Recommending Multidimensional Queries

Arnaud Giacometti, Patrick Marcel, and Elsa Negre

Université François Rabelais Tours
Laboratoire d'Informatique
France
{arnaud.giacometti,patrick.marcel,elsa.negre}@univ-tours.fr

Abstract. Interactive analysis of datacube, in which a user navigates a cube by launching a sequence of queries is often tedious since the user may have no idea of what the forthcoming query should be in his current analysis. To better support this process we propose in this paper to apply a Collaborative Work approach that leverages former explorations of the cube to recommend OLAP queries. The system that we have developed adapts Approximate String Matching, a technique popular in Information Retrieval, to match the current analysis with the former explorations and help suggesting a query to the user. Our approach has been implemented with the open source Mondrian OLAP server to recommend MDX queries and we have carried out some preliminary experiments that show its efficiency for generating effective query recommendations.

1 Introduction

Traditional OLAP users interactively navigate a cube by launching a sequence of queries over a datawarehouse, which we call an analysis session (or session for short) in the following. This process is often tedious since the user may have no idea of what the forthcoming query should be [1]. This difficulty might be related to the decline of interactive analysis pointed out in [2].

To better support this process, we proposed in [3] a framework for recommending OLAP queries. The idea is to leverage what the other users did during their former navigations on the cube, and to use this information as a basis for recommending to the user what his forthcoming query could be.

In this paper, we present a significant extension of this work that results in a system for recommending multidimensional queries expressed with MDX [4], the de facto standard. Namely we have changed the core of the framework, that is the distance between queries and the distance between sessions, to better handle the peculiarities of OLAP data. We have adapted our system to deal with real-case cubes and MDX queries. More precisely, our contribution include:

– A measurement of the distance of two MDX queries that leverages the peculiarities of OLAP data,
– A measurement of the distance of two sequences of MDX queries by using Approximate String Matching [5], a technique popular in Information Retrieval,

- A framework for using these measures to search the log of an OLAP server to find a set of sessions matching the current session and generate recommendations,
- An implementation of this approach into a recommender system that fully integrates with the open source Mondrian OLAP engine [6] to recommend MDX queries on the fly during an interactive analysis session,
- Experiments conducted to assess the efficiency and effectiveness of our approach.

The paper is organized as follows: Section 2 briefly reviews related work. A motivating simple example is given in Section 3. Section 4 introduces the distance for comparing two MDX queries, and Section 5 introduces the distance for comparing two analysis sessions. Finally Section 6 completes the description of the recommender system by detailing the algorithm for computing recommendations. Section 7 presents our experimental results. We conclude and discuss future work in Section 8. The proofs of the properties are omitted due to lack of space.

2 Related Work

The only other work we know that proposes to recommend queries for supporting database exploration is that of [7]. Although this work shares some common features with ours, it differs on two important aspects: First it deals only with SQL Select-Project-Join queries and second, the fact that a session is a sequence of queries is not taken into account. To the best of our knowledge, our work is the first work dealing with the problem of recommending multidimensional, especially MDX, queries.

The only work that proposed a framework for anticipating an OLAP query is the work of [8,9]. However the main concern of this work is to prefetch data, not to guide the user based on what other users did. In addition, [8,9] does not deal with MDX queries, and the similarity between queries only relies on the schema of the query (i.e., dimensions and levels) whereas the distance that we use takes the members into account. Finally, a Markov Model is used to predict the forthcoming query, whereas our approach is based on Approximate String Matching [5], a technique popular in Information Retrieval.

To support interactive analysis of multidimensional data, Sarawagi et al. introduced discovery driven analysis of OLAP cube in [10]. This and subsequent work [11,12,1] resulted in the definition of various OLAP operators to guide the user towards unexpected data in the cube or to propose to explain an unexpected result. The main difference with our work is that these operators are applied only on query results and they do not take into account what other users might have discovered.

Computing distances between queries logged by a database server has already been investigated by [13]. In this work, language modeling is used to detect sessions within OLTP query logs. With a different goal (recommending query instead of detecting sessions) our work also proposes a way of calculating a distance between queries where the distance computation takes advantage of the particularities of OLAP queries, like the possibility of navigating multidimensional data by changing the level of detail.

Our work can be seen as a way to integrate OLAP and Information Retrieval (IR) a domain where it is very popular to leverage what the other users did to generate recommendations [14]. Note that there is a recent interest for trying to combine IR and OLAP. For instance, in [15] the authors propose to query a datacube with only a set of keywords. Among the potential answers to the query, only the subcubes that are the most surprising are presented to the user.

3 Example

In this section we illustrate with a simple example the basic idea under our recommender system. Consider an OLAP server used by several users navigating a datacube. In what follows, this cube is a simplified version of the FoodMart datacube (the demo example coming with the open source Mondrian OLAP engine [6]) that is composed of four dimension tables and the Sales fact table, having respectively the following schemas:

- $sch(\textsc{Product}) = \{p\_id, Name, Brand, SubCateg, Category, Family, AllProducts\}$,
- $sch(\textsc{Time}) = \{t\_id, Day, Month, Quarter, Year, AllYears\}$,
- $sch(\textsc{Customer}) = \{c\_id, Name, City, State, Country, AllCustomers\}$,
- $sch(\textsc{Store}) = \{s\_id, Name, City, State, Country, AllStores\}$,
- $sch(\textsc{Sales}) = \{p\_id, t\_id, c\_id, s\_id, Unit\ Sales\}$

Each user can open a session on the server to navigate the cube by launching a sequence of queries. The server logs these sessions, i.e., the sequences of queries launched during each analysis session. Suppose the log contains the three sessions detailed in the appendix. Session s_1 analyzes the sales of alcoholic beverages in the USA, Session s_2 analyzes the sales of milk of the brand "Gorilla" in California, and Session s_3 analyzes the sales of milk and cereals in San Francisco.

Imagine now a new session, called the *current session* (or s_c), is performed by a user. Suppose the user issues the three following queries on the cube, named respectively q_1, q_2 and q_3, to analyze the sales of milk in San Francisco:

```
SELECT {[Store].[All Stores].Children} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}

SELECT {[Store].[All Stores].[USA].[CA].[San Francisco]} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}

SELECT {[Store].[All Stores].[USA].[CA].[San Francisco]} ON COLUMNS,
       {[Product].[All Products].[Drink].[Dairy].[Milk]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
```

The recommender system computes the distance between the current session and each session of the log in order to find those candidate sessions that resemble the current session the most. In our example, suppose that sessions s_2 and s_3 are found the closest to the current session. Intuitively this is because each i^{th} query of s_c is close to the i^{th} of the session s_2 (resp. s_3) and, in the case of s_3, having one more query does not increase the distance a lot.

Among the queries composing these candidate sessions, one must be recommended to the user. Considering that the outcome of a session is very often the result of the last query of this session, the recommender system will compute the distance between the last query of the current session and each last query of the candidate sessions. It will then select as the first recommendation the query that is the closest to the last query of the current session. In our example, this query is q_6 since it is closer to q_3 than q_5.

4 Comparing MDX Queries

In this section, we present our approach for computing a distance between MDX queries. We first begin by giving basic definitions.

4.1 Basic Definitions (Cube, References, Queries)

An n-dimensional *cube* $C = \langle D_1, \ldots, D_n, F \rangle$ is defined as the classical $n+1$ relation instances of a star schema, with one relation instance for each of the n dimensions and one relation instance for the fact table. Given a particular dimension table D_i, the members of the dimension are the values in this table[1]. These members are arranged into a graph H_i (traditionally a hierarchy)[2].

Given an n-dimensional cube $C = \langle D_1, \ldots, D_n, F \rangle$, a cell is a tuple of the fact table F. A cell *reference* (or reference for short) is an n-tuple $\langle r_1, \ldots, r_n \rangle$ where r_i is a member of dimension D_i for all $i \in [1, n]$.

MDX queries are modeled in the following way: Considering that the SELECT and WHERE clauses of an MDX expression define the set of references that the user wants to extract from the cube, we propose to see MDX queries as sets of references, for a given instance of a cube.

Formally, let $C = \langle D_1, \ldots, D_n, F \rangle$ be an n-dimensional cube, M be an MDX expression and for all $i \in [1, n]$, let R_i be the set of members of dimension D_i that is deduced from the SELECT and WHERE clause. The *query* over C that corresponds to M is the set of references $R_1 \times \ldots \times R_n$. In what follows, if q is a query we note $r \in q$ to denote that r is a reference of q.

Example 1. The query q_2 of section 3 corresponds to the following set of references: $\{\langle Drink, alltime, allcustomer, San\ Francisco\rangle, \langle Food, alltime, allcustomer, San\ Francisco\rangle\}$

[1] Note that this definition is done without loss of generality w.r.t the calculated members defined in MDX by the optional WITH MEMBER clause. Indeed, a calculated member is associated with a particular dimension, at a particular level of a hierarchy, and thus it is treated in the following as a regular member.

[2] Flat dimensions (like e.g., a measure dimension) are considered as arranged in a hierarchy as well, where all the members have as common ancestor the root of the hierarchy.

4.2 Distance between References

Given a dimension D with its hierarchy H, the distance between two members m, m' in this dimension is the shortest path [16] from m to m' in H. It is noted: $d_{members}(m, m')$. The distance between references is then defined in the following way from $d_{members}$.

Definition 1. *(Distance between references) Given two references* $r_1 = \langle r_1^1, ..., r_1^n \rangle$ *and* $r_2 = \langle r_2^1, ..., r_2^n \rangle$ *of an n-dimensional cube, the distance between* r_1 *and* r_2 *is:* $d_{references}(r_1, r_2) = \sum_{i=1}^{n} d_{members}(r_1^i, r_2^i)$

Example 2. As an example, consider the two references of query q_2 given in the previous example. These references only differ on the *Product* dimension. As members *Drink* and $Food$ have the same parent in the hierarchy of the dimension *Product*, then $d_{members}(Drink, Food) = 2$. Thus the distance between these two references is 2+0+0+0=2.

4.3 Distance between Queries

As MDX queries are modeled as sets of references, comparing two MDX queries boils down to comparing two sets of references. In our approach we use the classical Hausdorff distance [17] for comparing two sets based on a distance between the elements of the sets. Informally, two sets are close if every element of either set is close to some element of the other set.

Definition 2. *(Hausdorff distance) Given two queries* q_1, q_2*, the distance between* q_1 *and* q_2 *is:*

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

This distance d_h is combined with the distance $d_{dim}(q_1, q_2)$ that gives the number of dimensions where q_1 and q_2 differ (if $q_1 = R_1^1 \times \ldots \times R_n^1$ and $q_2 = R_1^2 \times \ldots \times R_n^2$, D_i is a dimension where q_1 and q_2 differ if $R_i^1 \neq R_i^2$). Thus the distance between queries is defined as the following function of these two distances.

Definition 3. *(Distance between queries) Given two queries* q_1, q_2*, the distance between* q_1 *and* q_2 *is :* $d^{\gamma}_{queries}(q_1, q_2) = \gamma \times d_{dim}(q_1, q_2) + (1 - \gamma) \times d_h(q_1, q_2)$ *where* $\gamma \in [0, 1]$.

Example 3. Consider query q_2 described above and the queries given in the appendix. $q_2 = \{r_1 = \langle Drink, alltime, allcustomer, San\ Francisco \rangle, r_2 = \langle Food, alltime, allcustomer, San\ Francisco \rangle\}$ and $q_2^2 = \{r_3 = \langle Drink, alltime, allcustomer, USA \rangle, r_4 = \langle Food, alltime, allcustomer, USA \rangle\}$. Note that q_2^2 rolls up q_2 from the city level to the country level. Their distance is computed as follows. $d_{references}$ is used to compare r_1 to r_3 and r_4. We have $d_{references}(r_1, r_3) = 2$ and $d_{references}(r_1, r_4) = 4$. The minimum is 2. r_2 is also compared to r_3 and r_4, the minimum being also 2. Thus the maximum of these two rounds of comparison is 2. Now r_3 is compared to r_1 and r_2 and so is r_4. In both cases the minimum is 2. Therefore $d^0_{queries}(q_2, q_2^2) = 2$.

The following property indicates the range of possible values for the distance $d^{\gamma}_{queries}$. The maximal value for this distance is denoted $d^{max}_{queries}$.

Property 1. Given an n-dimensional cube C, the distance $d^{\gamma}_{queries}$ ranges from 0 to $d^{max}_{queries} = \gamma \times n + (1-\gamma) \times 2 \times \sum_{i=1}^{n} h_i$ where h_i is the height of the hierarchy of dimension i.

5 Comparing Analysis Sessions

In this section, we present our approach for comparing two sessions. The basic idea stems from *Approximate String Matching* [5], which we introduce briefly in the following definitions.

5.1 Definitions (Edit Distance, Session, Log)

Given two sequences s_1, s_2, Approximate String Matching is the problem of matching the sequences allowing errors. The matching relies on the computation of a distance between the sequences, which is the minimal cost of the sequences of operations transforming s_1 into s_2. The classical Levenshtein (or edit) distance [18] is commonly used. It allows the following operations: insertions, deletions, substitutions. If the cost associated with each of these operations is 1, this distance can be thought of as the minimal number of insertions, deletions or substitutions to make the two sequences equal.

In our approach, the sequences we consider are sequences of MDX queries which we call analysis sessions (or sessions for short). A log is a set of sessions.

Example 4. Session s_c of Section 3 is the sequence $\langle q_1, q_2, q_3 \rangle$. The log given in appendix is the set $\{s_1, s_2, s_3\}$ and session $s_3 = \langle q_1, q_2^2, q_3, q_6 \rangle$. If insertions, deletions and substitutions are operations allowed on sessions, a sequence of operations that transforms s_c into s_3 is: substitute q_2 by q_2^2 and insert q_6 at the end. If all operations have the same cost 1, then this sequence costs 2. Another sequence that transforms s_c into s_3 is: delete all queries from s_c and insert respectively queries q_1, q_2^2, q_3 and q_6. Obviously the cost of this sequence is not minimal.

5.2 Distance between Sessions

we compute a distance that is the minimal cost of a sequence of operations (called an edit sequence) to transform s_1 into s_2. As in the edit distance the operations permitted are:

- The substitution of a query q_1 by a query q_2. The cost of this operation is the distance between q_1 and q_2 as defined in Definition 2, that is $d^{\gamma}_{queries}(q_1, q_2)$.
- The insertion (resp. deletion) of a query in a sequence. The cost of these operation is a constant α.

An intuitive reason for a fixed cost for insertion (or deletion) is the following. Suppose we want to compute a distance between session $\langle a \rangle$ and session $\langle a, b \rangle$ on the one hand and session $\langle a \rangle$ and session $\langle a, b' \rangle$ on the other hand. There is no reason for distinguishing or favoring the adding of b from the adding of b'. In both cases, a user found these two particular queries of interest, and the sessions are distant from $\langle a \rangle$ only in that a query has been added.

Now, the value for α can range from 0 to $d_{queries}^{max}$. Low values for e.g., insertion allow not to discriminate longer sessions too much. On the other hand, the value should be high enough since it should be more expensive to delete and then insert instead of substituting. Adjusting the value for this cost is part of the experiments described section 7.

Definition 4. *(Distance between sessions) The distance between two sessions s and s' is the minimal cost of all edit sequences transforming s into s'. It is noted $d_{sessions}$.*

The following property states that $d_{sessions}$ is a metric in the mathematical sense.

Property 2. $d_{sessions}$ is a metric in that it satisfies the following properties: non-negativity, symmetry, triangle inequality.

Example 5. Consider the sessions s_c presented in Section 3, and the sessions given in the appendix. Suppose $\gamma = 0$ and α (the cost for inserting or deleting) is $d_{queries}^{max}/2 = 14$. The sequence having minimal cost for transforming s_c into s_1 is: substitute q_2 by q_2^2 and then substitute q_3 by q_4. Substituting q_2 by q_2^2 costs $d_{queries}^0(q_2, q_2^2) = 2$ and substituting q_3 by q_4 costs $d_{queries}^0(q_3, q_4) = 6$ (cf. Example 3). Thus $d_{sessions}(s_c, s_1) = 8$. The sequence having minimal cost for transforming s_c into s_2 is: substitute q_2 by q_3^2 and then substitute q_3 by q_5. Its cost is: $d_{sessions}(s_c, s_2) = 4$ (for substituting q_2 by q_3^2) $+3$ (for substituting q_3 by q_5). The sequence having minimal cost for transforming s_c into s_3 is the first one given in Example 4. Its cost is: $d_{sessions}(s_c, s_3) = 2$ (for substituting q_2 by q_2^2) $+14$ (for inserting q_6).

6 The Recommender System

In this section, we present how we use the distances defined above to recommend MDX queries. The principle is the following: The log is searched for candidate sessions matching the current session. From these candidate sessions a set of recommended queries is obtained. These recommended queries are ranked and presented to the user as recommendations in the resulting order. The best ranked queries are called the best recommendations.

Before detailing the algorithm we introduce the following definitions. The candidate sessions are the closest to the current session in the sense of the distance between sessions.

Definition 5. *(Candidate sessions) Given a set L of sessions and a session s_c, the set of candidate sessions is defined by $Cand_{sessions}(s_c, L) = \{s \in L | \nexists s' \in S, d_{sessions}(s', s_c) < d_{sessions}(s, s_c)\}$*

To define the recommended queries, we use an analogy with Web search, where it has been shown that what is seen at the end of a session can be used to enhance further searches [19]. Indeed, even in our case, it makes sense to consider that if the session ended on this particular query, it is because the user found something of interest. We adopt this point of view and simply define a recommended query to be the last query of a candidate session. The best recommendations are the recommended queries that are the closest to the last query of the current session, in the sense of the distance between queries.

Definition 6. *(Recommended queries and best recommendations) Given a set L of sessions and a session s_c, the set of recommended queries is defined by $Reco_{queries}(s_c, L) = \{last(s) | s \in Cand_{sessions}(s_c, L)\}$ where $last(s)$ is the last query of session s. Given a set C of recommended queries and a session s_c, the best recommendations are: $best(s_c, C) = \{q \in C | \nexists q' \in C,\ d^{\gamma}_{queries}(q', last(s_c)) < d^{\gamma}_{queries}(q, last(s_c))\}$*

Note that changing these definitions can have an important impact on the subjective quality of the recommendations. Assessing this is part of our long-term goal as discussed in conclusion.

Finally the algorithm for recommending MDX queries is:

```
Input: A current session $s_c$ and a log $L$
Output: A sequence of recommendations
 1. Generate the set $C$ of recommended queries $C = Reco_{queries}(s_c, L)$
 2. Let Output be the empty sequence
 3. Repeat until $C$ empty
    (a) Generate the best recommendations $best(s_c, C)$
    (b) Append $best(s_c, C)$ to Output
    (c) Remove $best(s_c, C)$ from $C$
```

Example 6. Consider the distances computed in Example 5. There is only one candidate session which is s_2 since it is the closest to s_c. Thus there is only one candidate query q_5 which is then the recommendation. Suppose now the cost of the insertion operation used to compute the distance between sessions is 5. This means that there are now two candidate sessions s_2 and s_3. The candidate queries are q_5 and q_6. The query recommended first is q_6 since it is closer to q_3 than q_5.

7 Experiments

In this section, we present the results of the experiments we have conducted to assess the capabilities of our framework. We used synthetic data produced with our own data generator. Both our prototype for recommending queries and our generator are developed in Java using JRE 1.6.0_13. All tests are conducted with a Core 2 Duo - E4600 with 4GB of RAM using Linux CentOS5.

7.1 Data Set

We generated a set of sessions over the test database FoodMart supplied with the Mondrian OLAP engine [6]. Each session is generated in the following way: The first query of the session is selected by random among the 15 example queries supplied by Mondrian. Each subsequent queries is generated by choosing randomly one dimension and applying on the preceding query an OLAP operation (rollup, drilldown, changing the set of members) on this dimension. Our generator uses the following parameters: A number (X) of sessions in the log, a maximum number (Y) of queries per session. In our tests, we fixed the maximum number of references at 100 since it is reasonable to consider that users will very seldom produce a cross table larger than 10×10 as the answer to an MDX query.

7.2 Results

Note that, due to lack of space we have not included all the results of the experiments we have conducted.

7.2.1 Performance Analysis

Our first experiment assesses the time taken to generate the best recommendation for various log sizes. The performance is presented in Figure 1 according to various log sizes. These log sizes are obtained by multiplying parameters X (number of sessions) and Y (maximum number of queries per session). X ranges from 25 to 500 and Y ranges from 20 to 50. We thus obtain logs of size varying between 150 and 25000 queries. The best recommendation is computed for each of these logs, for current sessions of various sizes, generated with the session generator.

Figure 1 shows that the time taken to generate one recommendation increases linearly with the log size but remains highly acceptable and is slightly influenced by the current session size. Indeed, to recommend a query for a session s, the

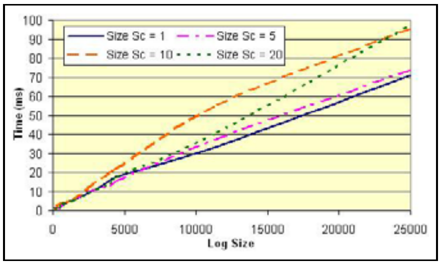


Fig. 1. Performance analysis

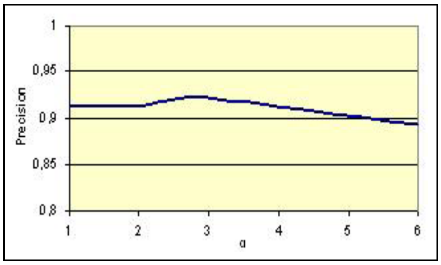


Fig. 2. Precision for various α (cost of insertion or deletion)

system only compares $last(s)$ to each query of the log and uses the distances previously computed for $s \setminus last(s)$.

7.2.2 Precision/Recall Analysis

We use a 10-fold cross validation to assess our framework in the spirit of the experimental validation done in [7]. The generated set of sessions is partitioned in 10 equally sized subsets and in each run 9 subsets are used as log and each session of the remaining subset is used as a basis for the current session. More precisely for each such session s_c of size n, we use the sequence of the first $n-1$ queries as the current session, and we compute the recommendations for the n-th query. The n-th query of s_c is called the expected query and is noted q_{ex}.

We evaluate the precision and recall [20] of the recommendations using the following metrics: precision=$|members(q_{ex}) \cap members(q_{rec})|/|members(q_{rec})|$ and recall=$|members(q_{ex}) \cap members(q_{rec})|/|members(q_{ex})|$, where $members(q)$ is the set of members of query q, q_{rec} is a recommended query and q_{ex} is the expected query. For each session, we report the maximum recall over all the recommended queries and the precision for the query achieving this maximum recall.The log generated for these tests has size 5877 queries (750 sessions).

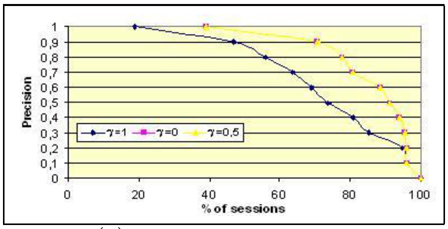
Figures 3 and 4 show the inverse cumulative frequency distribution (inverse CFD) of the recorded precision, recall and/or F-measure[3] for the sessions. A point (x, y) in these graphes signifie that $x\%$ of sessions had precision or recall or F-measure $\geq y$.

The first experiments allow us to tune our system in order to choose for α and γ the values that achieve best precision and recall. Precision is computed for α which is the cost of the insertion (or deletion) operation (see Section 5). Figure 2 shows that a precision above 0.9 is obtain for $\alpha \in [1, 5]$. In the subsequent experiment, the value for α is 2. Precision and recall are computed for $\gamma = 0, 0.5$ or 1. Figure 3(a) and Figure 3(b) show that the worse results are obtained for $\gamma = 1$ i.e., when the distance between queries only counts the number of dimensions that differ (see Section 4). For 0 and 0.5 the curves are confounded. This shows that for $\alpha = 0.5$ d_{dim}, ranging only from 0 to n (see Property 1), contributes for nothing to the distance between queries. Thus in what follows, $\gamma = 0$.

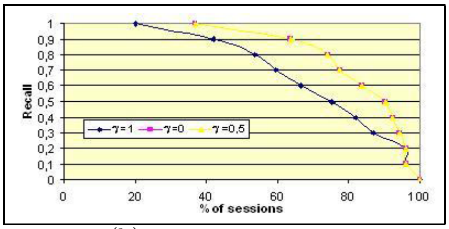
Figure 4 shows the inverse CFD of precision, recall and F-measure of the recommendations computed with our system for $\alpha = 2$ and $\gamma = 0$. The results demonstrate the effectiveness of our method since for around 80 % of the sessions, precision and recall are above 0.8. These good results can be explained by the density of the log generated, considering the relatively small number of queries (15) in the pool we used for seeding the generation.

Figure 5 displays the inverse CFD of the recorded F-measure for the sessions for various methods for recommending MDX queries. The first method, called *ClusterH*, is the one proposed in [3] that uses a k-medoid clustering

[3] The F-measure, $F = 2.(\text{precision} \cdot \text{recall})/(\text{precision} + \text{recall})$, is a measure of a test's accuracy.

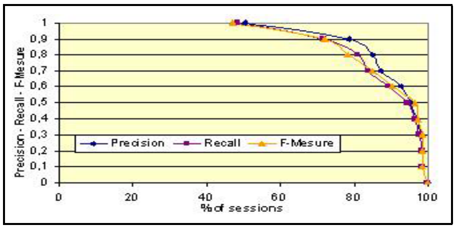


(a) Precision for various γ



(b) Recall for various γ

Fig. 3. Precision and Recall of the recommendations (for various γ)



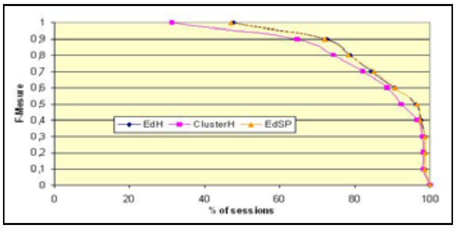


Fig. 4. Precision, Recall and F-Mesure of the recommendations ($\alpha = 2, \gamma = 0$)

Fig. 5. F-Mesure of the recommendations ($\alpha = 2, \gamma = 0$) for the 3 possible methodes

algorithm with a simple Hamming distance to compare references. The second method, called *EdSP* (Edit Distance with Shortest Path), is the one proposed in the present paper for $\alpha = 2$ and $\gamma = 0$. Finally the last method called *EdH* combines the Edit distance with the simple Hamming distance for comparing references. First we note that all methods achieve good results for our dense log. The method using a clustering algorithm performs slightly bad compared to the two others. It can also be seen that *EdSP* and *EdH* perform similarily, which may seem surprising at first since the Hamming distance for comparing references is coarse compared to the Shortest Path. However, it is to be noted that the way we compute precision and recall (inspired by [7]) favors *EdH*. Indeed, if *EdH* recommends a query close (in the sense of the Hamming distance) to the expected query it will have good precision and recall. But if *EdSP* recommends a query close (in the sense of the Shortest Path) to the expected query it can have bad precision and recall. Therefore it turns out that *EdSP* performs as well as *EdH* even though it is not favored by the computation of precision and recall.

8 Conclusion and Future Work

In this paper, we present a system for recommending MDX queries that is an evolution of the framework presented in [3]. Our framework leverages former navigations on a datacube and is based on two distances that we propose to compare MDX queries and analysis sessions. Our approach is implemented in a system that integrates with the open source Mondrian OLAP engine to recommend MDX queries on the fly. The experiments we have conducted show that recommendations can be computed on the fly efficiently and that our system can be tuned to obtain objectively good recommendations.

Our long term goal is to design a platform for generating MDX recommender systems by giving the user the possibility to adapt the approach to his/her needs. This can be done by proposing to the user various methods for computing candidate sessions and/or candidate queries. We are working on the definition of a new method that takes into account the measures' values and not only the references of the cells. A combination of the recommender system with techniques for OLAP query personalization [21] is also under consideration.

To fulfill this goal, we need to undertake experiments on real data sets with feedback from users. This will allow not only to improve the overall quality of the recommended queries but also to determine to which context a particular approach for computing candidate recommendations is adapted.

On the technical side, we need to propose an indexing method for organizing the log in order to make the search in the log even more efficient, and thus making it possible to search very large log files on the fly.

References

1. Sarawagi, S.: User-adaptive exploration of multidimensional data. In: VLDB, pp. 307–316 (2000)
2. Pedersen, T.B.: How is BI used in industry?: Report from a knowledge exchange network. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 179–188. Springer, Heidelberg (2004)
3. Giacometti, A., Marcel, P., Negre, E.: A framework for recommending olap queries. In: DOLAP, pp. 73–80 (2008)
4. Microsoft Corporation: Multidimensional expressions (MDX) reference (2008), `http://msdn.microsoft.com/en-us/library/ms145506.aspx`
5. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. 33(1), 31–88 (2001)
6. Pentaho Corporation: Mondrian open source OLAP engine (2009), `http://mondrian.pentaho.org/`
7. Chatzopoulou, G., Eirinaki, M., Polyzotis, N.: Query recommendations for interactive database exploration. In: SSDBM, pp. 3–18 (2009)
8. Sapia, C.: On modeling and predicting query behavior in OLAP systems. In: DMDW, pp. 2.1–2.10 (1999)
9. Sapia, C.: PROMISE: Predicting query behavior to enable predictive caching strategies for OLAP systems. In: Kambayashi, Y., Mohania, M., Tjoa, A.M. (eds.) DaWaK 2000. LNCS, vol. 1874, pp. 224–233. Springer, Heidelberg (2000)
10. Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-driven exploration of OLAP data cubes. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 168–182. Springer, Heidelberg (1998)
11. Sarawagi, S.: Explaining differences in multidimensional aggregates. In: VLDB, pp. 42–53 (1999)

12. Sathe, G., Sarawagi, S.: Intelligent rollups in multidimensional OLAP data. In: VLDB, pp. 531–540 (2001)
13. Huang, X., Yao, Q., An, A.: Applying language modeling to session identification from database trace logs. Knowl. Inf. Syst. 10(4), 473–504 (2006)
14. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng. 17(6), 734–749 (2005)
15. Wu, P., Sismanis, Y., Reinwald, B.: Towards keyword-driven analytical processing. In: SIGMOD Conference, pp. 617–628 (2007)
16. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
17. Hausdorff, F.: Grundzüge der Mengenlehre. von Veit (1914)
18. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8 (1966)
19. White, R.W., Bilenko, M., Cucerzan, S.: Studying the use of popular destinations to enhance web search interaction. In: SIGIR, pp. 159–166 (2007)
20. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press/Addison-Wesley (1999)
21. Bellatreche, L., Giacometti, A., Marcel, P., Mouloudi, H., Laurent, D.: A personalization framework for olap queries. In: DOLAP, pp. 9–18 (2005)

A Appendix: A Toy Query Log

Session $s_1 = \langle q_1, q_2^2, q_4 \rangle$: Sales of alcoholic beverages in the USA

```
SELECT {[Store].[All Stores].Children} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA]} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA].Children} ON COLUMNS,
       {[Product].[All Products].[Drink].[Alcoholic Beverages]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
```

Session $s_2 = \langle q_1, q_3^2, q_5 \rangle$: Sales of milk of the brand "Gorilla" in California

```
SELECT {[Store].[All Stores].Children} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA].[CA].[San Francisco]} ON COLUMNS,
       {[Product].[All Products].[Drink].[Dairy].[Milk].[Gorilla].Children,
       [Product].[All Products].[Drink].[Dairy].[Milk].[Gorilla]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA].[CA],
       [Store].[All Stores].[USA].[CA].Children} ON COLUMNS,
       {[Product].[All Products].[Drink].[Dairy].[Milk].[Gorilla].Children,
       [Product].[All Products].[Drink].[Dairy].[Milk].[Gorilla]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
```

Session $s_3 = \langle q_1, q_2^2, q_3, q_6 \rangle$: Sales of milk and cereals in San Francisco

```
SELECT {[Store].[All Stores].Children} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA]} ON COLUMNS,
       {[Product].[All Products].[Food], [Product].[All Products].[Drink]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA].[CA].[San Francisco]} ON COLUMNS,
       {[Product].[All Products].[Drink].[Dairy].[Milk]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
SELECT {[Store].[All Stores].[USA].[CA].[San Francisco]} ON COLUMNS,
       {[Product].[All Products].[Drink].[Dairy].[Milk],
       [Product].[All Products].[Food].[Breakfast Foods].[Breakfast Foods].[Cereal]} ON ROWS
FROM   [Sales]
WHERE  {[Measures].[Unit Sales]}
```