$, l'ensemble des membres correspondant à la requête q_4 est :*
{UnitSales, Canada, USA, Mexico, AllSize, AllStoreType, 1997, AllProduct, AllMedia, AllPromo, AllCustomers, AllEducLevel, F, M, AllMaritalStatus, AllIncome}

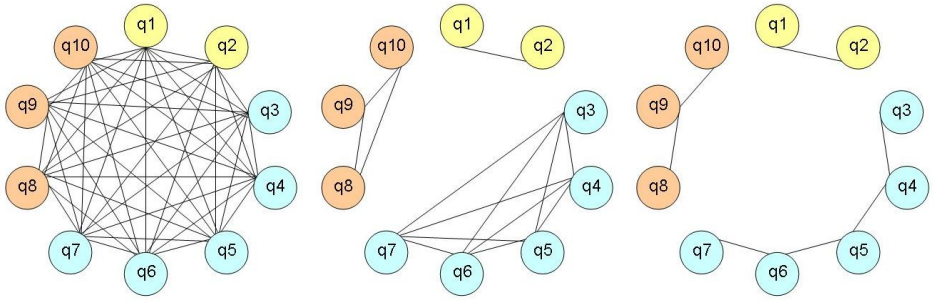
Une séquence finie de requêtes est ce que nous appelons une session d'analyse. Enfin, un log est un ensemble de sessions.

3.2 Adjacence

Afin de construire le graphe représentatif du fichier log, plusieurs moyens pour relier les nœuds sont possibles, prenant plus ou moins en compte le séquencement des requêtes dans les sessions de requêtes du log. Notons que si deux requêtes sont équivalentes (en terme de membres extraits de chaque requête), un seul nœud les représente.

3.2.1 Non prise en compte des sessions de requêtes

Une première approche consiste à voir le log comme un ensemble de requêtes indépendantes les unes des autres et donc à ne pas prendre en compte le séquencement des requêtes dans les sessions, i.e. à considérer que toute requête peut être rapprochée de toute autre. Le graphe représentatif du log sera alors complet. Le log L présenté Section 2.2 peut être représenté via le graphe de la Figure 2 (a).



(a) Non prise en compte du séquencement et des sessions.

(b) Prise en compte des sessions mais pas du séquencement.

(c) Prise en compte du séquencement et des sessions.

FIG. 2 – *Graphes représentatifs du log selon différents types d'adjacence.*

3.2.2 Prise en compte des sessions mais pas du séquencement de requêtes

Une seconde approche est de considérer que l'ordre dans lequel les requêtes ont été lancées dans une session n'est pas important. Dans ce cas, le fichier log est vu comme un ensemble de sessions contenant des requêtes non ordonnées, les nœuds représentants des requêtes issues d'une même session sont reliés entre eux. Cela revient à réaliser des graphes complets sur les requêtes d'une même session. Le log L présenté Section 2.2 peut être représenté via le graphe de la Figure 2 (b).

3.2.3 Prise en compte du séquencement de requêtes et des sessions

Une dernière approche est de considérer que l'ordre dans lequel les requêtes ont été lancées au sein d'une session est important. Dans ce cas, une succession ou une précédence existe entre deux requêtes dans au moins une session du fichier log et les nœuds représentants les deux requêtes sont liés. Ainsi, si une requête q_i précède ou succède à une requête q_j dans le log, une arête est placée entre les nœuds q_i et q_j dans le graphe. Chaque sous-graphe correspondant à une session est connexe. Le log L présenté Section 2.2 peut être représenté via le graphe de la Figure 2 (c).

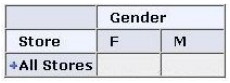
3.3 Similarité

En utilisant une des trois techniques décrites précédemment, nous disposons d'un graphe représentatif du log où chaque nœud représente une requête. Nous devons pondérer les arêtes du graphe en fonction de la distance entre les nœuds, i.e. les requêtes.

Nous proposons d'utiliser le TF/IDF[1] (Salton et McGill (1983)) qui est classiquement utilisé en Recherche d'Information pour estimer l'importance d'un mot par rapport au document qui le contient, en tenant compte du poids de ce mot dans le corpus complet. Dans notre cas, une requête $OLAP$ est vue comme un ensemble de membres, nous pouvons donc évaluer l'importance d'un membre dans une requête. Pour chaque requête i, on calcule un vecteur $tfidf_i$, où chaque composante est notée $tfidf_{i,j}$ avec j un membre de la requête i. La taille du vecteur est fonction de l'ensemble des membres présents dans le log de requêtes.

[1] Soient i une requête et j un membre de i, $tfidf_{i,j} = \frac{tf_{i,j}}{df_j}$ où $tf_{i,j}$ est la fréquence d'apparition du membre j dans la requête i et df_j est la fréquence d'apparition du membre j dans l'ensemble des requêtes du fichier log.

Exemple 3.2 *Considérons les requêtes q_3 et q_8 du log L présenté Section 2.2, le tableau suivant détaille le calcul de la distance entre q_3 et q_8 au sens du TF/IDF.*

<table>
<tr><td>q_3
 
<table><tr><td></td><td colspan="2">Gender</td></tr><tr><td>Store</td><td>F</td><td>M</td></tr><tr><td>+All Stores</td><td></td><td></td></tr></table></td><td>Ensemble des membres :
{UnitSales, AllStore, AllSize,AllStoreType, 1997, AllProduct, AllMedia, AllPromo, AllCustomers, AllEducLevel, F, M, AllMaritalStatus, AllIncome}
$X = tfidf_{q_3}$
$= (\frac{1}{10}, \frac{1}{5}, \frac{1}{10}, \frac{1}{5}, \frac{1}{10}, \frac{1}{7}, 0, 0, 0, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, 0, \frac{1}{5}, \frac{1}{5}, \frac{1}{10}, \frac{1}{10}, 0, \ldots, 0)$</td></tr>
<tr><td>q_8
 
<table><tr><td></td><td colspan="3">Product</td></tr><tr><td>Time</td><td>+Drink</td><td>+Food</td><td>+Non-Consumable</td></tr><tr><td>+1997</td><td></td><td></td><td></td></tr></table></td><td>Ensemble des membres :
{UnitSales, AllStore, AllSize, AllStoreType, 1997, Food, Drink, Non-Consumable, AllMedia, AllPromo, AllCustomers, AllEducLevel, AllGender, AllMaritalStatus, AllIncome}
$Y = tfidf_{q_8}$
$= (\frac{1}{10}, \frac{1}{5}, \frac{1}{10}, \frac{1}{5}, \frac{1}{10}, 0, 1, 1, 1, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{10}, \frac{1}{5}, 0, 0, \frac{1}{10}, \frac{1}{10}, 0, \ldots, 0)$</td></tr>
<tr><td colspan="2">$distance(q_3, q_8) = 1 - cos(X, Y)$
≈ 0.817</td></tr>
</table>

Finalement, les arêtes du graphe sont pondérées avec la valeur de la distance, au sens du TF/IDF, entre les deux nœuds, i.e. requêtes, qu'elles relient.

3.4 Racine

A partir du graphe représentant le log de requêtes, nous souhaitons construire un site web dont la structure sera un arbre calculé à partir du graphe. Un arbre possède un nœud racine qui représentera le point d'entrée dans le log. Nous considérons que le nœud racine est une requête existante dans le log. Notre choix s'est donc porté vers la requête médoïde. Il s'agit de la requête pour laquelle la distance moyenne à toutes les autres requêtes du log est minimale. Si plusieurs requêtes peuvent être le médoïde, c'est la première requête trouvée qui devient la racine de l'arbre.

Hormis le graphe représentant le log sans prendre en compte les sessions (comme la Figure 2 (a)), les graphes générés, à savoir celui représentant le log en prenant en compte les sessions mais pas le séquencement (comme la Figure 2 (b)) et celui prenant en compte le séquencement (comme la Figure 2 (c)), ne sont pas forcément connexes. Deux possibilités sont envisagées pour rendre connexe le graphe de départ :

- relier le médoïde à la première requête de chaque session (dans le cas où le séquencement de requêtes est pris en compte (comme la Figure 2 (c))),
- relier le médoïde à la requête la plus proche de chaque session.

Exemple 3.3 *Dans le log L présenté Section 2.2, le médoïde q_1 est relié à la première requête de chaque session, le graphe de la Figure 2 (c) est modifié pour ajouter les liens (q_1, q_3) et (q_1, q_8). Le graphe devient celui de la Figure 3.*

4 Construction de l'arbre de recouvrement

Le fichier log est représenté sous forme de graphe connexe où chaque nœud est une requête. Il s'agit de construire l'arbre couvrant de poids minimum du graphe représentant le log de requêtes.

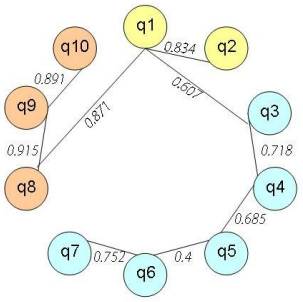


FIG. 3 – *Graphe pondéré représentatif du log prenant en compte le séquencement et les sessions où le médoïde est relié à la première requête de chaque session.*

4.1 Prim

A partir du graphe du fichier log, le problème consistant à obtenir un arbre couvrant de poids minimum (ACM) peut être résolu par l'algorithme de Prim (1957). Cet algorithme consiste à faire croître un arbre à partir d'un nœud racine (le premier sommet marqué). Dès lors, à chaque itération, il connecte un sommet non marqué à un sommet marqué (c'est-à-dire déjà placé dans l'arbre en construction). L'arbre va ainsi grossir jusqu'à ce qu'il couvre tous les sommets du graphe. A chaque étape, le nouveau sommet y à marquer est tel que y est adjacent à un sommet x marqué et que l'arête (x, y) est celle de plus faible poids. Chaque augmentation se fait donc de la manière la plus économique possible.

Cependant, avec l'algorithme de Prim, seuls les liens du graphe serviront dans la construction de l'ACM. Par exemple avec le graphe de la Figure 3, l'ACM est construit uniquement à partir des 9 arêtes existantes et de leurs poids. Ainsi pour les graphes prenant en compte les sessions (Figure 2 (b) et (c)), aucun rapprochement entre requêtes de sessions différentes ne sera établi par le simple fait d'un coût économique. Afin de permettre l'établissement de nouveaux liens non initialement présents mais permettant l'obtention d'un ACM de coût inférieur, nous allons ajouter une pondération supplémentaire à chaque chemin permettant de représenter la probabilité d'utilisation pour la construction de l'ACM. Pour ce faire nous avons utilisé un algorithme de fourmis artificielles (Reimann et Laumanns (2004); Colas et al. (2008)).

4.2 Ajustement des chemins par les fourmis artificielles

Dans leur milieu naturel les fourmis sont inconsciemment capables de trouver le plus court chemin entre le nid et une source de nourriture. Pour cela, elles déposent des phéromones (substances volatiles) sur le terrain, ce qui incite les autres fourmis à suivre le même chemin (et renforcent à leur tour le chemin). Cette caractéristique a été appliquée pour la recherche du plus court chemin dans un graphe (Colorni et al. (1991)), puis de nombreuses variantes ont été développées (Dorigo et al. (1996)). Désormais l'heuristique ACO (*Ant Colony Optimization*) regroupe toutes les variantes issues de cette rencontre fourmis-phéromones-optimisation. Le principe général de l'algorithme ACO utilisé pour la génération de plan par les fourmis artificielles est donné par l'algorithme 1 (Colas et al. (2008)).

Dans cet algorithme, les fourmis réalisent des circuits, représentant des regroupements de sources de nourriture (i.e. requêtes) proches les unes des autres, sur lesquels l'algorithme de Prim est utilisé. Le nombre de fourmis dans cet algorithme est égal au nombre de requêtes. La fourmi doit opérer des choix concernant le chemin à emprunter, choix influencés par la distance séparant les sources de nourriture (la distance entre requêtes) et par la quantité de phéromones

Algorithme 1 Génération de plan avec des fourmis artificielles

```
1: Initialiser les valeurs des phéromones
2: Répéter
3:    Pour tout les fourmis  Faire
4:       Générer les circuits
5:       Pour tout circuit  Faire
6:          Générer un ACM avec l'algorithme de Prim
7:       Fin pour
8:    Fin pour
9:    Mettre à jour les phéromones
10: Jusqu'à la validation du test d'arrêt
11: Retourner la meilleure solution trouvée depuis le début
```

sur les chemins. Dans le but de favoriser les liens mis en evidence dans le graphe de départ, la matrice de phéromones est initialisée avec la matrice d'adjacence du graphe initial, et, pour ne pas exclure les chemins inexistants au départ, une quantité faible mais non nulle de phéromones leur est attribué. Lors de la mise à jour des phéromones, la quantité de phéromones augmente pour les liens appartenant à la meilleure solution et diminue dans le cas contraire. Les fourmis étant attirées vers les chemins contenant la plus grande quantité de phéromones, un écart se creuse au niveau de la quantité de phéromones sur les liens, selon qu'ils appartiennent à la solution ou non. Le test d'arrêt est vérifié lorsque la quantité de phéromones sur les chemins appartenant à la meilleure solution est classiquement supérieure à 0.9, et inférieure à 0.1 pour ceux n'appartenant pas à la meilleure solution.

Exemple 4.1 *Dans le log L présenté Section 2.2, avec les choix menant au graphe de la Figure 3, l'arbre construit via l'algorithme Prim couplé aux fourmis artificielles est présenté Figure 4. Trois nouveaux liens ont été créés ((q_3, q_9), (q_1, q_5) et (q_1, q_6)) du fait de la proximité entre requêtes (par exemple sélection des membres "'F"' et "'M"' de la dimension Gender pour les requêtes q_3 et q_9). Ces nouveaux chemins ont donc vu leur quantité de phéromones augmenter au fur et à mesure des itérations, jusqu'à devenir des chemins incontournables pour atteindre l'ACM.*

5 Expérimentations

Dans cette section, nous présentons les expériences que nous avons menées pour évaluer les résultats produits. Les logs utilisés sont des données synthétiques produites grâce à notre propre générateur. Le générateur de logs et le prototype de construction de site sont développés en Java utilisant JRE 1.6.0_13. Tous les tests ont été réalisés avec un Xeon E5430 disposant de 32GB de RAM sous Linux CentOS5.

5.1 Génération des logs

Notre générateur de logs produit des ensembles de sessions de requêtes MDX sur la base de données FoodMart fournie avec le moteur OLAP Mondrian. Son principe est détaillé dans Negre (2009), nous en donnons ici les principales caractéristiques.

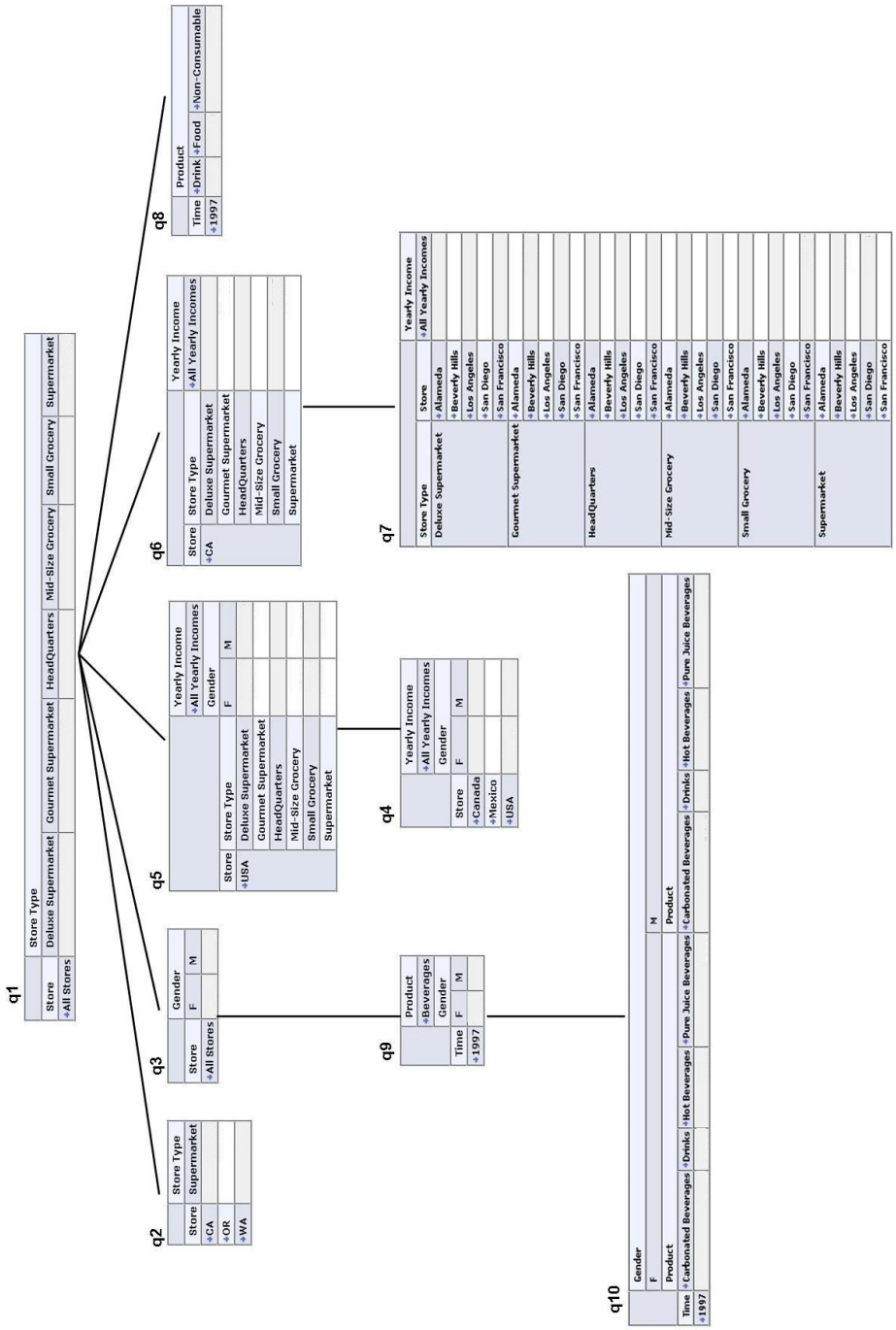


FIG. 4 – *Arbre couvrant de poids minimal fournissant la structure du site.*

Chaque session est générée en tentant de simuler une analyse pilotée par la découverte (Sarawagi (1999)). Pour une session, chaque requête est générée en fonction de la requête qui la précède dans la session, en appliquant au résultat de cette requête un des opérateurs proposé dans Sarawagi (1999) ou Sathe et Sarawagi (2001). Ces opérateurs ont, en effet, pour but d'expliquer par un ensemble de cellules du cube des mesures inattendues constatées dans le résultat d'une requête. Ainsi, en transformant le résultat d'un tel opérateur en requête, notre générateur fabrique une requête pouvant être vue comme tentant d'expliquer le résultat de la requête précédente.

La taille des logs obtenus est contrôlée par deux paramètres indiquant respectivement le nombre de sessions voulu et la taille maximale des sessions. La densité est contrôlée par un paramètre indiquant combien de dimensions peuvent être utilisées pour naviguer dans le cube (plus ce nombre est faible, moins il y a de chance d'explorer des parties du cube très différentes et plus le log sera dense).

Les tests ont été réalisés sur 3 logs de 25 sessions comportant entre 2 et 10 requêtes chacune, soit des logs de 95, 92 et 71 requêtes. Le nombre maximum de dimensions pour naviguer dans le cube étant fixé pour chacun des logs à 13 (densité faible), 9 (densité moyenne), et 5 (densité forte), la base de données FoodMart comportant 13 dimensions.

5.2 Les tests

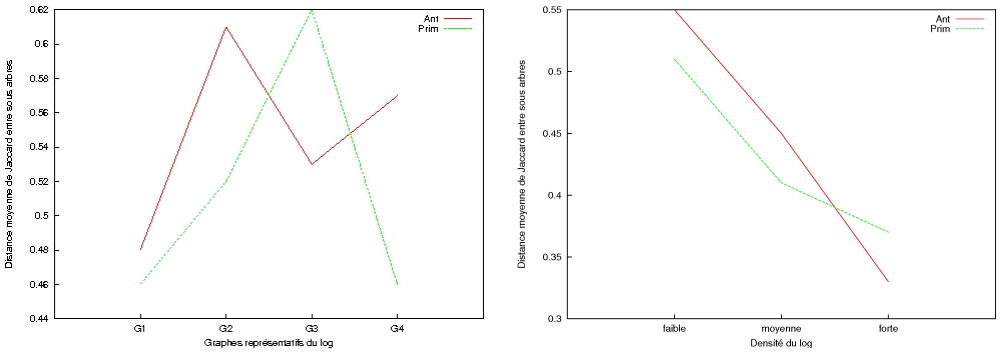
Les tests mis en place permettent d'extraire des indicateurs concernant les coûts (coût de l'arbre, i.e. somme des poids des arêtes de l'arbre, coût des liens ajoutés par la méthode des fourmis par rapport au graphe de requêtes), et la forme de l'arbre obtenu (hauteur, largeur, nombre de sous-chemins à chaque niveau, séparation des sous-arbres).

Le nombre de sous-chemins à chaque niveau est le nombre maximum de sous-arbres pour un nœud de l'arbre. Cet indicateur est directement comparable avec les recommandations dans le domaine du web concernant le nombre maximum de sous-items dans les menus ou de sous-rubriques dans les sites web. Pour évaluer la séparation entre les sous-arbres, nous avons utilisé la distance de Jaccard (l'indice de Jaccard étant le rapport entre la cardinalité de l'intersection des ensembles et la cardinalité de leur union) en prenant pour chaque sous-arbre l'union des membres des requêtes composant le sous-arbre.

5.2.1 La séparation des sous-arbres

La Figure 5 (a) présente, pour le log de faible densité, la distance moyenne de Jaccard entre sous-arbres en fonction des graphes de requêtes. Le graphe "G1" (Figure 2 (a)) correspond au graphe ne prenant pas en compte le séquencement et les sessions ; "G2" et "G3" (Figure 2 (c)) correspondent aux graphes prenant en compte le séquencement et les sessions avec le médoïde relié respectivement aux premières requêtes de chaque session ("G2") et à la plus proche requête de chaque session ("G3") ; et "G4" (Figure 2 (b)) correspond au graphe prenant en compte les sessions mais pas le séquencement (avec le médoïde relié à la plus proche requête de chaque session). Il apparaît que le graphe "G1" donne de moins bons résultats en terme de séparation de sous-arbres que les graphes prenant en compte les sessions.

Nous observons en général une meilleure séparation des sous-arbres avec la méthode des fourmis artificielles. La Figure 5 (b) présente la distance moyenne de Jaccard entre sous-arbres de niveau 1 en fonction de la densité du log. Pour chaque log, les valeurs correspondent aux moyennes des valeurs obtenues pour chacun des quatre graphes. En moyenne, nous observons une meilleure séparation avec la méthode des fourmis par rapport à l'algorithme de Prim pour



(a) (b)

FIG. 5 – *Distance moyenne de Jaccard entre sous-arbres en fonction des graphes de requêtes du log de faible densité (a), et en fonction de la densité des logs (moyenne des valeurs pour les 4 graphes) (b).*

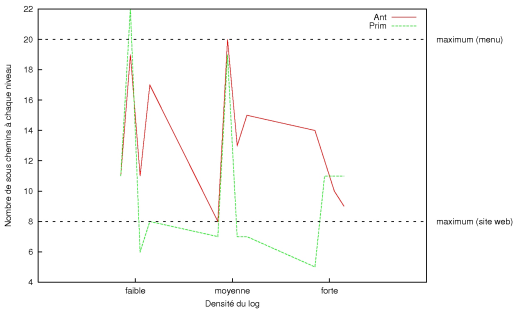


FIG. 6 – *Nombre de sous-chemins à chaque niveau de l'arbre en fonction de la densité des logs.*

les logs de faible densité, et une distance plus faible pour les logs de forte densité (requêtes en moyenne plus proches les unes des autres).

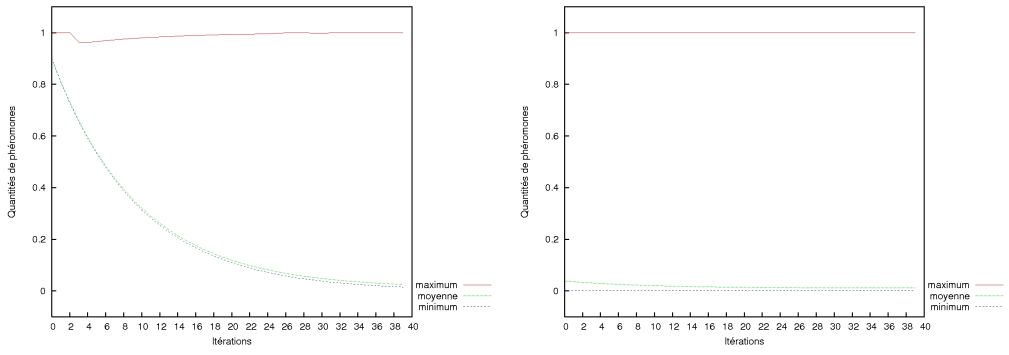
5.2.2 Le nombre de sous-chemins

La Figure 6 présente le nombre de sous-chemins à chaque niveau de l'arbre en fonction de la densité du log. Pour chaque log, 4 valeurs sont représentées correspondant aux nombres de sous-chemins pour chacun des graphes de requêtes "G1" à "G4".

Nous pouvons remarquer que les recommandations en terme de nombre maximum de sous-chemins à chaque niveau (maximum 20 pour les menus, et 8 pour les sites web), ne sont pas toujours respectées, celle concernant les sites web ne l'étant quasiment jamais. Afin d'être sûr de respecter ces critères il faudrait résoudre le d-MST (« Degree-Constrained Minimum Spanning Tree ») en fixant par exemple la valeur du degré maximum à 8 pour respecter les recommandations des sites web. Pour résoudre ce problème l'algorithme d-Prim (Narula et Ho (1980)) peut être utilisé au lieu de Prim ; et toujours dans le but d'autoriser dans le d-MST des liens inexistant dans le graphe initial, l'algorithme de fourmis artificielles doit être adapté en limitant le nombre de circuits et en utilisant l'algorithme d-Prim sur les regroupements de requêtes réalisés par les fourmis artificielles.

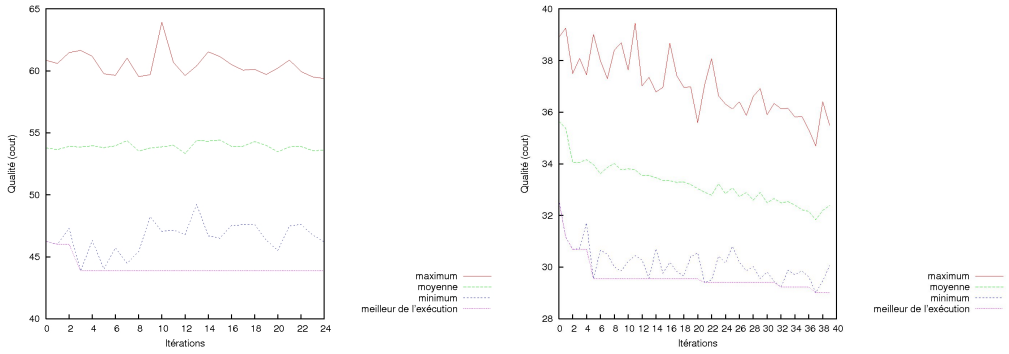
5.2.3 Pondération des chemins par les fourmis

Les Figures 7 et 8 permettent d'illustrer le déroulement de la méthode utilisant une pondération des chemins par les fourmis artificielles. Elles montrent respectivement l'évolution de la valeur des phéromones, et de la qualité des plans pour le log de faible densité avec le graphe de requêtes "G1" (obtenu en ne prenant pas en compte les sessions) (a) puis avec le graphe de requêtes "G4" (obtenu en prenant en compte les sessions mais pas le séquencement) (b). Les courbes obtenues pour les autres logs et graphes de requêtes sont très similaires à celles-ci.



(a) (b)

FIG. 7 – *Valeur des phéromones (max, min, et moyenne sur tous les chemins) pour le log de faible densité, et le graphe des requêtes obtenu en prenant en compte (b) ou non (a) les sessions.*



(a) (b)

FIG. 8 – *Coût de l'arbre (max, min, moyenne, et min depuis le début de l'exécution) pour le log de faible densité, et le graphe des requêtes obtenu en prenant en compte (b) ou non (a) les sessions.*

Sur la Figure 7, nous observons que la quantité moyenne de phéromones sur tous les chemins diminue au fur et à mesure des itérations, la quantité de phéromones étant initialement proche de 1 pour la Figure 7 (a) (tous les liens étant initialement présents) et déjà proche de 0 pour la Figure 7 (b) (peu de liens étant initialement présents). Cette évaporation moyenne des phéromones illustre la suppression de liens entre le graphe de requêtes et l'arbre de recouvrement, mais aussi l'échange entre un lien existant et un lien de plus faible coût choisi par les fourmis.

Sur la Figure 8 (b), la valeur moyenne du coût des solutions décroissante montre qu'au fur et à mesure des itérations les fourmis sont incitées à choisir des chemins (grâce aux phéromones)

qui mèneront à une solution de faible coût. Cela n'est pas visible sur la Figure 8 (a) car les phéromones étant présentes en grande quantité sur tous les liens, les fourmis choisissent des chemins sans critère d'évaluation globale de la solution.

Finalement, les fourmis artificielles sont capables de choisir les liens qui permettront d'obtenir une solution de coût réduit. Elles ont bien pondéré les chemins apportant une amélioration à la solution finale, puisque les coûts des arbres se trouvent inférieurs à ceux des arbres générés uniquement par Prim. De plus, elles apportent une amélioration en terme de séparation des sous-arbres pour les logs de faible densité.

6 Conclusion

Dans cet article nous proposons une technique pour transformer un log de requêtes OLAP en un site web. Cette technique est implémentée et a été testée pour traiter des logs de requêtes MDX. De manière générale, ce travail préliminaire est une contribution à un outil convivial et complet de gestion de requêtes OLAP. Un tel outil aura pour but de rendre plus efficace l'exploration en ligne de cubes de données dans un contexte multi-utilisateurs.

Plusieurs aspects sont envisagés pour la suite de nos travaux, comme la mise en place des deux méthodes permettant de contraindre le nombre de sous-arbres à chaque niveau de l'arbre de recouvrement, ou encore l'utilisation à la place du TF/IDF d'une distance déjà utilisée en OLAP pour calculer la similarité entre requêtes comme la distance de Hausdorff (Negre (2009)) qui, couplée à une distance basée sur le plus chemin entre deux membres permet de prendre en compte les hiérarchies des dimensions. D'autre part, cet algorithme pourrait également être utilisé pour obtenir un plan de sessions, en adoptant une distance entre sessions (par exemple la distance de Levenstein) à la place d'une distance entre requêtes. Le développement futur intégrera aussi une interface web prenant en compte des critères ergonomiques et d'IHM afin de répondre au mieux à notre problématique de réorganisation conviviale de logs de requêtes. Finalement, nous conduirons des tests sur l'analyse de données réelles. A cette fin, un travail est en cours pour l'analyse de 500000 questionnaires de santé fournis par l'IRSA (Institut inter-Régional de la SAnté).

Références

Aouiche, K., P.-E. Jouve, et J. Darmont (2006). Clustering-Based Materialized View Selection in Data Warehouses. In *ADBIS*, pp. 81–95.

Bobillier-Chaumon, M. E. et F. Sandoz-Guermond (2006). Apports croisés des démarches d'inspection et de test d'usage dans l'évaluation de l'accessibilité de e-services. In *Actes de congrès ERGO IA'2006*, Biarritz, pp. 11–13.

Choong, Y.-W., A. Giacometti, D. Laurent, P. Marcel, E. Negre, et N. Spyratos (2007). 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*, pp. 131–146. Cépaduès.

Colas, S., N. Monmarché, et M. Slimane (2008). Génération de plan de site web pour les non-voyants par des fourmis artificielles. In *Revue d'Intelligence Artificielle , Numéro Spécial Métaheuristiques (RIA)*, Volume 22, pp. 137–159. Hermes, ISBN 978-2-7462-2156-7.

Colorni, A., M. Dorigo, et V. Maniezzo (1991). Distributed Optimization by Ant Colonies. In *Proceedings of the first European Conference on Artificial Life (ECAL'91), edited by F. Verela and P. Bourgine*, Cambridge, Mass, USA, MIT Press, pp. 134–142.

Dorigo, M., V. Maniezzo, et A. Colorni (1996). The Ant System : Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B 26*(1), 29–41.

Giacometti, A., P. Marcel, E. Negre, et A. Soulet (2009). Query recommendations for olap discovery driven analysis. In *DOLAP*, pp. 81–88.

Huang, X., Q. Yao, et A. An (2006). Applying language modeling to session identification from database trace logs. *Knowl. Inf. Syst. 10*(4), 473–504.

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

Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society 7*, pp. 48–50.

Landauer, T. et D. Nachbar (1985). Selection from alphabetic and numeric menu trees using a touch screen : breadth, depth and width. In *CHI'85, ACM*, New-York, pp. pp. 73–78.

Leulier, C., J. M. C. Bastien, et D. L. Scapin (1998). Compilation of ergonomic guidelines for the design and evaluation of Web sites. In *Commerce & Interaction Report*, Rocquencourt (France) : Institut National de Recherche en Informatique et en Automatique.

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

Miller, C. S. (2004). Modeling information navigation : Implications for information architecture. In *Human Computer Interaction*, Volume 19, pp. pp. 225–271.

Narula, S. et C. Ho (1980). Degree-constrained minimum spanning trees. In *Computers and Operations Research*, Volume 7, pp. 239–249.

Negre, E. (2009). *Exploration collaborative de cubes de données*. Ph. D. thesis, Université François Rabelais Tours.

Nogier, J.-F. (2005). Ergonomie du logiciel et design web (3e édition). Dunod, Paris, France.

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

Prim, R. (1957). Shortest connection networks and some generalizations. In *Bell Systems Technology Journal 36*, pp. 1389–1401.

Reimann, M. et M. Laumanns (2004). A hybrid aco algorithm for the capacitated minimum spanning tree problem. In *First Internationnal Workshop on Hybrid Metaheuristics (HM 2004)*.

Salton, G. et M. McGill (1983). Introduction to modern information retrieval. In *McGraw-Hill Book Company*.

Sapia, C. (2000). PROMISE : Predicting Query Behavior to Enable Predictive Caching Strategies for OLAP Systems. In *DaWaK*, pp. 224–233.

Sarawagi, S. (1999). Explaining Differences in Multidimensional Aggregates. In *VLDB*, pp. 42–53.

Sathe, G. et S. Sarawagi (2001). Intelligent Rollups in Multidimensional OLAP Data. In *VLDB*, pp. 531–540.

Organisation d'un log de requêtes OLAP

Summary

If seen as a simple list of queries, an OLAP server query log is a very user unfriendly structure. In this paper we propose to organize an OLAP server query log as a website of queries, for presentation to the user. This has many advantages, like quickly understanding how the cube was queried or supporting subsequent analyses. The technique we use has been used successfully for website reorganisation. It relies on transforming the log into a graph of queries weighted with a similarity between queries. This similarity is computed by taking into account both the facts queried and the belonging of the two queries to the same session. The weights are adjusted by a heuristic using artificial ants to balance these two factors. Finally the log is presented as a website whose structure is the minimum spinning tree of this graph. This technique is implemented in java to process logs of MDX queries. Experiments were conducted to assess the quality of the resulting website.