Summarizing and Querying Logs of OLAP Queries

Julien Aligon, Patrick Marcel, Elsa Negre

Summarizing and querying logs of OLAP queries

Julien Aligon and Patrick Marcel and Elsa Negre

Abstract Leveraging query logs benefits the users analyzing large data warehouses with OLAP queries. But so far nothing exists to allow the user to have concise and usable representations of what is in the log. In this article, we present a framework for summarizing and querying OLAP query logs. The basic idea is that a query summarizes another query and that a log, which is a sequence of queries, summarizes another log. Our formal framework includes a language to declaratively specify a summary, and a language for querying and manipulating logs. We also propose a simple measure based on precision and recall, to assess the quality of summaries, and two strategies for automatically computing log summaries of good quality. Finally we show how some simple properties on the summaries can be used to query the log efficiently. The framework is implemented using the Mondrian open source OLAP engine. Its interest is illustrated with experiments on synthetic yet realistic MDX query logs.

1 Introduction

It is becoming accepted that leveraging query logs would help the user analyzing large databases or data warehouses [10]. As a clear evidence of this, it has recently been shown that browsing and querying logs actually speeds up the query formulation, by supporting better query reuse [11].

This is particularly relevant in a collaborative context for instance to issue recommendations [3, 7, 8, 19]. But to the best of our knowledge, even the simple problem

Julien Aligon, Patrick Marcel
Université François Rabelais Tours, Laboratoire d'Informatique, France. e-mail: firstname.lastname@univ-tours.fr

Elsa Negre
Université Paris-Dauphine, LAMSADE, France. e-mail: elsa.negre@dauphine.fr

of providing the end user with a concise representation of what is inside a large log has rarely been addressed, apart from helping a DBA to tune the RDBMS [10].

Using such a summary, that avoids overwhelming the user, would have many advantages, including:

- allowing a decision maker to have a rough idea of the queries launched by other decision makers,
- helping the user to access the precise part of the log containing particular queries he/she is looking for,
- helping an administrator to manage and tune the OLAP server, e.g., if the summary indicates the frequently accessed members,
- aassisting the decision maker to perform new analysis sessions by considering the previous queries.

In this article we present and develop the work initiated in [1, 2]. In these papers, we proposed a framework for summarizing an OLAP query log, and we studied basic properties of the framework for helping the user to query the log. The present article provides a detailed presentation of the framework and introduces its implementation as a system for summarizing and querying log files. To this end, we extend the search facilities introduced in [2] to obtain a declarative language with which complex queries over a log file can be expressed.

Our approach is based on the idea that a log, which is a sequence of queries, is summarized by another sequence of queries, i.e., by another (much shorter) log. It entails that a query summarizes other queries. Our framework includes:

- A language tailored for OLAP queries, named *QSL*, for declaratively expressing summaries. This language is composed of binary and unary operators that allow to summarize queries.
- A greedy algorithm using *QSL* for automatically constructing summaries of query logs.
- A quality measure adapted from the classical precision and recall, that allows to measure how faithful the constructed summaries are.
- Two sub-languages of *QSL* whose properties w.r.t. the quality measure are used to ensure that summaries can help query the log efficiently.
- Compositional search operators with which the user can query the log for particular OLAP queries.

This paper is organized as follows. Next section motivates the approach with a toy example. The *QSL* query language which is at the core of our framework is presented in Section 3. Section 4 describes the quality measure based on precision and recall, that is used to assess the summaries expressed in *QSL*. In Section 5, we present the algorithm that automatically constructs summaries based on *QSL* and the quality measure. We also introduce the properties of the summaries constructed with sub-languages of *QSL*. Section 6 presents the language for querying logs, and describes how properties of the framework can be used to ensure efficient searches. Section 7 describes the implementation of the framework and the experiments conducted to evaluate its effectiveness. Section 8 discusses related work. We conclude and draw perspectives in Section 9.

2 Motivating example

In this section, we illustrate with a toy example our approach for summarizing a log of OLAP queries. The context of this example is that of a user navigating a data warehouse. In our example, the data warehouse records sales of beverages in different locations at different times. The dimensions of this data warehouse are given in Figure 1. Consider a sequence of queries $L = \langle q_1, q_2, q_3 \rangle$ where q_1 is the first query launched, q_2 the second one and q_3 the last one. Suppose these queries are logged in a log L and ask respectively for:

1. The sales of Pepsi and Coke for July 2008, in cities Paris or Marseille,
2. The sales of Coke for July 2008, in regions North or South,
3. The sales of Orangina for the second semester 2008, in regions North or South.

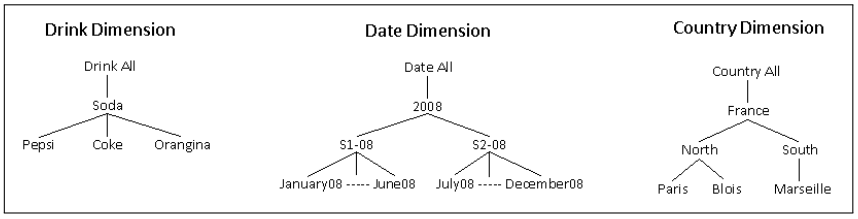

Fig. 1 Dimensions used in the toy example

Assume we want to summarize these queries by another query. Various solutions are possible. First, we can summarize the queries by retaining for each dimension the most frequent members. This could be of interest for a DBA who would like to know what indices to store. In that case, the resulting query would ask for sales of Coke in regions North or South during July 2008 (i.e., query q_2). A second alternative would be to summarize the queries with another query having for each dimension the members that cover all members present in the initial queries. For example, note that Pepsi, Coke and Orangina are sodas, cities Paris and Marseille and regions North and South are in France and all three queries concern year 2008. The query summarizing the log L would then ask for the sales of Soda in France in 2008. The user interested in more details on the query could then query the log to find the queries that were indeed launched. Finally, note that we can have a compromise by summarizing q_1 and q_2 first, say with the second alternative, and then summarizing the resulting summary with q_3, say with the first alternative. In that case, we would obtain the query asking for the sales of Soda and Orangina in France, region North and region South, for year 2008 and the second semester of 2008.

These examples show the need for flexibility in how the summary is computed. This is why in our approach we propose that summaries can be specified declaratively with a query manipulation language called *QSL*. *QSL* expressions are used to combine several queries into a query that summarises them. Note that so far, we have illustrated the problem of summarizing queries by another query. But a set of

queries could be summarized by another set of queries. Moreover, summaries for a log should respect the fact that logs are sequences of queries. For instance, consider again L, this log could be summarized by the sequence $\langle q'_1, q_3 \rangle$ where q'_1 is a summary of q_1 and q_2 asking for the sales of Soda in France in the second semester of 2008. To automatically construct a summary from a log, we propose an algorithm that constructs *QSL* expressions for summarising subsequences of the log.

In addition, as various summaries can be computed from one log, the quality of these summaries should be evaluated. For instance, for our first alternative, the quality measure should take into account the fact that 'Orangina' is present in the log but not in the summary. In our second alternative, this measure should take into account that indeed 'North' and 'South' covers 'Paris' and 'Marseille' but also 'Blois', that is not present in the log. We propose such a quality measure that extends the classical notions of precision and recall.

Finally, note that summaries computed from a log may not give precise information on the queries in the log. For instance the user may wish to know if a query on member 'Blois' appears in the log, what are all the queries of the log that deal with 'drink' or one of its descendant, or what are the queries in the log following queries dealing with 'Coke'. We thus propose two operators that allow to express such searches on a log (or even on a summary).

3 *QSL*: a Query Summarizing Language

In this section, we formally define the manipulation language, called *QSL*, used to summarize OLAP queries.

3.1 Preliminary definitions

As the query summarizing language is tailored for OLAP queries, we first begin with the definition of an OLAP query. Note that in this paper, we do not consider query result, and thus the definition of a query result is not given.

An n-dimensional cube $C = \langle D_1, ..., D_n, F \rangle$ is defined as the classical $n+1$ relation instances of a star schema, with one relation instance for each of the n dimensions D_i and one relation instance for the fact table F. For a dimension D_i having schema $S_i = \{L^i_1, \ldots, L^i_{d_i}\}$, a member m is any constant in $\bigcup_{L^i_j \in S_i} \pi_{L^i_j}(D_i)$. For a dimension D_i, we consider that members are arranged into a hierarchy $<_i$ and we note $m <_i m'$ (or $m < m'$ or m' covers m) the fact that the member m' is the ancestor of m in this hierarchy.

Given such a cube, a cell reference (or reference for short) is an n-tuple $\langle m_1, ..., m_n \rangle$ where m_i is a member of dimension $D_i, \forall i \in [1, n]$. We define multidimensional queries as sets of references that can be expressed as Cartesian products of multisets. The reason for having multisets is to be able to define operators that count

members' occurrences. In this work, we distinguish between a query and its expression called query expression. A query expression is a tuple of multisets, one multiset of members in each dimension. The cross-product of these multisets is a multiset of references, which is the query.

Definition 1. (Query expression and Query) Given an n-dimensional cube $C = \langle D_1, ..., D_n, F \rangle$, let R_i be a multiset of members of dimension $D_i, \forall i \in [1,n]$. A query expression $q = \langle R_1, ..., R_n \rangle$ is a tuple of multisets of members, one for each dimension D_i of C. Given such an expression, the query specified by q is the multiset of references $R_1 \times ... \times R_n$.

The distinction between query expression and query is needed since a query can be specified by different query expressions. For instance, the two following expressions $\langle \{a\}, \{b,b\} \rangle$ and $\langle \{a,a\}, \{b\} \rangle$ both specify query $\{\langle a,b \rangle, \langle a,b \rangle\}$. When the context is clear, a query expression and the query it specifies will be confounded.

A log L is a finite sequence of query expressions.

Definition 2. (Log) Let C be a cube and S_C be a set of queries over C. A log L of m queries over C is a function from an ordered set $pos(L)$ of integers (called positions) of size m to S_C.

A log will be noted $L = \langle q_1, \ldots, q_m \rangle$. The set of positions of a log L is noted $pos(L)$. The set of query expressions appearing in a log L is noted $queries(L)$. We note $q \in L$ for a log L if $q \in queries(L)$. In what follows, we assume an n-dimensional cube $C = \langle D_1, ..., D_n, F \rangle$. In the subsequent definitions, i ranges from 1 to n. For a query expression $q = \langle R_1, ..., R_n \rangle$, $m_i(q) = R_i$ denotes its multiset of members in dimension D_i. The multiset $m_i(q)$ will be noted $\langle S_i, f_i \rangle$, where S_i is a set and f_i is a function giving the occurrences of each element of S_i.

Example 1. Consider the three queries q_1, q_2 and q_3 of the toy example described in the previous section. Note that q_1 can be expressed in the MDX query language:
SELECT {[*Drink*].[*DrinkAll*].[*Soda*].[*Pepsi*],
[*Drink*].[*DrinkAll*].[*Soda*].[*Coke*]} *ON COLUMNS*
Crossjoin({ [*Country*].[*CountryAll*].[*France*].[*North*].[*Paris*],
[*Country*].[*CountryAll*].[*France*].[*South*].[*Marseille*]},
{[*Date*].[*DateAll*].[2008].[*S*2-08].[*July*08]}) *ON ROWS*
FROM SalesCube

We have $m_1(q_1) = \{Pepsi, Coke\}, m_2(q_1) = \{July08\}, m_3(q_1) = \{Paris, Marseille\}$. The query expression is: $q_1 = \langle \{Pepsi, Coke\}, \{July08\}, \{Paris, Marseille\} \rangle$. The query expressions q_2 and q_3 are:

- $q_2 = \langle \{Coke\}, \{July08\}, \{North, South\} \rangle$
- $q_3 = \langle \{Orangina\}, \{S2\text{-}08\}, \{North, South\} \rangle$

The language we propose is composed of unary operators and binary operators that manipulate query expressions and output a query expression, that is called a summary query (or simply summary for short). The main idea behind the definition of these operators is that they operate dimension-wise: They define a new query expression from the one(s) in parameter by treating each dimension independently. We now present formally these operators, starting with the binary operators.

3.2 The binary operators of QSL

The first operators are the classical bag operators [6] extended to multiple dimensions.

Definition 3. (Bag operators) Given two query expressions q_1 and q_2 and $op \in \{\cup_B, \cap_B, \setminus_B\}$, $q_1\ op\ q_2$ is the query expression q with $\forall i \in [1,...,n], m_i(q) = m_i(q_1)\ op\ m_i(q_2)$.

Example 2. Consider the first two query expressions of Example 1, we have:

- $q_4 = q_1 \cup_B q_2 = \langle \{Pepsi, Coke, Coke\}, \{July08, July08\}, \{Paris, Marseille, North, South\} \rangle$
- $q_5 = q_1 \cap_B q_2 = \langle \{Coke\}, \{July08\}, \emptyset \rangle$
- $q_6 = q_1 \setminus_B q_2 = \langle \{Pepsi\}, \emptyset, \{Paris, Marseille\} \rangle$

Note that q_5 and q_6 are two different expressions of the same query which is the empty set.

The next operator gives priority to one query expression over the other.

Definition 4. (Priority operator) Given two query expressions q_1 and q_2. $q_1 \lhd q_2$ gives priority to q_1 over q_2. Hence, the priority operator $\lhd$ is simply defined by $q_1 \lhd q_2 = q_1$.

3.3 The unary operators of QSL

Our first operator outputs, for a query expression q in parameter, a query expression for which only the most frequent members of q in each dimension are retained.

Definition 5. (Mostfreq operator) Let q be a query expression with $m_i(q) = \langle S_i, f_i \rangle$ for all i. $mostfreq(q)$ is the query expression q' with $\forall i \in [1,n], m_i(q') = \langle S'_i = \{m \in S_i | \nexists m' \in S_i, f_i(m') > f_i(m)\}, f_{i_{|S'_i}} \rangle$ ($f_{i_{|X}}$ denotes the restriction of a function f_i to the set X).

Example 3. $mostfreq(q_4) = \langle \{Coke, Coke\}, \{July08, July08\}, \{Paris, Marseille, North, South\} \rangle$.

Our second operator outputs, for a query expression q in parameter, a query expression for which only the most general members of q in each dimension are retained, w.r.t. the hierarchy of the dimension.

Definition 6. (Max operator) Let q be a query expression. $max(q)$ is the query expression q' with $\forall i \in [1,n], m_i(q') = \langle S'_i = \{m \in m_i(q) | \nexists m' \in m_i(q), m <_i m'\}, f_{i_{|S'_i}} \rangle$.

Example 4. $max(q_4) = \langle \{Pepsi, Coke, Coke\}, \{July08, July08\}, \{North, South\} \rangle$.

Our last operator outputs, for a query expression q in parameter, a query expression for which only the lowest common ancestors of the members of q in each dimension are retained, w.r.t. the hierarchy of the dimension.

Definition 7. (lca operator) Let q be a query expression. Let lca be the function that outputs, for a given set of members M in dimension D_i, their common ancestor w.r.t. $<_i$, i.e., $\{m \in D_i | \forall m' \in M, (m' <_i m) \wedge \nexists m'', (m' <_i m'' \wedge m'' <_i m)\}$, or, if $lca(m) = \emptyset$ (i.e., if m is the *All* member) then $lca(m) = \{m\}$. Then, $lca(q)$ is the query expression q' with $\forall i \in [1,...,n], m_i(q') = \langle lca(m_i(q)) \rangle$.

Example 5. $lca(q_4) = \langle \{Soda\}, \{S2\text{-}08\}, \{France\} \rangle$.

3.4 Expression of various summaries

We now briefly illustrate how *QSL* can be used. For instance, consider a log L composed of 3 query expressions: $L = \langle q_1, q_2, q_3 \rangle$. This log can be summarized by the query expression q_s^1 that retains only the members that appear in all queries for each dimension, i.e., $q_s^1 = q_1 \cap_B q_2 \cap_B q_3$. Alternatively, L can be summarized by taking into account the frequency of the members used in the log: $q_s^2 = mostfreq(q_1 \cup_B q_2 \cup_B q_3)$. Finally, L can be summarized by a query roughly indicating the parts of the cube that were explored: $q_s^3 = lca(q_1 \cup_B q_2 \cup_B q_3)$. We illustrate these possibilities on our running example.

Example 6. Summarizing by retaining the common members of all queries for each dimension gives: $q_s^1 = (q_1 \cap_B q_2 \cap_B q_3) = \langle \emptyset, \emptyset, \emptyset \rangle$. Summarizing basing on the frequencies of the members gives: $q_s^2 = mostfreq(q_1 \cup_B q_2 \cup_B q_3) = \langle \{Coke\}, \{July08\}, \{North, South\} \rangle$. Summarizing with lca gives: $q_s^3 = lca(q_1 \cup_B q_2 \cup_B q_3) = \langle \{Soda\}, \{2008\}, \{France\} \rangle$.

3.5 Properties of QSL

We first note that the *QSL* language cannot be presented as an algebra. In particular, it is neither minimal, nor complete with respect to query expressions. For instance, the intersection operator can be simulated using the difference operator, hence the non minimality. In addition, not all query expressions can be computed using *QSL* due to the fact that no operation enables to move down along hierarchies. Achieving minimality and completeness, though theoretically compelling, may be of little practical use. For instance, it is well known that dropping minimality enables dedicated optimisations, as it is the case for outer-join in the relational algebra. Nevertheless, in the case of *QSL*, minimality can be achieved by dropping intersection. As to completeness, instead of defining other operators, *QSL* completeness can be characterized with respect to the kind of query expressions it can compute, which are more

general expressions (in the sense of Definition 10, introduced in Section 6.2). While a precise characterization is part of our future work, we list below the properties of the *QSL* operators. Some of these properties, like for instance the distributivity of *max* or the commutativity of *max* and *lca* are used in our implementation of the framework.

Let q, q_1, q_2 be query expressions. We have the following:

- $\cup_B, \cap_B, \setminus_B$ keep their classical properties [6].
- *max* and *mostfreq* are idempotents: $max(max(q)) = max(q)$ and $mostfreq(mostfreq(q)) = mostfreq(q)$.
- *max* is distributive over $\cup_B$ and $\setminus_B$: $max(q_1 \cup_B q_2) = max(max(q_1) \cup_B max(q_2))$ and $max(q_1 \setminus_B q_2) = max(q_1 \setminus_B max(q_2))$.
- $\lhd$ is associative: $q \lhd (q_1 \lhd q_2) = (q \lhd q_1) \lhd q_2 = q$.
- *max* and *lca* commute: $max(lca(q)) = lca(max(q)) = lca(q)$.
- $mostfreq(lca(q)) = lca(q)$.

4 Assessing the quality of a summary

In this section, we present the measure used to evaluate the quality of summaries. We begin with an intuitive presentation, then give the formal definition and we finally give the properties of the *QSL* operators w.r.t. this measure.

4.1 Intuition

The measure should assess to which extends a query (respectively, a log), which is a set of references (respectively, of queries), is a faithful summary of another query (respectively, another log). The operators of *QSL* define summaries by adding or removing references to their operands. For instance the *lca* operator summarizes by adding references containing ancestors. The measure should thus assess the proportion of what is added or removed to define the summary. This is achieved by adapting the classical notion of precision and recall. In our context, these measures should be extended to take into account the cover relation used by the operators.

For instance, in Example 5, the expression $lca(q_1 \cup_B q_2)$ summarizes q_1 and q_2 by $\langle \{Soda\}, \{S2\text{-}08\}, \{France\} \rangle$, which specifies the query $q = \{Soda\} \times \{S2\text{-}08\} \times \{France\}$. Looking at the references of q_1, q_2 and q, it can be seen that q is obtained by removing references $\{Coke, Pepsi\} \times \{July08\} \times \{Paris, Marseille, North, South\}$ and adding the reference $\{Soda\} \times \{S2\text{-}08\} \times \{France\}$. If we apply the classical precision and recall measures to evaluate its quality, both are null. However, we can consider this summary as a good summary with a good quality since the added reference covers the removed references. Its recall would then be 1 and its precision would depend on the number of references covered by the added reference and not in the removed references.

We propose to extend recall and precision by taking into account a cover relation between the elements of the two sets, the summary and the summarized. In this article we use the cover relation defined over references since both queries and logs can be seen as sets of references, and thus the quality measure can be used on queries or on logs, or on any sets of references. Note that the definition of the measure is even more general in the sense that it does not rely on a particular cover relation. We now formalize these notions.

4.2 Definitions and properties

We first introduce the notion of coverage of references.

Definition 8. (Coverage) A reference $r = \langle m_1, ..., m_n \rangle$ covers another reference $r' = \langle m'_1, ..., m'_n \rangle$ if $\forall_i \in [1,n]$, $m_i >_i m'_i$ or $m_i = m'_i$. For a set R of references, cover(R)=$\{f \in \Pi_{L_1^1}(D_1) \times \Pi_{L_1^2}(D_2) \times ... \times \Pi_{L_1^n}(D_n) \mid \exists r \in R, r \text{ covers } f\}$.

Figure 2 illustrates this principle. We note L the set of references of some queries to be summarized, S the set of references of the summary, $K = L \cap S$, $D = L \setminus K$ and $A = S \setminus K$. The coverages of D and A are references (denoted by $cover(A)$ and $cover(D)$ and depicted with the same color as A and D respectively) in the most detailed level.

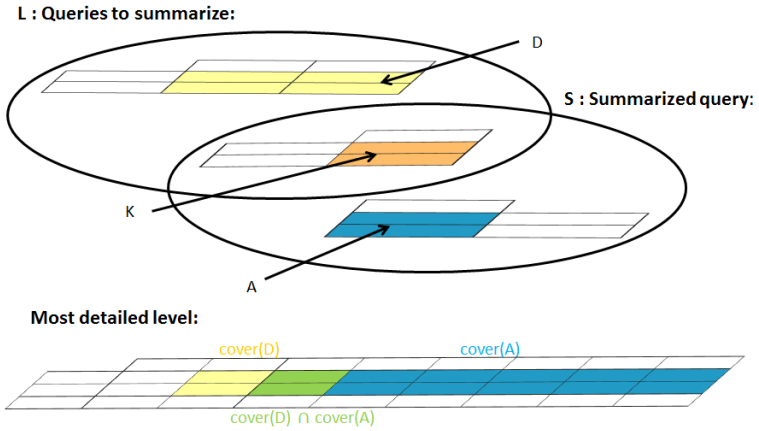

Fig. 2 Principle of the Quality Measure

For instance, consider Example 5. $L = q_1 \cup q_2$, $S = lca(q_4)$, $K = \emptyset$ and $cover(A) = \{Pepsi, Coke, Orangina\} \times \{July08, August08, September08, October08, November08, December08\} \times \{Paris, Blois, Marseille\}$ with $|cover(A)| = 54$. $cover(D) = \{Pepsi, Coke\} \times$

$\{July08\} \times \{Paris, Marseille\} \cup \{Coke\} \times \{July08\} \times \{Blois\}$ and $|cover(D)| = 5$. We have $cover(D) \subset cover(A)$. Intuitively, we expect a maximum recall and a bad precision because all covered references are recalled but a lot of other references are introduced.

To formalize this intuition, our measure of recall is the proportion of covered references existing in $cover(D)$ and found in $cover(A)$ compared with the set of references in $cover(D)$. Moreover, recall favours maximality of K. Our measure of precision is the proportion of covered references existing in $cover(D)$ and found in $cover(A)$ compared with the set of references in $cover(A)$. As for recall, precision encourages maximality of K. Of course if the summary is empty then the measure should be zero.

Definition 9. (*hf-measure*) Let L and R be two sets and $K = L \cap R$, $D = L \setminus K$ and $A = R \setminus K$. Let $\{D_1, ..., D_n\}$ be the set of dimensions defining the coverage. *h-recall* is $r = \frac{|K \cup (cover(D) \cap cover(A))|}{|K \cup cover(D)|}$ and *h-precision* is $p = \frac{|K \cup (cover(D) \cap cover(A))|}{|K \cup cover(A)|}$. These measures are aggregated with the classical F-measure: $hf\text{-}measure\ (L, R, \{D_1, ..., D_n\}) = 2\ \times \frac{p \times r}{p + r}$.

We conclude this section by noting that all operators of *QSL* maximize either *h-recall* or *h-precision*. Indeed, $\cup_B$ and *lca* lead to a *h-recall* of 1 and precision in 0 and 1, and all other operators lead to a *h-precision* of 1 and a recall in 0 and 1. Table 1 gives the range of values for *h-recall*, *h-precision*, recall and precision of each operator of *QSL*. The following property can easily be shown.

Property 1. Let L and R be two sets and $\{D_1, ..., D_n\}$ be a set of dimensions defining a coverage. $hf\text{-}measure\ (L, R, \{D_1, ..., D_n\}) = 1$ if and only if R and L cover exactly the same set of references.

operators	*h-precision*	*h-recall*	precision	recall
$\cup_B$	[0..1]	1	[0..1]	1
$\cap_B$	1	[0..1]	1	[0..1]
$\setminus_B$	1	[0..1]	1	[0..1]
$\triangleleft$	1	[0..1]	1	[0..1]
lca	[0..1]	1	[0..1]	[0..1]
max	1	[0..1]	1	[0..1]
most freq	1	[0..1]	1	[0..1]

Table 1 Table of *h-recall*, *h-precision*, recall and precision for each operator

5 Automatic summarization of a query log

In this section, we present an algorithm for summarizing a log, based on *QSL* and our quality measure hf-*measure*. The main idea is that a summary of a log is also a log. We also present the properties of the summaries constructed with the algorithm.

5.1 SummarizeLog Algorithm

SummarizeLog algorithm is a greedy algorithm successively summarizing the queries of a log using *QSL* operators until a given length α for the summary is reached. The *QSL* expression used is that maximizing hf-*measure* while changing the log. Two strategies are defined for the choice of the expression. The first one checks for each query or each pair of consecutive queries what is the *QSL* operation maximizing hf-*measure*. The chosen expression is this particular operation (strategy 1). The second strategy checks for each pair of consecutive queries what is the *QSL* binary operation maximizing hf-*measure* and applies this operation. Then the strategy looks for on this result, the unary operation maximizing hf-*measure* (strategy 2). In this case, the *QSL* expression used is of the form $u(q\ b\ q')$ where u is a unary operator and b is a binary operator. In what follows, if q is a query resulting of a *QSL* expression, we call $queries(q)$ the set of queries involved in the *QSL* expression defining q.

Let us illustrate briefly how strategy 1 operates on the toy example. Suppose it is called with the following parameters: $L = \langle q_1, q_2, q_3 \rangle$, $\mathscr{U}$ and $\mathscr{B}$ are respectively the sets of unary and binary operators of *QSL*, D is the set of dimensions of the toy example and $\alpha = 2$. All unary operators are applied on each query q_1, q_2, q_3 of the log and only the output that effectively summarizes the query is considered, i.e., a summary different from the query it summarizes and that achieves the best hf-*measure* (line 2-10). In our example, this is $lca(q_3)$. Then all binary operators are applied on each pair of consecutive queries q_1, q_2 and q_2, q_3. Again, only the summary achieving the best hf-*measure* is considered (line 11-12), in our example this is $q_1 \cup_B q_2$. Finally, among the two summaries considered, the one achieving the best hf-*measure* is used to produce the summary of the log at this step. In our example, the resulting summary at this step is $\langle q_1 \cup_B q_2, q_3 \rangle$. The algorithm then stops since the desired length of the summary, 2, is reached.

5.2 Properties of the summaries

We first note that by construction, the summary S of a log L defines a partition of the log. Indeed, each query of S is defined by a *QSL* expression that involves a distinct subsequence of queries in L.

Algorithm 1 SummarizeLog (strategy 1)

INPUT:
L: A log
$\mathscr{U}$: A set of unary operators
$\mathscr{B}$: A set of binary operators
D: A set of dimensions
α: A positive integer
OUTPUT: A summary of L
VARIABLES:
q_u, q_b: Queries
max_u, max_b: Real

while $|L| > \alpha$ **do**
  $max_u \leftarrow 0$
  for each $op \in \mathscr{U}$ **do**
    for each $q \in L$ **do**
      if $op(q) \neq q$ and $hf\text{-}measure(q, op(q), D) > max$ **then**
        $max_u \leftarrow hf\text{-}measure(q, op(q), D)$
        $q_u \leftarrow op(q)$
      end if
    end for
  end for
  $q_b \leftarrow argmax_2(\{hf\text{-}measure(q \cup q', op(q,q'), D) | op \in \mathscr{B}, L^{-1}(q) = L^{-1}(q') - 1\})$
  $max_b \leftarrow max(\{hf\text{-}measure(q \cup q', op(q,q'), D) | op \in \mathscr{B}, L^{-1}(q) = L^{-1}(q') - 1\})$
  if $max_u > max_b$ **then**
    replace in L $queries(q_u)$ by q_u
  else
    replace in L $queries(q_b)$ by q_b
  end if
end while
return L

Algorithm 2 SummarizeLog (strategy 2)

INPUT:
L: A log
$\mathscr{U}$: A set of unary operators
$\mathscr{B}$: A set of binary operators
D: A set of dimensions
α: A positive integer
OUTPUT: A summary of L
VARIABLES: q_u, q_b: Queries

while $|L| > \alpha$ **do**
  $q_b \leftarrow argmax_2(\{hf\text{-}measure(q' \cup q'', q''', D) | q''' = op(q', q''), op \in \mathscr{B}, q', q'' \in L\})$
  $q_u \leftarrow argmax_2(\{hf\text{-}measure(q_b, q'', D) | q'' = op(q_b), op \in \mathscr{U}\})$
  replace in L $queries(q)$ by q_u
end while
return L

Property 2. (Partitioning) A summary $S = \langle s_1, \ldots, s_m \rangle$ of a log L defines a partition of L where each s_i summarizes with a *QSL* expression a non empty subsequence of L, the summarized sequences being pairwise disjoint and covering exactly L.

Using the properties of the *QSL* operators, we identify two sublanguages called respectively QSL^r and QSL^p. QSL^r is the language composed of operators maximizing the *h-recall* i.e., $QSL^r = \{\cup_B, lca\}$ and QSL^p is the language composed of operators maximizing *h-precision*, i.e., $QSL^p = \{\cap_B, \setminus_B, \lhd, mostfreq, max\}$. These two languages lead to the following simple properties. In what follows, we call for a query q, $member(q)$ the set of members appearing in q, i.e., $member(q) = \bigcup_i m_i(q)$ and for a set X of queries, $member(X) = \bigcup_{q \in X} member(q)$.

Property 3. (Query defined with QSL^r) Let q^r be a query defined with a QSL^r expression and let m be a member. If there is no member $m' \in member(q^r)$ such that $m' \geq m$ then $m \notin member(queries(q^r))$ and $\nexists m'' \in member(queries(q^r))$ such that $m > m''$. If $\exists m' \in member(q^r)$ such that $m > m'$ then $\exists m'' \in member(queries(q^r))$ such that $m > m''$.

This property states that if a summary is constructed only with operators maximizing *h-recall*, then every member not covered by a member appearing in the summary cannot appear in the queries involved in the expression. A dual property holds for *h-precision*.

Property 4. (Query defined with QSL^p) Let q^p a query defined with a QSL^p expression and m a member. If $m \in member(q^p)$ then $m \in member(queries(q^p))$.

These two properties extend straightforwardly to summaries.

Property 5. (Summary defined with QSL^r) Let S^r be a summary constructed with QSL^r expressions from a log L. If a member m is not covered by a member appearing in S^r, then neither m nor none m' covered by m can appear in L. If m covers some members of S^r, then m covers members of L.

Property 6. (Summary defined with QSL^p) Let S^p be a summary constructed with QSL^p expressions from a log L. A member m appearing in S^p appears necessarily in L.

The following section illustrates the interest of these properties.

6 Querying the log efficiently

In this section, we propose a language for searching a log. We first begin by describing how the properties given in the previous section allow for efficient searches in the log.

6.1 Using summaries for an efficient search

If a query log is very large, and does not fit in main memory, searching for a member in this log can be very costly. We now describe how the basic properties of *QSL* operators can be used for efficient querying. Suppose that for a given log L, two summaries are available, the first one S^r constructed with QSL^r and the second one S^p constructed with QSL^p. Consider a first boolean function called $lookup(m)$ that returns true if a member m is present in some queries of the log, or false otherwise. The $lookup$ algorithm (see Algorithm 3), uses properties 2 to 6 to avoid accessing all the log.

Algorithm 3 lookup

INPUT:
L: a log,
S^r: a summary of L constructed with QSL^r,
S^p: a summary of L constructed with QSL^p,
m : a member.
OUTPUT: A boolean.
1: **if** $m \in member(S^p)$ **then**
2: return True
3: **end if**
4: **if** $\exists q \in queries(S^r)$ with $q = q_1 \cup_B \ldots \cup_B q_x$ and $m \in member(q)$ **then**
5: return True
6: **end if**
7: **for** each $q \in queries(S^r)$ such that $\exists m' \in member(q)$ with $m' \geq m$ **do**
8: **for** each $q' \in candidateQueries(q,m)$ **do**
9: **if** $m \in q'$ **then**
10: return True
11: **end if**
12: **end for**
13: **end for**
14: return False

Algorithm 4 candidateQueries

INPUT:
q: a query,
m : a member.
OUTPUT: A set of queries where m may appear.
VARIABLE: A set of queries Q, a set of members M.
1: $Q \leftarrow \emptyset$
2: let $lca(e_1) \cup_B \ldots \cup_B lca(e_x) \cup_B q_1 \cup_B \ldots \cup_B q_y$ be the QSL expression defining q
3: $M \leftarrow \{m' \in member(q) | m' \geq m\}$
4: **if** $m \in M$ **then**
5: $Q \leftarrow Q \cup \{q_1, \ldots, q_y\}$
6: **end if**
7: **for** each $m' \in M$ **do**
8: **for** each q' appearing in $lca(e_1) \cup_B \ldots \cup_B lca(e_x)$ **do**
9: **if** (q' appears in a number of compositions of lca $\leq level(m') - level(m)$) OR m is DefaultMember **then**
10: $Q \leftarrow Q \cup \{q'\}$
11: **end if**
12: **end for**
13: **end for**
14: return Q

Example 7. Consider the log of Example 1 and its summaries $S^r = \langle q'_1, q'_2 \rangle$ and $S^p = \langle q'_3 \rangle$, where $q'_1 = lca(q_1) = \langle \{Soda\}, \{S2\text{-}08\}, \{France\} \rangle$, $q'_2 = q_2 \cup_B q_3 = \langle \{Coke, Orangina\}, \{July08, S2\text{-}08\}, \{North, South\} \rangle$, and $q'_3 = q_1 \lhd q_2 \lhd q_3 = q_1 = \langle \{Pepsi, Coke\}, \{July08\}, \{Paris, Marseille\} \rangle$. The call to $lookup(Pepsi)$ requires only to access S^p to answer $true$ and the call to $lookup(2008)$ requires only to access S^p and S^r to answer $false$. $lookup(Orangina)$ requires only to access S^p and S^r to answer $true$ (cf. lignes 4 to 6). To output $false$, $lookup(August08)$ requires to access S^p, S^r and finally q_1, but avoids the access to q_2 and q_3 since $August08$ cannot appear in the operands of an union whose result does not contain it (cf. lines 3 to 6 of *candidateQueries*).

lookup algorithm also serves as the basis for the algorithm $lookupCover(m)$, that particularly uses property 5. *lookupCover* returns true if there is at least one member covered by m in the log L and false otherwise.

Example 8. Consider the same queries of Example 8. The call to $lookupCover(2008)$ requires only to access S^p to answer $true$.

AlgorithmCover 5 lookupCover

INPUT:
L: a log,
S^r: a summary of L constructed with QSL^r,
S^p: a summary of L constructed with QSL^p,
m : a member.
OUTPUT: A boolean.
1: **if** $\exists m' \in member(S^p)$ *with* $m \geq m'$ **then**
2: return True
3: **end if**
4: **if** $\exists q \in queries(S^r)$ and $\exists m' \in member(q)$ *with* $m \geq m'$ **then**
5: return True
6: **end if**
7: **for** each $q \in queries(S^r)$ such that $\exists m' \in member(q)$ *with* $m' \geq m$ **do**
8: **for** each $q' \in candidateCoveredQueries(q,m)$ **do**
9: **if** $\exists m'' \in member(q')$ *with* $m \geq m''$ **then**
10: return True
11: **end if**
12: **end for**
13: **end for**
14: return False

Algorithm 6 candidateCoveredQueries

INPUT:
q: a query,
m : a member.
OUTPUT: A set of queries where m may appear.
VARIABLE: A set of queries Q, a set of members M.
1: $Q \leftarrow \emptyset$
2: let $lca(e_1) \cup_B \ldots \cup_B lca(e_x) \cup_B q_1 \cup_B \ldots \cup_B q_y$ be the QSL expression defining q
3: $M \leftarrow \{m' \in member(q) | m' \geq m\}$
4: **if** $m \in M$ **then**
5: $Q \leftarrow Q \cup \{q_1, \ldots, q_y\}$
6: **end if**
7: **for** each q' appearing in $lca(e_1) \cup_B \ldots \cup_B lca(e_x)$ **do**
8: $Q \leftarrow Q \cup \{q'\}$
9: **end for**
10: return Q

We introduce now function *getQueries*, returning the queries of the log where member m is present. It can be easily deduced from *lookup* by removing the first lines and outputting the relevant queries instead of a boolean. *getQueries* can also be used to find the queries where $m \times m'$ appears, since this corresponds to $getQueries(m) \cap getQueries(m')$, and thus it can also be used to query the log using references. *getQueries* is at the core of *getCoveredQueries* since it only requires to implement fully property 5.

Algorithm 7 getQueries

INPUT:
L: a log,
S^r: a summary of L constructed with QSL^r,
m : a member.
OUTPUT: A set of queries from L.
VARIABLES: A set of queries Q.
1: $Q \leftarrow \emptyset$
2: **for** each $q \in queries(S^r)$ such that $\exists m' \in member(q)$ with $m' \geq m$ **do**
3: **for** each $q' \in candidateQueries(q,m)$ **do**
4: **if** $m \in member(q')$ **then**
5: $Q \leftarrow Q \cup \{q'\}$
6: **end if**
7: **end for**
8: **end for**
9: return Q

AlgorithmGet 8 getCoveredQueries

INPUT:
L: a log,
m : a member.
OUTPUT: A set of queries from L.
VARIABLES: A set of queries Q.
1: $Q \leftarrow \emptyset$
2: **for** each $q \in queries(L)$ **do**
3: **for** each $q' \in candidateCoveredQueries(q,m)$ **do**
4: **if** $\exists m' \in member(q')$ *with* $m \geq m'$ **then**
5: $Q \leftarrow Q \cup \{q'\}$
6: **end if**
7: **end for**
8: **end for**
9: return Q

6.2 Querying a log

In the previous subsection, we propose algorithms to search efficiently a member in the log. We now describe a language that enables to declaratively express complex searches for retrieving queries in a log. Consider the following simple queries on a log:

- Are there queries in the log that contain the members of the query q?
- Are there queries in the log that contain members covered by the members of q?
- What are the queries in the log that contain the members of q? That contain members covered by the members of q?
- What are the queries of the log that follow a query containing members the members of q?

To define operators for searching the log with a query expression as parameter, we define the two following relations over query expressions.

Definition 10. (Specialization relation over query expression) Let q and q' be two query expressions. q specialises q', noted $q \prec q'$, if $\forall i \in [1,n]$ and for all members $m' \in m_i(q')$, there is a member $m \in m_i(q)$ such that m' covers m.

Definition 11. (Inclusion of query expressions) Let q and q' be two query expressions. $q \sqsubseteq q'$ if for all i ∈ [1,n], $m_i(q) \subseteq m_i(q')$

Example 9. $\langle\{Soda\},\{all\},\{all\}\rangle$ is more general than $\langle\{Pepsi,Drink\},\{All\},\{All\}\rangle$. The opposite is not true. $\langle\{Pepsi\},\emptyset,\emptyset\rangle$ is included in $\langle\{Pepsi,Poke\},\{2008\},\{All\}\rangle$.

The search language is composed of two operators for querying a log. The first one is unary and allows to filter the log with a query. It is noted $filterLog(L,q,comp)$ where L is a log, q is a query expression and $comp$ is a comparison symbol, either $\sqsubseteq$ or $\prec$. The second operator is binary and allows to find neighbors of queries. It is noted $getNeighbor(L,L',dir)$ where L,L' are logs and dir is one of $succ, pred$. These two operators output a log of queries as answer. We now give the formal definitions.

Definition 12. Let L be a log, q a query expression and $comp$ a comparator in $\{\sqsubseteq,\prec\}$, $filterLog(L,q,comp) = L'$ where L' is the restriction of L to the set $\{a_1,\ldots,a_p\}$ such that for all a_i, q $comp$ $L(a_i)$ is true and for all $x \in \{1,n\} \setminus \{a_1,\ldots,a_p\}$, q $comp$ $L(x)$ is false. Let L be a log, L' be a log such that $pos(L') \subset pos(L)$, with $pos(L') = \{a_1,\ldots,a_p\}$, $getNeighbor(L,L',dir) = L'' \subset L$ where, if dir is $succ$ (resp. $pred$), $pos(L'') = \{a_1+1,\ldots,a_p+1\}$ (resp. $pos(L'') = \{a_1-1,\ldots,a_p-1\}$) and for all p in $pos(L'')$, $L''(p) = L(p)$ if defined.

$filterLog(L,q,comp)$ can be implemented naively by scanning L. A more efficient implementation is proposed in Algorithm 9, where *candidateQueries* (resp., *candidateCoveredQueries*) is used for accessing only the relevant parts of the log L when $comp$ is $\sqsubseteq$ (resp., $\prec$). We illustrate these operators with some simple searches over the running example.

Example 10. Let us query the log $L = \langle q_1, q_2, q_3 \rangle$ where q_1, q_2, q_3 are the query expressions of Example 1. The query: "is member Perrier in the log?" is expressed by: $filterLog(L, \langle \{Perrier\}, \emptyset, \emptyset \rangle, \sqsubseteq)$ As this expression returns the empty set, the answer is interpreted as no. The query "what are the queries covered by Pepsi and S2-08?" is expressed by $filterLog(L, \langle \{Pepsi\}, \{S2\text{-}08\}, \{All\} \rangle, \prec)$ which returns $\langle q_1 \rangle$. The query "what are the queries that immediately follow those queries covered by Pepsi and S2-08" is expressed by $getNeighbor(L, filterLog(L, \langle \{Pepsi\}, \{S2\text{-}08\}, \{All\} \rangle, \prec), succ)$ and returns $\langle q_2 \rangle$. Finally note that summaries can also be used to query logs. Indeed SummarizeLog can be seen as an operator that outputs a log by summarizing another log. For instance, the expression $L' = SummarizeLog(filterLog(L, \langle \emptyset, \emptyset, \{North\} \rangle, \sqsubseteq))$ summarizes only queries q_2 and q_3 and $filterLog(L', \langle \{Drink\}, \emptyset, \emptyset \rangle, \sqsubseteq)$ checks if member Drink is used to summarize those queries.

Algorithm 9 filterLog

INPUT:
L: A log,
q: A query expression
$comp$: A comparator
OUTPUT: A log.
VARIABLES: A set of queries C.

Let S^r be a summary of L constructed with QSL^r
$C \leftarrow \emptyset$
for each $m_i(q)$ **do**
  for each $m \in mi(q)$ **do**
    if $comp = \sqsubseteq$ **then**
      for each $q' \in queries(S^r)$ such that $\exists m' \in member(q')$ with $m' \geq m$ **do**
        $C \leftarrow C \cap candidateQueries(q, m)$
      end for
    else
      $C \leftarrow C \cap candidateCoveredQueries(q, m)$
    end if
  end for
end for
return $L|_{\{L^{-1}(q') | q' \in C, q \ comp \ q'\}}$ {Access to L}

6.3 Use case: defining new analytical sessions

We conclude the section by presenting a realistic use case recapitulating the interest of summarizing and querying a log of OLAP queries.

Let L be a log containing a large number of past queries focused on the sales of various products.

We suppose that a user wishes to conduct a new analysis. In order to prepare his analysis, he decides to visualize a summary of L composed with only ten queries.

For summarizing a log by generalizing it, the *SummarizeLog* operator will use the QSL^r language with *Strategy* 2 because the *lca* operator (generalizing the queries) is used in each step of summarization.

Thus, the user applied the function $SummarizeLog(L)$ that outputs a summary of L.

We suppose that the user decides to conduct a new analysis about the *cola sodas*. Visualizing the summary, he notes no queries of the summary are composed with *Coke* products. However, *Soda* appears in these queries. Because *Soda* generalizes *Coke*, the user has to check if queries of the initial log are involved in an analysis about *Coke*. Therefore, he filters the initial log by using the function $filterLog(L, \langle \{Coke\}, \emptyset, \emptyset \rangle, \sqsubseteq)$. A sequence of queries L_{filter} is returned to the user. Thus, he follows these query examples for forming the first query of his new analysis session. For the rest of his analysis session, the user decides to obtain the queries immediately following the queries of L_{filter} by using the function $getNeighbor(L, filterLog(L, \langle \{Coke\}, \emptyset, \emptyset \rangle, \sqsubseteq), succ)$.

7 Implementation and tests

The framework is implemented with Java 6. The implementation has been done considering that dimensions fit in main memory. These dimensions are represented by trees storing for each member the cardinality of its coverage at the most detailed level. Tests have been run on a computer equipped with Intel Core 2 Duo CPU E8400 clocked at 3.00 GHz with 3.48 Go of usable RAM, under windows 7 ultimate edition. The logs used are synthetic logs on the Foodmart database example coming with the Mondrian OLAP engine. The process of log generation is detailed in [8] and aims at simulating real sessions. This process is based on a random choice between the DIFF and RELAX operators (described in [17] and [18]), applied on the data of the Foodmart database. These operators can automatically explore a cube by a sequence of drill-downs or roll-ups, identifying interesting differences between cell pairs. We suppose that these differences are likely to be identified by a real user, hence the simulation of an OLAP analysis.

The query generator is parametrized by a number of dimensions, called the *density*, that represents the number of dimensions used for navigation. Another parameter indicates the maximum number of queries per session. For our tests, we have used logs of high density (5 dimensions out of the 13 available are used to simulate the navigation) and low density (13 dimensions are used to simulate the navigation). The high density logs are respectively composed of 119, 242, 437 and 905 queries. The low density logs are respectively composed of 121, 239, 470 and 907 queries. In what follows, the length of summaries are expressed as a ratio of the original log size.

We have conducted a large set of tests, and we report the main results here in 4 categories:

- Study of the quality measure,
- Assessment of the two strategies proposed for SummarizeLog,
- Sensitivity of the approach to the log density,
- Efficiency of the operators for searching logs.

7.1 Study of the quality measure

The aim of our first tests is to study our quality measure. We begin by assessing the overall usage of each operator of *QSL* existing in the *QSL* expression built by SummarizeLog. Figure 3 shows for *QSL* that the hf-measure favors the union and lca operator, and that the $\setminus_B$ operator is never used.

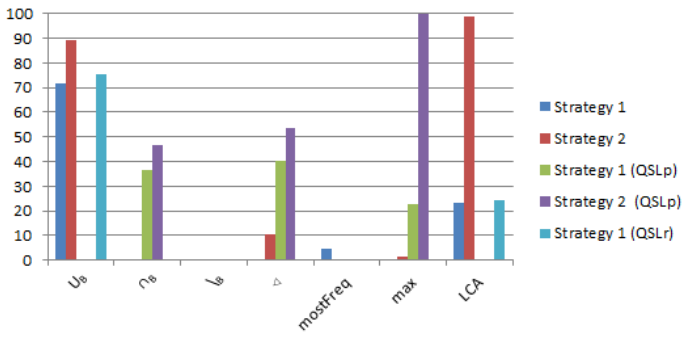

Fig. 3 Usage of the operators of *QSL*, QSL^p and QSL^r in two strategies on logs of high density

We note *hf-measure* favours the *lca* and $\cup_B$ operators. This demonstrates that the sublanguage QSL^r, which is used for implementing the search operators efficiently, is indeed of particular interest.

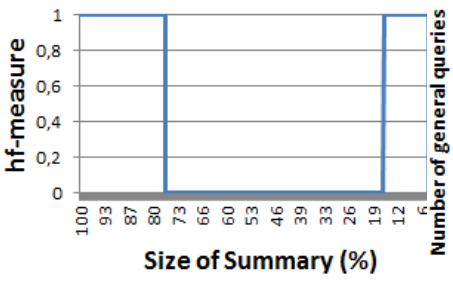

Fig. 4 Overall quality for *QSL* on 905 queries of high density in strategy 1

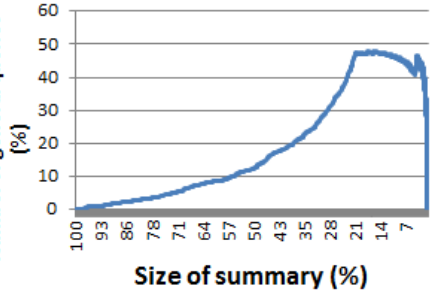

Fig. 5 Ratio of general queries for *QSL* on 905 queries of high density in strategy 1

Our second test is to compute the ratio of general queries a summary contains. A general query is a query having only the All member in each dimension. Such queries reveal little information to the user and thus their appearance in summaries should be limited. Figures 5, 10, 11, 15, 16 show that the ratio of general queries increases as the length of the summary decreases, as expected. We note that the

number of general queries never exceeds 50 % of the number of queries in the summary. This test shows that our quality measure can limit these queries and favours more interesting ones.

Finally we investigate the usefulness of the quality measure to assess the overall quality of the summaries. This overall quality is evaluated as follows: For each query q of the summary, we evaluate its quality using hf*-measure* between its references and the references of the queries of the log that q summarizes. The overall quality of the summary is the minimum, i.e., the worst, of the qualities of all queries of the summary. Interestingly, Figures 4 shows that this overall quality is eventually good for summaries of small length. It can also be seen on Figures 8,9, 17 and 18, for logs of different lengths.

7.2 Assessment of the two strategies

This series of tests assess the efficiency and effectiveness of the two strategies proposed for SummarizeLog. The behaviours reported below are observed whatever the log length. Figures 6 (with a logarithmic scale) and 7 report the computation time needed for summarizing.

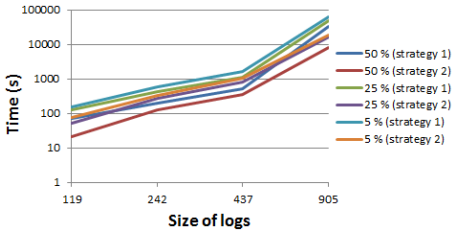

Fig. 6 Efficiency for QSL for the two strategies on logs of high density

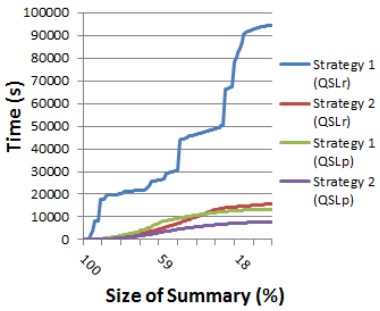

Fig. 7 Efficiency for QSL^p and QSL^r for the two strategies on 905 queries of high density

Note that, as expected, the computation time is polynomial in the length of the log. This time is quite expensive for large logs. Strategy 2 is globally more efficient, requiring less quality tests (the most expensive part of SummarizeLog). Note that due to the fact that hf*-measure* is evaluated on references and computes a coverage at the lowest level of details, the computation time for languages including $\cup_B$ can be extremely high. Indeed, for QSL^r, strategy 1 can result in successive unions that produce queries that are large sets of references, whereas strategy 2 systematically uses lca that reduces the number of references. Figures 8 and 9 show the overall quality of the summaries produced with the two strategies. It can be seen that strat-

egy 1 globally achieves a better quality than strategy 2. This can be explained by the fact that strategy 1 explores more combinations of *QSL* operators than strategy 2.

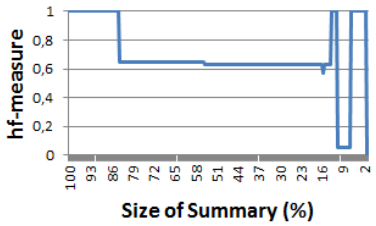

Fig. 8 Overall quality for *QSL* on 242 queries of high density for strategy 1

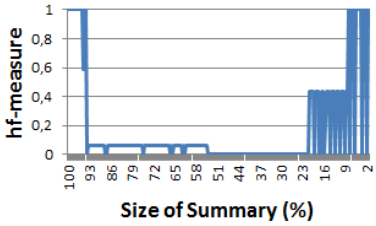

Fig. 9 Overall quality for *QSL* on 242 queries of high density for strategy 2

Finally, Figures 10, 11 and 12 indicate the ratio of general queries in the summaries constructed with each of the strategies.

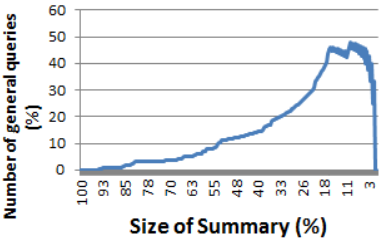

Fig. 10 Ratio of general queries for *QSL* on 242 queries of high density for strategy 1

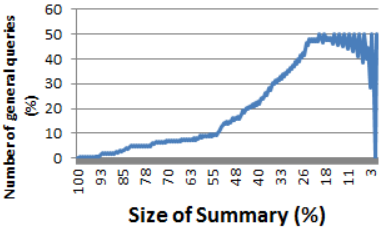

Fig. 11 Ratio of general queries for *QSL* on 242 queries of high density for strategy 2

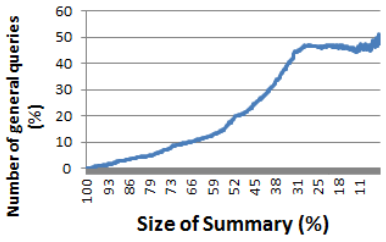

Fig. 12 Ratio of general queries for *QSL* on 905 queries of high density for strategy 2

It can be seen that strategy 1 and 2 have a similar behaviour, the number of general in queries produced with strategy 1 increasing more slowly. Consequently,

strategy 1 produces fewer general queries than strategy 2 which confirms that strategy 1 computes summaries of better quality.

7.3 Sensitivity to the log density

Figures 13, 14, 15, 16, 17, 18 report the result of tests on logs of similar lengths but different densities, in terms of efficiency, global quality and ratio of general queries. It can be seen that SummarizeLog achieves both a better quality and a better computation time on logs of high density, as expected. Remarkably, even on logs of low density, the ratio of general queries remains acceptable.

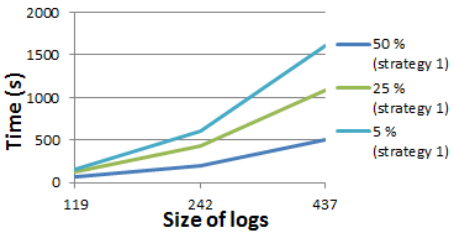

Fig. 13 Efficiency for *QSL* on three logs of high density for strategy 1

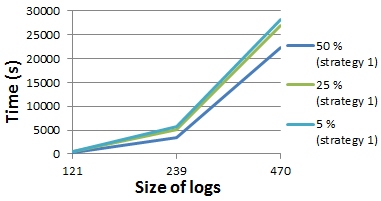

Fig. 14 Efficiency for *QSL* on three logs of low density for strategy 1

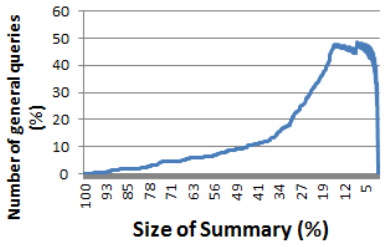

Fig. 15 Ratio of general queries for *QSL* on 437 queries of high density for strategy 1

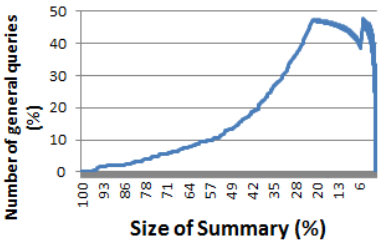

Fig. 16 Ratio of general queries for *QSL* on 470 queries of low density for strategy 1

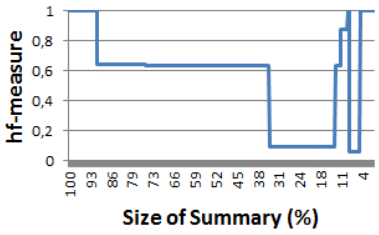

Fig. 17 Overall quality for *QSL* on 437 queries of high density for strategy 1

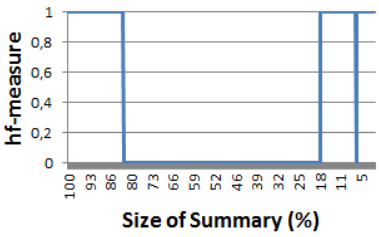

Fig. 18 Overall quality for *QSL* on 470 queries of low density for strategy 1

7.4 Efficiency of the search operators

Our last series of tests assess the efficiency of the Lookup algorithm, that is at the core of the search operators. This operator relies on two summaries constructed respectively with QSL^p and QSL^r. Figures 19 and 20 show the proportion of general queries produced by QSL^r. (Note that summaries constructed with QSL^p cannot have general queries unless already present in the initial log, which for our tests was not the case.)

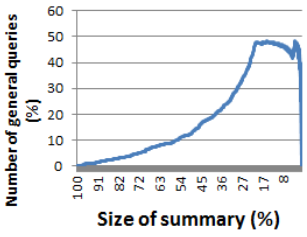

Fig. 19 Ratio of general queries for QSL^r on 905 queries of high density for strategy 1

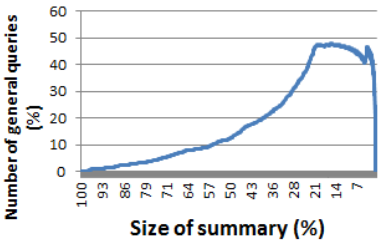

Fig. 20 Ratio of general queries for QSL^r on 907 queries of low density for strategy 1

Figures 21 and 22 show the average gain in efficiency for a lookup search of 3210 members chosen randomly, from two summaries computed with QSL^p and QSL^r for strategy 1 on logs of high and low density. The gain is the ratio of computation time between the lookup algorithm and a basic scan with disk accesses of the log file.

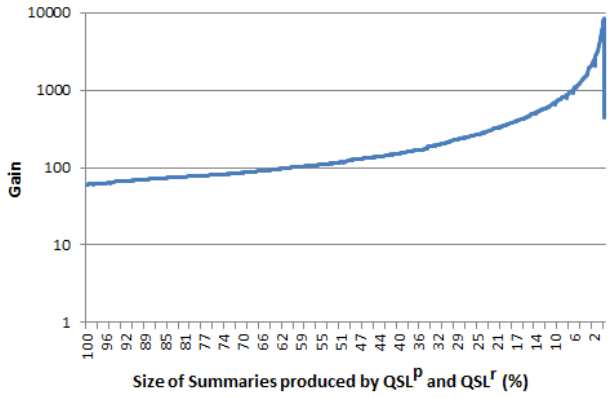

Fig. 21 Gain for Lookup Algorithm on 905 queries of high density for strategy 1

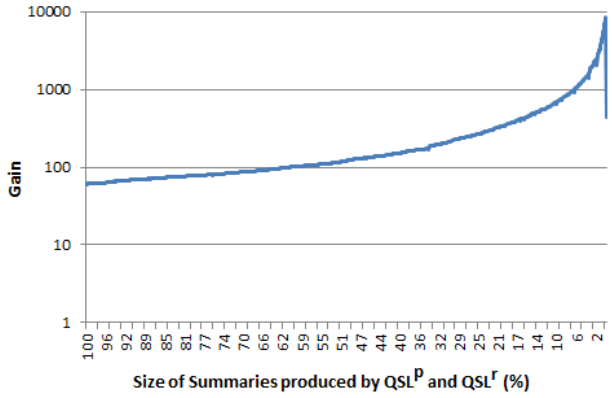

Fig. 22 Gain for Lookup Algorithm on 907 queries of low density for strategy 1

It can be seen that the gain is in favour of lookup algorithm whatever the density.

8 Related Work

Summarization of structured data has attracted a lot of attention in various domain, covering web server log [20] pattern mining (see e.g., [13] that includes a brief survey), sequences of event [14], database instance [16], multidimensional data stream [15], database workloads [4] and datacubes [12].

Many of these works rely on fuzzy set theory [20, 16] and/or are compression techniques for which it is important that original data can be regenerated [12, 13]. Moreover, it can be the case that the summary has not the same type as the data it summarizes. In the domain of databases [16, 15, 12], summarizing is applied to the database instance where, for OLAP data, measure values are taken into account.

In this paper we address the problem of summarizing an OLAP server query log. Our approach has the following salient features:

- We do not summarize a database instance, but a sequence of database queries.
- Summaries can be expressed declaratively with a manipulation language or constructed automatically.
- We do not assume any imprecise description of the members used in the queries, that could e.g., be described via fuzzy set theory. Instead, we only need the information at hand, i.e., the hierarchies described in the dimension tables.
- The type of the summary is the same as the type of the original data.
- We do not address the problem of regenerating the data from the summary, instead we focus on how to use summaries to efficiently search the log.

To the best of our knowledge, no work have yet addressed the problem of summarizing a database query log in a suitable and concise representation. As pointed out in [10], many RDBMs provide query logging, but logs are used essentially for physical tuning, and noticeably, the term workload is often termed instead of log. Notable papers are [4] for relational databases and [9] for multidimensional databases. [4] defines various primitives for summarizing query logs, essentially to filter it. The model of queries used covers both the query expression and query evaluation information (indexes used, execution cost, memory used, etc.) In this work, summarization aims at satisfying a given objective function for assisting DBA like finding queries in the log that have a low index usage. In [9], logs are analysed for identifying views to materialize, using an operator that resembles our *lca* operator.

Usually, when a query log is displayed, often in flat table or file, it is not suitable for browsing or searching into it. In our earlier work [5], we propose to organize an OLAP query log under the form of a website. But if the log is large, browsing this website may be tedious. An effective log visualization and browsing tool is yet to be designed, and the present work is a step in that direction.

9 Conclusion and Perspectives

In this article, we propose a framework for summarizing and querying OLAP query logs. This framework relies on the idea that a query can summarize another query and that a log can summarize another log. Our contributions include a query manipulation language that allows to declaratively specify a summary, and an algorithm for automatically computing a query log summary of good quality. We also propose operators for querying OLAP query logs and show how summaries can be used to

achieve an efficient implementation. The framework has been implemented and tests were conducted to show its interest.

Future work include the development and study of the different languages proposed in this article, as well as the validation of the approach on real and large query logs. Our long term goal is to study how query logs can support effectively the On-Line Analysis Process. Our future work will thus include the extension of our framework to a collaborative context where a log, composed of many sequences of queries performed by different users, each with a particular goal in mind, can be efficiently browsed and searched. Another direction is the generalisation of our framework to other types of logs (like web query logs for instance).

References

1. Julien Aligon, Patrick Marcel, and Elsa Negre. A framework for summarizing a log of OLAP queries. In *IEEE ICMWI, Special Track on OLAP and Data Warehousing*, 2010.
2. Julien Aligon, Patrick Marcel, and Elsa Negre. Résumé et interrogation de logs de requêtes OLAP. In *Proc. 11ème Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances EGC*, 2011.
3. Gloria Chatzopoulou, Magdalini Eirinaki, and Neoklis Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, pages 3–18, 2009.
4. Surajit Chaudhuri, Prasanna Ganesan, and Vivek R. Narasayya. Primitives for Workload Summarization and Implications for SQL. In *VLDB*, pages 730–741, 2003.
5. Sonia Colas, Patrick Marcel, and Elsa Negre. Organisation de log de requêtes OLAP sous forme de site web. In *EDA 2010*, volume B-6 of *RNTI*, pages 81–95, Toulouse, Juin 2010. Cépaduès.
6. Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2008.
7. Arnaud Giacometti, Patrick Marcel, and Elsa Negre. Recommending multidimensional queries. In *DaWaK*, pages 453–466, 2009.
8. Arnaud Giacometti, Patrick Marcel, Elsa Negre, and Arnaud Soulet. Query Recommendations for OLAP Discovery-Driven Analysis. *IJDWM*, 7(2):1–25, 2011.
9. Matteo Golfarelli. Handling large workloads by profiling and clustering. In *DaWaK*, pages 212–223, 2003.
10. 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.
11. Nodira Khoussainova, YongChul Kwon, Wei-Ting Liao, Magdalena Balazinska, Wolfgang Gatterbauer, and Dan Suciu. Session-based browsing for more effective query reuse. In *SSDBM*, pages 583–585, 2011.
12. Laks V. S. Lakshmanan, Jian Pei, and Jiawei Han. Quotient cube: How to summarize the semantics of a data cube. In *VLDB*, pages 778–789. Morgan Kaufmann, 2002.
13. Marie Ndiaye, Cheikh Talibouya Diop, Arnaud Giacometti, Patrick Marcel, and Arnaud Soulet. Cube based summaries of large association rule sets. In *ADMA (1)*, pages 73–85, 2010.
14. Wei Peng, Charles Perng, Tao Li, and Haixun Wang. Event summarization for system management. In Pavel Berkhin, Rich Caruana, and Xindong Wu, editors, *KDD*, pages 1028–1032. ACM, 2007.
15. Yoann Pitarch, Anne Laurent, and Pascal Poncelet. Summarizing multidimensional data streams: A hierarchy-graph-based approach. In *PAKDD (2)*, pages 335–342, 2010.

16. Régis Saint-Paul, Guillaume Raschia, and Noureddine Mouaddib. General purpose database summarization. In *VLDB*, pages 733–744, 2005.
17. Sunita Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB*, pages 42–53, 1999.
18. Sunita Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.
19. Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. "You May Also Like" Results in Relational Databases. In *PersDB*, 2009.
20. Slawomir Zadrozny and Janusz Kacprzyk. Summarizing the contents of web server logs: A fuzzy linguistic approach. In *FUZZ-IEEE*, pages 1–6. IEEE, 2007.