Mining Preferences from OLAP Query Logs for Proactive Personalization

Julien Aligon[1], Matteo Golfarelli[2],
Patrick Marcel[1], Stefano Rizzi[2], and Elisa Turricchia[2]

[1] Laboratoire d'Informatique – Université François Rabelais Tours, France
{julien.aligon,patrick.marcel}@univ-tours.fr
[2] DEIS – University of Bologna, Italy
{matteo.golfarelli,stefano.rizzi,elisa.turricchia2}@unibo.it

Abstract. The goal of personalization is to deliver information that is relevant to an individual or a group of individuals in the most appropriate format and layout. In the OLAP context personalization is quite beneficial, because queries can be very complex and they may return huge amounts of data. Aimed at making the user's experience with OLAP as plain as possible, in this paper we propose a proactive approach that couples an MDX-based language for expressing OLAP preferences to a mining technique for automatically deriving preferences. First, the log of past MDX queries issued by that user is mined to extract a set of association rules that relate sets of frequent query fragments; then, given a specific query, a subset of pertinent and effective rules is selected; finally, the selected rules are translated into a preference that is used to annotate the user's query. A set of experimental results proves the effectiveness and efficiency of our approach.

1 Introduction and Motivation

Personalization has attracted a lot of attention in the database community during the last few years, and also raised plenty of interest in the OLAP area. The goal of personalization is to deliver information that is relevant to an individual or a group of individuals in the most appropriate format and layout, and in the OLAP area it has been pursued using different approaches:

- *Query recommendation*: Based on the current query and on the past sessions, the system suggests further queries to help users navigating the cube [1].
- *Personalized visualization*: Users specify a set of constraints that are used to determine a preferred visualization [2].
- *Result ranking*: Query results are organized in a total or partial order so that the user visualizes the most relevant data first [3].
- *Query contextualization*: The query is enhanced by adding preference predicates that depend on the query context [4].

These approaches differ from different points of view, in particular:

J. Eder, M. Bielikova, and A.M. Tjoa (Eds.): ADBIS 2011, LNCS 6909, pp. 84–97, 2011.

- Formulation effort: personalization criteria for queries may be either manually specified by users, or transparently inferred from the context and from the user profile.
- Prescriptiveness: personalization criteria may either be used as "hard" constraints that are added to queries, or be meant as "soft" constraints, i.e., preferences.
- Proactiveness: some approaches propose new queries to the user based on the query log and on the context, while others change the current query or post-process its results before returning them to the user.

With reference to the above, the user's experience with OLAP can be made as plain as possible by decreasing the formulation effort (i.e., having query personalization criteria inferred), providing low prescriptiveness (i.e., annotating queries with preferences rather than constraints), and enhancing proactiveness (i.e., transparently changing the current query). The result ranking approach we propose in this paper goes in this direction by coupling an MDX-based language for expressing OLAP preferences to a mining technique for automatically deriving a set of preferences for a user's query from the log of past MDX queries issued by that user. This is done in four steps:

1. The user's query log is mined off-line to extract a set of association rules that relate sets of frequent query fragments (such as group-by attributes, returned measures, selection predicates).
2. When the user formulates a query q, among the rules whose antecedent matches with q, a subset of rules is selected whose cardinality depends on a parameter set by the user to express the desired personalization degree, i.e., the complexity of the preference that will be formulated.
3. The selected rules are translated into an OLAP preference p concerning the group-by set for aggregating data, the measures to be returned, and the values of levels or measures.
4. Query q is annotated with p and executed. The results returned are ranked according to p, so that the user can more effectively explore them by focusing on the most relevant data first.

Remarkably, like in the other result ranking approaches, the overall set of tuples returned by q annotated with p is the same set of tuples that would be returned by q without annotation, because p expresses a soft constraint. This guarantees that the user's intentions are preserved, and makes our approach non-invasive.

The paper outline is as follows. After summarizing the related work in Section 2, we introduce a formal setting to manipulate multidimensional data in Section 3. In Section 4 we describe the main features of the MYMDX language we adopt to express OLAP preferences, while Section 5 describes in detail our approach. Section 6 shows an implementation and reports the results of some experimental tests we performed to test our approach for effectiveness and efficiency.

2 Related Work

Several approaches to personalization were devised in the OLAP context.

In the field of *profile-based personalization*, we mention [2], that presents a framework for providing personalized visualization of OLAP results based on user profiles in form of constraints, and [4], that achieves OLAP personalization by dynamically enhancing queries with context-aware user preferences. Both approaches are proactive and demand low formulation effort, but in both cases the user profile is given, nothing being said on its construction. A recommendation framework for OLAP systems is presented in [5]; new queries are suggested to users based on the current analysis context and on the user's profile. Though the authors mention that the profile could be mined from the user's previous behavior, no specific suggestion is given to this end. A non-prescriptive approach is presented in [3,6], where the MYOLAP algebra for formulating and evaluating OLAP preferences is introduced; the proposed algebra is very expressive, but at the cost of a substantial formulation effort.

The term *history-based personalization* is borrowed from [7], and refers to approaches that suggest a new database query based on the past actions recorded in a log file. The following approaches fall into this category and do not rely on a user profile; they are proactive and demand no formulation effort —like our approach—, but they are prescriptive. The approaches in [1,8] are aimed at suggesting OLAP queries based on a comparison between the current session and former sessions stored in a query log. Also [9] has a similar goal in the context of SPJ queries; here, recommendations are computed based on the presence of tuples in sessions. This approach is further improved in [10] by relying on query fragments instead of tuples. A query log is exploited in [11] to support users in writing new SQL queries; the log is transformed into a graph of query fragments, where edges are labelled with the conditional probability of having one fragment given another fragment. Noticeably, all these work generally assume that history is taken from a query log shared by all users.

To the best of our knowledge, our work is the first that proposes to extract preferences from database query logs. However, the same idea has been used in other contexts. In the context of information retrieval, [12] presents algorithms to extract association rules at query time from a set of documents. These rules are used to associate the documents retrieved by a query to a relevance class and eventually to rank them. In the context of the web, [13] introduces algorithms for preference extraction from web logs, with a targeted preference language. Extraction is based on the frequency of the terms appearing in the log, and clustering is used for identifying preference constructs. A comprehensive overview of the techniques using data mining for personalization can be found in [14].

3 Preliminaries

3.1 Schemata and Instances

Our datacube formalization involves hierarchies; however, to keep the formalism simpler, and without actually restricting the validity of our approach, we will consider hierarchies without branches, i.e., consisting of chains of levels.

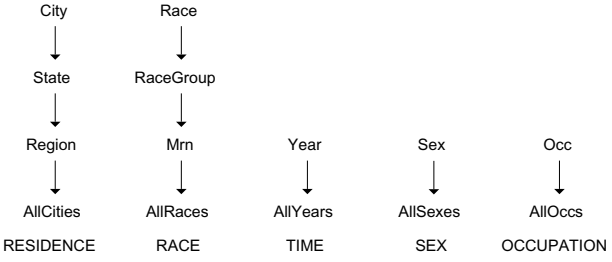


Fig. 1. Roll-up orders for the five hierarchies in the CENSUS schema (Mrn stands for MajorRacesNumber)

Definition 1 (Multidimensional Schema). *A* multidimensional schema *(or, briefly, a* schema*) is a triple* $\mathcal{M} = \langle A, H, M \rangle$ *where:*

- A *is a finite set of* levels*, each defined on a categorical domain* $Dom(a)$*;*
- $H = \{h_1, \ldots, h_n\}$ *is a finite set of* hierarchies*, each characterized by (1) a subset* $Lev(h_i) \subseteq A$ *of levels (such that the* $Lev(h_i)$*'s for* $i = 1, \ldots, n$ *define a partition of* A*); (2) a* roll-up *total order* $\succeq_{h_i}$ *of* $Lev(h_i)$*;*
- *a finite set of* measures M*, each defined on a numerical domain* $Dom(m)$*.*

For each hierarchy h_i, the top level of the order determines the finest aggregation level for the hierarchy. Conversely, the bottom level has a single possible value and determines the coarsest aggregation level.

A group-by set includes one level for each hierarchy, and defines a possible way to aggregate data. A coordinate of a group-by set is a point in the n-dimensional space defined by the levels in that group-by set.

Definition 2 (Group-by Set). *Given schema* $\mathcal{M} = \langle A, H, M \rangle$*, let* $Dom(H) = Lev(h_1) \times \ldots \times Lev(h_n)$*; each* $G \in Dom(H)$ *is called a* group-by set *of* $\mathcal{M}$*. Let* $G = \langle a_{k_1}, \ldots, a_{k_n} \rangle$ *and* $Dom(G) = Dom(a_{k_1}) \times \ldots \times Dom(a_{k_n})$*; each* $g \in Dom(G)$ *is called a* coordinate *of* G*.*

Example 1. The CENSUS schema includes the five hierarchies whose roll-up orders are shown in Figure 1, and measures AvgIncome, AvgCostGas, and AvgCostElect. It is City $\succeq_{\text{RESIDENCE}}$ State; examples of group-by sets are:

$$G_0 = \langle \text{City}, \text{Race}, \text{Year}, \text{Occ}, \text{Sex} \rangle$$
$$G_1 = \langle \text{Region}, \text{Mrn}, \text{Year}, \text{Occ}, \text{Sex} \rangle$$
$$G_2 = \langle \text{AllCities}, \text{AllRaces}, \text{AllYears}, \text{AllOccs}, \text{AllSexes} \rangle$$

A schema is populated with facts, each recording a useful information for the decision-making process. A fact is characterized by a group-by set G that defines its aggregation level, by a coordinate of G, and by a value for one measure.

Definition 3 (Fact). *Given schema* $\mathcal{M} = \langle A, H, M \rangle$*, a group-by set* $G \in Dom(H)$*, and a measure* $m \in M$*, a fact is a couple* $f_{G,m} = \langle g, v \rangle$*, where* $g \in Dom(G)$ *and* $v \in Dom(m)$*. The space of all facts for* $\mathcal{M}$ *is*

$$\mathcal{F}_{\mathcal{M}} = \bigcup_{G \in Dom(H), m \in M} (Dom(G) \times Dom(m))$$

Example 2. An example of fact is $f_{G_1,\mathsf{AvgIncome}} = \langle \langle$'Pacific', 'White', '2008', 'Dentist', 'Male'$\rangle, 600 \rangle$.

Finally, an instance of a schema (*datacube*) is a set of facts $D \subseteq \mathcal{F}_{\mathcal{M}}$ such that no two facts characterized by the same coordinate and measure exist in D.

3.2 Queries

The MDX (*MultiDimensional eXpressions*) language is a de-facto standard for querying multidimensional databases [15]. Some of its distinguishing features are the possibility of returning query results that contain data with different aggregation levels and the possibility of specifying how the results should be visually arranged into a multidimensional representation. In this paper we consider MDX queries that aggregate data at one or more group-by sets, optionally select them using a predicate in CNF, and return one or more measures. The semantics of such an MDX query is that of a union of GPSJ queries[1] whose group-by sets are the cross product of n sets of levels, one for each hierarchy. This semantics corresponds to the following subset of MDX:

- Clauses **SELECT**, **FROM**, **WHERE** are supported.
- All functions for navigating hierarchies are supported: **AllMembers**, **Ancestor**, **Ascendants**, **Children**, etc.
- All functions for manipulating sets of members or tuples are supported (**Crossjoin**, **Except**, **Exists**, **Extract**, **Filter**, **Intersect**, etc.) except the union.
- All functions for manipulating members/tuples are supported.

To effectively use association rules for modeling frequent portions of queries, we formally split MDX queries into fragments as explained below.

Definition 4 (Query Fragment, Query, Log). *Given schema* $\mathcal{M} = \langle A, H, M \rangle$*, a* query fragment *is either a level in* A*, a measure in* M*, or a simple Boolean predicate involving a level and/or a measure. A* qf-set *is a set of query fragments. A* multidimensional query *(briefly,* query*) is represented by a qf-set that includes at least one level for each hierarchy in* H *and at least one measure in* M*. A* log *is a set of multidimensional queries.*

[1] A GPSJ query takes form $\pi_{a_{k_1},\ldots,a_{k_n},Aggr}\sigma_p(\chi)$ where, in our context: χ is the star join between the fact table and the n dimension tables; p is a selection formula in CNF; $\{a_{k_1}, \ldots, a_{k_n}\}$ is a group-by set; and $Aggr$ is a list of aggregations of the form $\alpha_j(m_j)$, where m_j is a measure and α_j is an aggregation operator.

Representing an MDX query as a qf-set q means:

1. Including a fragment m in q for each measure m returned by the MDX query.
2. Including a fragment a in q for each level a used in the MDX query to aggregate data.
3. Including a fragment $(a \in V)$ in q for each simple predicate on a level/measure a used in the MDX query to filter data.

Example 3. The MDX query on the CENSUS schema

SELECT AvgIncome **ON COLUMNS,**
Crossjoin(OCCUPATION.members,
Crossjoin(**Descendants**(RACE.AllRaces,RACE.Mrn),
Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) **ON ROWS**
FROM CENSUS **WHERE** TIME.Year.[2009]

is the union of four GPSJ queries:

$$\pi_{\mathsf{AllCities,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{AllCities,Mrn,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,AllRaces,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$
$$\pi_{\mathsf{Region,Mrn,Occ,Year,AllSexes},AVG(\mathsf{AvgIncome})}\sigma_{\mathsf{Year}=2009}(\chi_{\mathsf{CENSUS}})$$

and is represented by the qf-set $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{Mrn}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} \in 2009)\}$.

4 The MYMDX Preference Language

The language we adopt in this paper to express OLAP preferences is MYMDX [6], an extension of the MDX language based on the MYOLAP algebra [3]. In this section we summarize its features of interest for this work.

A (qualitative) preference on a datacube is a *strict partial order* (i.e., an irreflexive and transitive binary relation) on the space $\mathcal{F}_{\mathcal{M}}$ of all facts. In the MYOLAP algebra, preferences are inductively engineered by writing a *preference expression* that can be either a *base constructor* or a *composition operator* applied to two preference expressions. The constructors used in this paper are[2]:

- $\mathsf{POS}(a, V)$, where $V \subset Dom(a)$, that operates on level values; facts for which a takes a value in V are preferred to the others.
- $\mathsf{BETWEEN}(m, v_{low}, v_{high})$, where m is a measure and $v_{low}, v_{high} \in Dom(m)$, that operates on measure values. Facts whose value of m is between v_{low} and v_{high} are preferred; the other facts are ranked according to their distance from the $[v_{low}, v_{high}]$ interval.

[2] The constructors we adopt are actually a generalization of those presented in [3] from two points of view. Firstly, the CONTAIN constructor is extended to work also on a fake hierarchy including all measures. Secondly, all constructors except BETWEEN are extended to operate on sets of values rather than on single values.

- CONTAIN(h, L), where h is a hierarchy and $L \subset Lev(h)$, that operates on levels. Facts whose group-by set includes a level in L are preferred to the others.
- CONTAIN(measures, $Meas$), where $Meas \subset M$, that operates on measures. Facts whose measure is in $Meas$ are preferred to the others.

Preference composition relies on the Pareto operator ($\otimes$), that gives the same importance to both the composed preferences. Remarkably, the Pareto operator is closed on the set of preferences.

The myMDX language allows an MDX query to be annotated with a preference expression through a PREFERRING clause.

Example 4. The MDX query in Example 3 can be annotated with preference expression BETWEEN(AvgIncome,500,1000) $\otimes$ POS(Occ,'Engineer') $\otimes$ CONTAIN (RESIDENCE, Region) to state that facts aggregated by region and related to engineers with average income between 500 and 1000 kiloeuros are equally preferred. The corresponding myMDX query is:

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
PREFERRING AvgIncome BETWEEN 500 AND 1000
AND Occ POS 'Engineer' AND RESIDENCE CONTAIN Region
```

5 A Proactive Approach to OLAP

As sketched in the Introduction, our approach relies on four steps:

1. *Log mining.* For efficiency reasons this step is executed off-line, before the current query session starts. It consists in running a data mining algorithm on the user's query log to extract the set R of association rules whose support and confidence are above a given threshold.
2. *Rule selection.* When that user formulates an MDX query q, a subset $R_q \subseteq R$ of rules is selected. Each rule in R_q is *pertinent*, meaning that its antecedent matches with q, and *effective*, meaning that the preference it would be translated into can actually induce an ordering on the facts returned by q. Then, let a positive integer *personalization degree* α be chosen by the user to express the desired preference complexity. A qf-set F_α is generated from R_q in such a way that α base constructors are included in the overall preference expression the fragments of F_α will be translated into.
3. *Fragment translation.* Each fragment in F_α is translated into a base constructor; the resulting base constructors are then coalesced and composed using the Pareto operator into a preference expression p.

Algorithm 1. Extract rules with support and confidence adjustment

Input: Log: A set of queries; $minSup, minConf$: Floats
Output: R: A set of association rules
Uses: $mine(set, float, float)$: An association rule extractor
Variables: $stop$: A Boolean; $confidence, support$: Floats; $Covered$: A set of qf-sets

```
1:  stop =false
2:  confidence = 1
3:  support = 1
4:  while !stop do
5:      R = mine(Log, support, confidence)        ▷ Mine rules above support and confidence
6:      R = R \ {r ∈ R s.t. |r.cons| > 1}         ▷ Only keep rules with singleton consequent
7:      Covered = ∅
8:      for each rule r ∈ R do
9:          Covered = Covered ∪ {q ∈ Log|r.ant ∪ r.cons ⊆ q}
10:     if Covered = Log then                     ▷ If all queries in the log are covered in R stop...
11:         stop =true
12:     else                                      ▷ ...else mine again with lower thresholds
13:         confidence = confidence − 0.1
14:         if confidence < minConf then
15:             support = support − 0.1
16:             confidence = 1
17:             if support < minSupp then
18:                 stop = true
19: return R
```

4. *Querying.* Query q is annotated with p, translated into MYMDX, and executed. As shown in [6], the user can effectively explore query results by visually interacting with a graph-like structure that emphasizes the better-than relationships induced by p between different sets of facts. Preferred facts are then displayed in a multidimensional table.

The following subsections explain in detail how steps 1, 2, and 3 are carried out. For details about step 4, see [3,6].

5.1 Log Mining

We now briefly describe the mining step. The input of this step is a set of qf-sets that represents the user's query log, while the output is a set R of association rules.

Interestingly, the problem of associating a query with a set of fragments representing user preferences bears resemblance to the problem of associating objects with a set of most relevant labels. This problem, named *label ranking*, is a form of classification. Both label ranking and classification have been proved to be effectively handled by association rules (see for instance [16,17]). In this context, rules have a set of features that should match the object to be classified as antecedent, and one label as consequent. We adopt a similar approach here, and we search for rules having exactly one item as consequent, so each rule $r \in R$ takes the form $ant \rightarrow cons$, where ant is a qf-set and $cons$ is a single query fragment. In the following, $r.cons$ (resp., $r.ant$) denotes the consequent (resp., antecedent) of rule r, and $conf(r)$ its confidence.

The mining step is done off-line, and uses any classical association rule extractor that is parametrized by support and confidence thresholds (e.g., Apriori [18]). The only issue in this step is to extract rules that faithfully represent the user's query log. Since the user is not involved at this step, support and confidence have to be adjusted automatically [16]. Algorithm 1 is used for this purpose, and it extracts rules until the whole log is covered by the set of rules extracted. More precisely, the algorithm starts extracting rules with confidence and support equal to 1 (lines 2,3). If the set of rules covers the entire log, then the algorithm stops (line 11,12). Otherwise, extraction starts again with a lower confidence (line 13), and confidence is decreased until the log is entirely covered or the confidence is considered too low (line 14). In this case, confidence goes back to 1 and support is decreased (line 16,17), and extraction is launched again. If both support and confidence are considered too low, then the algorithm stops.

Algorithm 1 needs two thresholds, $minConf$ and $minSupp$. Realistic values for these thresholds can be learned by training the algorithm on query logs, or be derived from log properties like size and sparseness.

5.2 Rule Selection

The output of the mining step, R can be a large set. In this section we present the algorithm that first selects, among the rules in R, the subset R_q of pertinent and effective rules for query q, and then returns a qf-set F_α including a subset of the query fragments that appear as consequents of the rules in R_q. These fragments will be used for annotating q with a preference.

Following the approach presented in [12], the selection of query fragments is made by associating a score to each group of rules in R_q having the same fragment φ as consequent. This score is the average confidence of the rules in the group, i.e., $score(\varphi) = avg_{r \in R_\varphi} conf(r)$ where $R_\varphi \subseteq R_q$ is the subset of rules having φ as a consequent. The selected query fragments are those with highest scores, and are limited by the number α of base preference constructors that the user wants to annotate her queries with.

Given schema $\mathcal{M} = \langle A, H, M \rangle$ and a qf-set F, we adopt the following notation:

- $F.hier(h) = F \cap Lev(h)$ is the set of levels of hierarchy $h \in H$ in F;
- $F.meas = F \cap M$ is the set of measures in F;
- $F.val(a) = \bigcup_{(a \in V_k) \in F} V_k$ denotes the set of selected values for level/measure $a \in A \cup M$ in F.

Algorithm 2 selects, among the set R of association rules mined from the log, the consequents of rules that will be used to annotate the current query with preferences. It starts by removing from R all non-pertinent rules (i.e., those whose antecedent does not match q — line 1), and some non-effective rules (those whose consequent, if it is an attribute or a measure, does not appear in the list of group-by attributes or returned measures of q — line 2). The remaining rules are grouped by their consequent and the score of each group is computed (line 3). Then the top consequents corresponding to α base constructors are returned

Algorithm 2. Select Consequents

Input: R: A set of rules; q: A query represented as a qf-set; α: A user-defined *personalization degree*
Output: F_α: A qf-set that will be used to annotate q with a preference
Variables: $numBC$: The current number of base constructs; R_q: The set of pertinent and effective rules; F, F_{sim}: Two qf-sets

1: $R = R \setminus \{r \in R | r.ant \not\subseteq q\}$ ▷ Drop non-pertinent rules
2: $R_q = R \setminus \{r \in R | r.cons \in A \cup M, r.cons \notin q\}$ ▷ Drop non-effective rules
3: $F = \{r.cons | r \in R_q\}$ ▷ Consequents of the rules in R_q
4: $F_\alpha = \emptyset$
5: $numBC = 0$
6: **while** $numBC \leq \alpha$ and $F \neq \emptyset$ **do** ▷ Iteratively construct F_α...
7: let $\varphi = ArgMax_F\ score(\varphi)$ ▷ ...starting with the fragment having highest score
8: $F = F \setminus \{\varphi\}$
9: **if** $makesIneffective(\varphi, F_\alpha, q)$ **then** ▷ If φ drives the preference ineffective...
10: $F_{sim} = \{\varphi' \in F_\alpha | similar(\varphi, \varphi')\}$ ▷ ...find the similar fragments, if any...
11: $F_\alpha = F_\alpha \setminus F_{sim}$ ▷ ...and drop them
12: **if** $F_{sim} \neq \emptyset$ **then**
13: $numBC - -$
14: **else**
15: **if** $\exists \varphi' \in F_\alpha | similar(\varphi, \varphi')$ **then** ▷ Other similar fragments were already added to F_α...
16: $F_\alpha = F_\alpha \cup \{\varphi\}$ ▷ ...so $numBC$ must not be increased
17: **else**
18: **if** $numBC < \alpha$ **then** ▷ Add φ only if this does not violate the α constraint
19: $F_\alpha = F_\alpha \cup \{\varphi\}$
20: $numBC + +$
21: **return** F_α

Function 3. makesIneffective

Input: φ: A fragment; F_α: A qf-set; q: a query represented as a qf-set
Output: A Boolean

1: **if** $\exists h \in H | \varphi \in Lev(h)$ **then** ▷ φ is a level
2: **if** $(F_\alpha.hier(h) \cup \{\varphi\}) = q.hier(h)$ **then** ▷ All query hierarchies are preferred
3: return true
4: **if** $\varphi \in M$ **then** ▷ φ is a measure
5: **if** $(F_\alpha.meas \cup \{\varphi\}) = q.meas$ **then** ▷ All query measures are preferred
6: return true
7: **if** $\varphi = (a \in V)$ **then** ▷ φ is a predicate
8: **if** $q.val(a) \neq \emptyset$ and $!((F_\alpha.val(a) \cup V) \subset q.val(a))$ **then** ▷ All values for a are preferred
9: return true
10: **return** false

Function 4. similar

Input: φ_1: A fragment; φ_2: A fragment
Output: A Boolean

1: **if** $\exists h \in H | \varphi_1 \in Lev(h)$ and $\varphi_2 \in Lev(h)$ **then** ▷ Two levels of the same hierarchy
2: return true
3: **if** $\varphi_1 \in M$ and $\varphi_2 \in M$ **then** ▷ Two measures
4: return true
5: **if** $\varphi_1 = (a \in V_1)$ and $\varphi_2 = (a \in V_2)$ **then** ▷ Two predicates on the same attribute
6: return true
7: **return** false

(lines 4-21). If a fragment φ that is about to be selected drives the preferences ineffective because it states that *all* the query results are preferred (Function 3), it is removed together with the other similar fragments (lines 10-13).

Example 5. Consider the qf-set of Example 3, $q = \{\mathsf{Region}, \mathsf{AllCities}, \mathsf{Mrn}, \mathsf{AllRaces}, \mathsf{Occ}, \mathsf{Year}, \mathsf{AllSexes}, \mathsf{AvgIncome}, (\mathsf{Year} \in 2009)\}$. Let the set R of rules extracted from the log be as follows:

r_1: (Region ∈ {'Pacific','Atlantic'}) → Year (0.8)
r_2: Year → Region (0.80)
r_3: Year → AllCities (0.60)
r_4: AvgIncome → Region (0.60)
r_5: Year → Sex (0.90)
r_6: (Year ∈ 2009) → Region (0.70)
r_7: Year → (Year ∈ 2009) (0.50)
r_8: Year → (AvgIncome ∈ [500, 1000]) (0.55)
r_9: AvgIncome → Mrn (0.45)
r_{10}: Occ → Region (0.70)
r_{11}: Occ → Year (0.10)
r_{12}: AvgIncome → Year (0.70)

and let Algorithm 2 be called with $\alpha = 2$. First, the algorithm removes r_1 (non pertinent) and r_5 (non effective). Then the remaining rules are grouped by their consequents, resulting in the set of fragments $F = \{$Region, AllCities, (AvgIncome ∈ [500, 1000]), (Year ∈ 2009), Mrn, Year$\}$ (listed by decreasing order of score). The fragments in F are now orderly explored. The first two fragments are not selected since, together, they drive the preference ineffective (they are exactly the fragments of hierarchy RESIDENCE included in q). Fragment (AvgIncome ∈ [500, 1000]) is selected. Fragment (Year ∈ 2009) is not selected since it corresponds precisely to the selection on Year of q. Then fragment Mrn is selected and, finally, Algorithm 2 outputs $F_\alpha = \{$(AvgIncome ∈ [500, 1000]), Mrn$\}$.

5.3 Fragment Translation

The output F_α of Algorithm 2 is a qf-set used to annotate the current query q with a preference. To this end, each query fragment $\varphi \in F_\alpha$ is translated into a base constructor (see Section 4); the resulting base constructors are then coalesced and composed using the Pareto operator.

The rules for translating fragment φ are explained below:

- if φ is a level $a \in A$, it is translated into a constructor CONTAIN(h, a), where h is the hierarchy a belongs to.
- If φ is a measure $m \in M$, it is translated into a constructor CONTAIN (measures,m).
- If φ is a Boolean predicate on a level, ($a \in V$), it is translated into a constructor POS(a, V).
- If φ is a Boolean predicate on a measure, ($m \in [v_{low}, v_{high}]$), it is translated into a constructor BETWEEN(m, v_{low}, v_{high}).

The resulting base constructors are coalesced by merging all CONTAIN's on the same hierarchy, all POS's on the same level, and all BETWEEN's on the same measure.

Example 6. The preference expression that translates the qf-set F_α in Example 5 is p = BETWEEN(AvgIncome,500,1000) ⊗ CONTAIN(RACE, Mrn). The MYMDX formulation for q annotated with p is:

```
SELECT AvgIncome ON COLUMNS,
        Crossjoin(OCCUPATION.members,
                Crossjoin(Descendants(RACE.AllRaces,RACE.Mrn),
                Descendants(RESIDENCE.AllCities,RESIDENCE.Region))) ON ROWS
FROM CENSUS WHERE TIME.Year.[2009]
PREFERRING AvgIncome BETWEEN 500 AND 1000 AND RACE CONTAIN Mrn
```

6 Experimental Results and Conclusions

In this paper we proposed a proactive approach to personalization, where mining techniques are applied to transparently annotate OLAP queries with preferences. This section briefly describes the implementation of our approach and reports the results of tests assessing its efficiency and effectiveness.

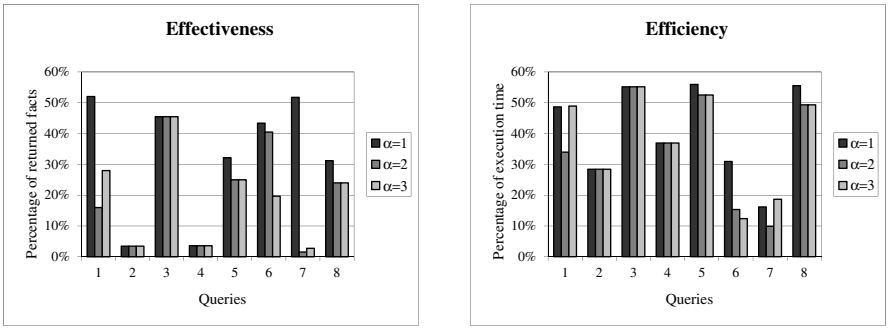
The approach was implemented in Java, using the Mondrian API for handling MDX queries, the Weka implementation of Apriori for rule extraction, and the MYOLAP tool for evaluating preferences [6]. The tests were conducted starting from synthetic MDX logs generated through Algorithm 5, that uses the *Diff* operator proposed in [19]. This operator explores the reasons why an aggregate is significantly lower in one fact compared to another. It takes as parameters two facts f and f' and an integer N, and looks into the two isomorphic sub-cubes C and C' that detail the two facts (i.e., that are aggregated to form f and f'). As a result, it summarizes the differences in these two sub-cubes by providing the top-N informative pairs of cells. Our generator simulates OLAP sessions on a datacube by starting from a random query q and then deriving the subsequent queries in the session using the result of the *Diff* operator applied to q. The Java implementation of *Diff* was obtained from [20]; N is set to 20 to simulate OLAP sessions including no more than 20 queries.

Algorithm 5. Generate a log

Input: $minSize$: Minimum log size
Output: Log: A set of queries
Uses: $Diff(cell, cell)$: The *Diff* operator defined in [19]
Variables: q: A query ; $nbGenerated$: Integer

1: $nbGenerated = 0$
2: **while** $nbGenerated < minSize$ **do**
3: randomly generate a query q on a sub-cube
4: $Log = Log \cup \{q\}$
5: $nbGenerated++$
6: let f_1, f_2 be facts that show the maximum difference in the result of q
7: **for** each pair $\langle f_1', f_2' \rangle \in Diff(f_1, f_2)$ **do**
8: let q' be the drill-down of q to the group-by set of f_1' and f_2'
9: $Log = Log \cup \{q'\}$
10: $nbGenerated++$
11: **return** Log

The architecture used for testing is an Intel Core 2 Duo 3 GHz, with 4GB RAM. All tests were made on the CENSUS schema, using real data extracted from the IPUMS database [21], corresponding to about 10^7 facts stored on Oracle 11g. For our tests, we generated a log of about 1000 queries; the initial query of



(a) (b)

Fig. 2. Effectiveness and efficiency of our approach

each session was generated randomly by selecting group-by sets, measures and selections from a small pool. A small selection pool (3 selections on different dimensions) is used to simulate the log of a single user querying a sub-cube. Then, 8 queries to be personalized were extracted randomly from the log and removed from it. Minimum support and confidence were adjusted with Algorithm 1 to 0.6 and 0.7, respectively, resulting in 20 rules that cover the log and have an average support and confidence of 0.63 and 0.85, respectively. The confidence ranges from 0.76 to 1, with a standard deviation of 0.063.

As to effectiveness, Figure 2.a reports, for each query in the benchmark, the ratio between the number of preferred facts returned by the annotated query (i.e., those included in the *best-match only* result of the query [3]) and the one returned by the original query, when the personalization degree ranges between 1 and 3. Our approach is always effective in reducing the number of facts returned to the user. Though in general the reduction gets stronger as the personalization degree is increased, two different trends are apparent. In some cases (queries 2, 3, and 4) the reduction is independent on the personalization degree since only one pertinent and effective fragment was found. In other cases (queries 1 and 7), as the complexity of the preference increases, there are no facts that fully satisfy it so a larger set of facts that partially satisfy the preference are returned.

As to efficiency, we point out that the log mining step was executed in less than 4 secs, while the time for rule selection and fragment translation never exceeded 5 msecs. Figure 2.b reports the ratio between the time taken to execute each annotated query and the time to execute the original query. The reduction is always above 40%, and it is not relevantly affected by the personalization degree. Overall, we can conclude that our approach to personalization not only puts no overhead on the querying process, but it significantly reduces query response times.

While in this paper we used preference mining for result ranking, in our future work we will attempt to generalize it to address query recommendation as well. Besides, we will investigate the feasibility of extending our approach to incrementally manage OLAP sessions, i.e., to take delta queries into account at runtime without having to mine the log from scratch.

References

1. Giacometti, A., Marcel, P., Negre, E.: Recommending multidimensional queries. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2009. LNCS, vol. 5691, pp. 453–466. Springer, Heidelberg (2009)
2. Bellatreche, L., Giacometti, A., Marcel, P., Mouloudi, H., Laurent, D.: A personalization framework for OLAP queries. In: Proc. DOLAP, Bremen, Germany, pp. 9–18 (2005)
3. Golfarelli, M., Rizzi, S., Biondi, P.: myOLAP: An approach to express and evaluate OLAP preferences. In: IEEE TKDE (to appear 2011)
4. Jerbi, H., Ravat, F., Teste, O., Zurfluh, G.: Management of context-aware preferences in multidimensional databases. In: Proc. ICDIM, London, UK, pp. 669–675 (2008)
5. Jerbi, H., Ravat, F., Teste, O., Zurfluh, G.: Applying recommendation technology in OLAP systems. In: Filipe, J., Cordeiro, J. (eds.) ICEIS. LNBIP, vol. 24, pp. 220–233. Springer, Heidelberg (2009)
6. Biondi, P., Golfarelli, M., Rizzi, S.: Preference-based datacube analysis with myOLAP. In: Proc. ICDE (to appear 2011)
7. Stefanidis, K., Drosou, M., Pitoura, E.: "You May Also Like" results in relational databases. In: Proc. PersDB, Lyon, France (2009)
8. Giacometti, A., Marcel, P., Negre, E., Soulet, A.: Query recommendations for OLAP discovery driven analysis. In: IJDWM (to appear 2011)
9. Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query recommendations for interactive database exploration. In: Winslett, M. (ed.) SSDBM 2009. LNCS, vol. 5566, pp. 3–18. Springer, Heidelberg (2009)
10. Akbarnejad, J., Chatzopoulou, G., Eirinaki, M., Koshy, S., Mittal, S., On, D., Polyzotis, N., Varman, J.S.V.: SQL QueRIE recommendations. PVLDB 3(2), 1597–1600 (2010)
11. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: Snipsuggest: Context-aware autocompletion for SQL. PVLDB 4(1), 22–33 (2010)
12. Veloso, A., de Almeida, H.M., Gonçalves, M.A., Meira Jr, W.: Learning to rank at query-time using association rules. In: Proc. SIGIR, Singapore, pp. 267–274 (2008).
13. Holland, S., Ester, M., Kießling, W.: Preference mining: A novel approach on mining user preferences for personalized applications. In: Lavrač, N., Gamberger, D., Todorovski, L., Blockeel, H. (eds.) PKDD 2003. LNCS (LNAI), vol. 2838, pp. 204–216. Springer, Heidelberg (2003)
14. Mobasher, B.: Data mining for web personalization. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web, pp. 90–135. Springer, Heidelberg (2007)
15. Microsoft: MDX reference (2009), `http://msdn.microsoft.com/`
16. Sá, C., Soares, C., Jorge, A., Azevedo, P., Costa, J.: Mining association rules for label ranking. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part II. LNCS, vol. 6635, pp. 432–443. Springer, Heidelberg (2011)
17. Li, W., Han, J., Pei, J.: CMAR: Accurate and efficient classification based on multiple class-association rules. In: Proc. ICDM, pp. 369–376 (2001)
18. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. VLDB, pp. 487–499. Santiago de Chile, Chile (1994)
19. Sarawagi, S.: Explaining differences in multidimensional aggregates. In: Proc. VLDB, Edinburgh, Scotland, pp. 42–53 (1999)
20. Sarawagi, S.: I3: Intelligent, interactive inspection of cubes (2009), `http://www.cse.iitb.ac.in/sunita/icube/`
21. Minnesota Population Center: Integrated public use microdata series (2008), `http://www.ipums.org`