Cold-Start recommender system problem within a multidimensional data warehouse

Elsa Negre, Franck Ravat, Olivier Teste, Ronan Tournier

Cold-Start Recommender System Problem Within a Multidimensional Data Warehouse

Elsa Negre*, Franck Ravat†, Olivier Teste‡ and Ronan Tournier†

*Université Paris-Dauphine, LAMSADE, Place du Maréchal de Lattre de Tassigny, F-75016 Paris, France
Email: Elsa.Negre@dauphine.fr

†Université Toulouse 1 Capitole, IRIT-SIG, 2 rue du doyen Gabriel Marty, 31042 Toulouse Cedex 9, France
Email: Franck.Ravat@irit.fr; Ronan.Tournier@irit.fr

‡Université Toulouse 2 / IUT Blagnac, IRIT-SIG, 1 place Georges Brassens, BP 60073, 31703 Blagnac Cedex, France
Email: Olivier.Teste@irit.fr

***Abstract*—Data warehouses store large volumes of consolidated and historized multidimensional data for analysis and exploration by decision-makers. Exploring data is an incremental OLAP (On-Line Analytical Processing) query process for searching relevant information in a dataset. In order to ease user exploration, recommender systems are used. However when facing a new system, such recommendations do not operate anymore. This is known as the cold-start problem. In this paper, we provide recommendations to the user while facing this cold-start problem in a new system. This is done by patternizing OLAP queries. Our process is composed of four steps: patternizing queries, predicting candidate operations, computing candidate recommendations and ranking these recommendations.**

I. Introduction and context

Decision support systems allow decision-makers to gain insight into corporate or organization data by analyzing aggregated historical business or scientific data [1]. These systems generally rest on a centralized storage system: a data warehouse [2]. Users (i.e. decision-makers) explore and analyze the data warehouse content by using On-Line Analytical Processing (OLAP). However, users face an increasing volume of information due to the computing capacity and storage increases [3]. Exploring a vast data warehouse can be very tedious and a user can easily get lost.

Thus, computer techniques easing this search and extraction of relevant information are needed. One of them is recommendation, especially in the case of exploring large volumes of data where the information access paradigm is based on query expression. The recommendation process will guide the user during his/her exploration of the volume of available information by searching for him/her, the information that appears relevant.

Recommender systems are known for their use in e-commerce [4], [5], where they advise a customer on choosing an item based on his/her preferences. However, some works focused on recommendation in the field of databases [6], [7], [8] and proposed methods and algorithms to assist the user. There is also research in the field of data warehouses analyzed by OLAP queries. Among these (see [9] and [10] for detailed study), some focused on exploiting user profiles and preferences [11], [12], [13], others on the discoveries made during analyses [14], [15] as well as on exploiting logs containing sequences of queries previously run by other users on the same cube [16], [6], [17], [18], [19], [20]. More recently, [21] proposes a multidimensional algebra for describing analytical sessions.

Although these approaches allow recommending relevant queries for a given user, they are rapidly blocked, e.g. by updates of the data warehouse or when cubes and/or decision-makers are different. Missing information about a new item or/and a new user is called in the literature the cold-start problem. This problem is encountered when recommendations are required for items and/or users for whom we have no information yet, either explicitly or implicitly [22]. [23] has identified several types of cold-start problems: new user problem [24], [25], new item problem [22], [26], non-transitive association and overfitting.

In the data warehouse context, we face the cold-start problem when the system intents to make recommendations for a new cube. For example, for a given data warehouse containing data up to the year 2010, a data cube C_1 on years 2009 and 2010 and a log of queries launched on C_1, suppose that the data warehouse (DW) has been updated with the 2011 data and a new cube C_2 on years 2010 and 2011 is created, the system will recommend to the first decision-makers who will explore C_2, very few queries, and maybe none.

This problem is more complex than for the Web area because we face a whole set of new items (a new cube) and new users (new decision-makers or previous decision-makers with a new function), i.e. we face what we call a new system. Therefore, we must go beyond the recommendation and try to suggest to these decision-makers queries that have not yet been launched on a given cube. In the DW field, to the best of our knowledge, there is no research on cold-start problems for new systems.

The paper is organized as follows: Section II motivates our proposal with a toy example, Section III presents our conceptual model. Section IV details our cold-start process and Section V concludes our presentation.

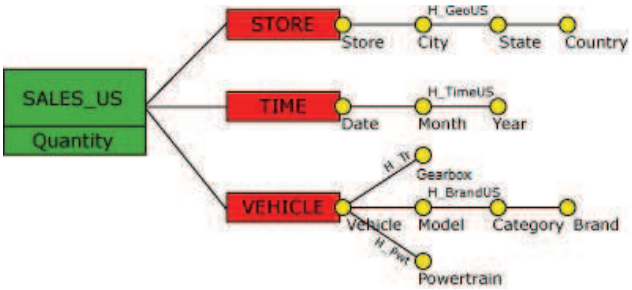

Fig. 1. Data warehouse for analyzing US vehicle sales from 2009 to 2011.

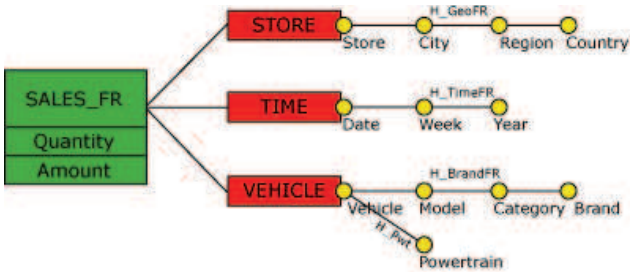

Fig. 2. Data warehouse for analyzing French vehicle sales from 1998 to 2008.

II. Motivating example

Description of the example

Our example concerns a newly hired decision-maker in a vehicle manufacturer company. His task will consist in developing a reseller network in a new country (northern America). The vehicle manufacturer was able to assemble a data warehouse using publicly available US vehicle sales data. This new data warehouse was designed similarly compared to the current data warehouse of the manufacturer. The current system allows the analysis of vehicle sales in France during the last decade ($SALES\_FR$ star schema illustrated Figure II), whereas the new system monitors car sales done in the US, but only during the past two years (the $SALES\_US$ star schema illustrated Figure 1). Both data warehouses have the same fact and the same dimensions. Note that in both schemas, all dimension hierarchies end by an ALL level (All^{Store}, All^{Time} and $All^{Vehicle}$), not displayed in the figures. However, when taking a closer look to both systems, the measures of the facts and the levels of the dimensions are not exactly the same. In France, the quantity of vehicles sold is recorded as well as the amount of money these sales represented. In the US, only quantities are available. In France, cities are grouped into regions whereas in the US they are grouped in states. Sales are grouped weekly in France and monthly in the US. Vehicles have different gearboxes (manual, automatic, etc.) for the US, whereas in France, all vehicles sold have a manual gearbox. Moreover, data within these star schemas are different. The geographic region monitored corresponds either to France or to northern America. The time dimension data spans over ten years (1998-2008) for the French system and only two years (2009-2011) for the US system. Although vehicle brands are identical, most models are different and some categories are not identical (e.g. in France, there are small and low consumption city cars). Finally, in France, the powertrain of the vehicles is either forward wheel drive or 4x4 while in the US there is also rear wheel drive.

Log session of the French star schema

The French system has been used for several years. From this system query session logs can be extracted. These correspond to the successive manipulations (OLAP queries) performed by the users on this system. In our example (see the log extract in Table I), a French user starts by (q_3), that analyses the sale quantities for all vehicles for the country France during the year 2007. He/she then "zooms into" the geographic location of the sales. Thus, in two steps, this user drills down from the Country level to firstly the Region level (q_4) and secondly to the City level (q_5), hence finally monitoring the vehicle quantities sold for all French cities (see the top part of the Table I).

log sessions				
	$Sales\_FR$	Vehicle	Store	Time
q_3	Quantity	$All^{Vehicle}$	France (Country)	2007 (Year)
$q_3 \rightarrow q_4$	$Identity$	$Identity$	Drilldown	$Identity$
q_4	Quantity	$All^{Vehicle}$	North (Region)	2007 (Year)
$q_4 \rightarrow q_5$	$Identity$	$Identity$	Drilldown	$Identity$
q_5	Quantity	$All^{Vehicle}$	North cities (City)	2007 (Year)
current session				
	$Sales\_US$	Vehicle	Store	Time
q_1	Quantity	$All^{Vehicle}$	US (Country)	2011 (Year)
$q_1 \rightarrow q_2$	$Identity$	$Identity$	Drilldown	$Identity$
q_2	Quantity	$All^{Vehicle}$	Texas (State)	2011 (Year)
$q_2 \rightarrow q_{pred}$	$Identity$	$Identity$	Drilldown	$Identity$
q_{pred}	Quantity	$All^{Vehicle}$	Texas cities (City)	2011 (Year)

TABLE I. Example of an analysis session log extracted from a log (top part) compared to an ongoing analysis session.

Recommendation for the US star schema exploration

The US decision support system is brand new. It is thus impossible to get query logs from this system to recommend queries that were executed previously to the new user. Moreover, our user is a new employee of the company; he has experience neither with the domain of the company data (vehicle sales), nor with the company's decision-maker's analysis practices. Nevertheless, on the one hand, analysis practices could be suggested to him by exploiting the usage of the French decision support system by the French analysts. However, both star schemas are not identical and the data is different. On the other hand, although company data exists for French sales, the US vehicle selling activity is bran new. Thus the new decision-maker can be helped neither by the existing company system nor by users. The consequence is a need for providing data warehouse exploration guidance to the new user but without taking into account the (new) data. For example, our new user performs an analysis of vehicle sales quantities for all vehicles for the US during the year 2011 (q_1 in the bottom part of the current session log illustrated in Table I). If he then drills down on the store location from the country level to the state level (q_2), the system could suggest him a "French" way of exploration, i.e. when analyzing quantities of all the products sold for a specific year the French perform a double drill-down on the geographical hierarchy (q_{pred}).

Recommending high level operations

Typically, in our example, these navigation suggestions have to be independent of the data (2011 sales for the US data warehouse while it was 2007 sales in the French version).

Although both star schemas are slightly different, they both have a "sales" fact, a dimension with a "geographical" location, a dimension representing time and another one representing sold products (vehicles). Although these dimensions do not have the same hierarchical structure, they are quite similar, thus exploration queries can be handled in a similar way on both schemas. However, to do this, the recommendations must take into account only high level operations (such as drilling down twice on a time hierarchy or rotating from a geographical hierarchy to a product hierarchy). Such operations rest on OLAP algebraic operators. Moreover, these recommendations must not take into account data as these may differ significantly from one schema to another. Recommendations based on operations rather than values can be useful for the cold-start problem of recommender systems.

III. Conceptual Modeling

Our approach is located at a conceptual level. Using the conceptual level we get rid of all implementation constraints.

A. Multidimensional schema and cube

Let us define F and D such as:

- $F = \{F_1, ..., F_n\}$ is a finite set of facts, $n \geq 1$,
- $D = \{D_1, ..., D_m\}$ is a finite set of dimensions, $m \geq 2$,

Definition 1. *A cube schema, denoted C_i, is defined by $(F_i, Star(F_i))$, such as:*

- *$F_i \in F$ is a fact,*
- *$Star(F_i) = \left\{ D^{n^{D_j}} \mid \forall j \in [1..m], D^{n^{D_j}} \in D \right\}$ is the set of the dimensions according to which it is relevant to analyze the fact F_i.*

A cube schema organizes data according to subjects of analysis, called facts, and axes of analysis, called dimensions. Dimensions are usually organized as hierarchies, supporting different levels of data aggregation. Let us define $\mathcal{N} = \{n_1, n_2, ...\}$ a finite set of non-redundant names.

Definition 2. *A fact, denoted F_i, is defined by (n^{F_i}, M^{F_i}), where:*

- *$n^{F_i} \in \mathcal{N}$ is the name that identifies the fact,*
- *$M^{F_i} = \{m_1, ..., m_{p_i}\}$ is a set of measures.*

Definition 3. *A dimension, denoted D_i, is defined by $(n^{D_i}, A^{D_i}, H^{D_i})$, where:*

- *$n^{D_i} \in \mathcal{N}$ is the name that identifies the dimension,*
- *$A^{D_i} = \{a_1^{D_i}, ..., a_{r_i}^{D_i}\}$ is the set of the attributes of the dimension,*
- *$H^{D_i} = \{H_1^{D_i}, ..., H_{s_i}^{D_i}\}$ is a set of hierarchies.*

Hierarchies organize the attributes of a dimension, called parameters, from the finest graduation to the most general graduation. A hierarchy defines valid navigation paths on an axis of analysis.

Definition 4. *A hierarchy, denoted H_j (abusive notation of $H_j^{D_i}, \forall i \in [1..m], \forall j \in [1..s_i]$) is defined by $(n^{H_j}, P^{H_j}, \prec^{H_j})$, where:*

- *$n^{H_j} \in \mathcal{N}$ is the name that identifies the hierarchy,*
- *$P^{H_j} = \{p_1^{H_j}, ..., p_{q_j}^{H_j}\}$ is a set of attributes called levels, $P^{H_j} \subseteq A^{D_i}$,*
- *$\prec^{H_j} = \{(p_x^{H_j}, p_y^{H_j}) \mid p_x^{H_j} \in P^{H_j} \wedge p_y^{H_j} \in P^{H_j}\}$ is an antisymmetric and transitive binary relation between parameters. Remember that the antisymmetry means that $(p_{k_1}^{H_j} \prec^{H_j} p_{k_2}^{H_j}) \wedge (p_{k_2}^{H_j} \prec^{H_j} p_{k_1}^{H_j}) \Rightarrow p_{k_1}^{H_j} = p_{k_2}^{H_j}$ while the transitivity means that $(p_{k_1}^{H_j} \prec^{H_j} p_{k_2}^{H_j}) \wedge (p_{k_2}^{H_j} \prec^{H_j} p_{k_3}^{H_j}) \Rightarrow p_{k_1}^{H_j} \prec^{H_j} p_{k_3}^{H_j}$.*

In the rest of this paper we will use simple notations for describing hierarchies such as $H_j = (n^{H_j}, P^{H_j}, \prec^{H_j}) = (n^{H_j}, path^{H_j})$ where $path^{H_j} = \langle Id^{D_i}, ..., All^{D_i} \rangle$ is an ordered set of attributes from the root parameter to the extremity parameter.

Example
Figure 1 shows an example of a cube $(F_1, Star(F_1))$ that allows analyzing US vehicle sales from 2009 to 2011. This cube is noted using graphical notations [27]. It is formally defined as follows.

- $F_1 = ('SALES\_US', Quantity)$,
- $Star(F_1) = \{D^{STORE}, D^{TIME}, D^{VEHICLE}\}$, where
  - $D^{STORE} = ('Store', \{Store, City, State, Country\}, \{('H\_GeoUS', \langle Store, City, State, Country \rangle)\})$,
  - $D^{TIME} = ('Time', \{Date, Month, Year\}, \{('H\_TimeUS', \langle Date, Month, Year \rangle)\})$,
  - $D^{VEHICLE} = ('Vehicle', \{Vehicle, Model, Category, Brand, Trim, Powertrain\}, \{('H\_Tr', \langle Vehicle, Trim \rangle), ('H\_BrandUS', \langle Vehicle, Model, Category, Brand \rangle), ('H\_Pwt', \langle Vehicle, Powertrain \rangle)\})$.

B. OLAP analysis

OLAP systems offer capabilities to interactively analyze data by applying a set of specialized operations, such as drill-down, roll-up and slice-and-dice [27]. It has been recognized that the workload of an OLAP application can be characterized by the user's navigational data analysis task: the user defines a first query then successively manipulates the results applying OLAP operations. An OLAP analysis consists in exploring interactively the multidimensional data warehouse. The user performs a sequence of OLAP operations in order to find relevant data for decision making.

A current visualization is a data cube representation, displaying a fact and detailed information of all dimensions.

Within a cube both structures and values are displayed. In the context of our research, the term query pattern refers structures that are displayed in a given instant of the analysis. We model the query pattern through a set of multidimensional structures of displayed levels and measures.

Definition 5. *A query pattern P_{q_i} is defined by $(M_k, Level)$, where:*

- *$M_k \in M^{F_i}$ is a measure of a fact F_i,*
- *$Level : Star(F_i) \rightarrow P$ is a function that associates each dimension to a set of parameters.*

Example
Figure 5 shows examples of query patterns that are formally defined as follows:

- $P_{q_3} : (\{Quantity\}, \{Level(Vehicle) = All^{Vehicle}, Level(Store) = Country, Level(Time) = Year\})$
- $P_{q_4} : (\{Quantity\}, \{Level(Vehicle) = All^{Vehicle}, Level(Store) = Region, Level(Time) = Year\})$
- $P_{q_5} : (\{Quantity\}, \{Level(Vehicle) = All^{Vehicle}, Level(Store) = City, Level(Time) = Year\})$

We define user analysis by graphs where edges represent the OLAP operations and nodes represent results. Each node is defined by a query pattern, which describes the structure elements of the resulting multidimensional table. Each edge is defined by an OLAP operator. There is no consensus on a set of operations for a multidimensional algebra. Table II describes OLAP operators. Note that we restrict the set of operators to a kernel of operators that transform the multidimensional table structures. For more details see [27].

The operator DISPLAY defines sessions. A session begins by the DISPLAY operator and ends when a new DISPLAY is triggered.

Definition 6. *A patternized session is defined as a graph (V, E) where:*

- *V = $\{P_{q_1}, P_{q_2}, P_{q_3}...\}$ is a set of query patterns,*
- *E = $\{(Op_k, P_{q_{k+1}}) \mid Op_k$ is an OLAP operator and $P_{q_{k+1}} \in V\}$*

Example
Figure 3 shows an extract of session formally defined as:

- V = $\{...P_{q_3}, P_{q_4}, P_{q_5}...\}$,
- E = $\{...(DrillDown(P_{q_3}; Store; Region), P_{q_4}), (DrillDown(P_{q_4}; Store; City), P_{q_5})...\}$

IV. Cold-start process

IIn this section we detail the cold-start process in our OLAP query recommender system. The process uses both the sequence of queries of the current session, and the query log of an OLAP server. This log contains the sequences of queries formerly launched on another cube similar to the cube on which are launched the queries of the current session. Right now, we suggest the use of the work of [28] on cube interoperability to detect similar cubes. But the notion of similarity between data cubes will be defined in our future work.

As illustrated in Figure IV, our process consists of the four following steps:

1) Patternizing queries (i.e. obtaining the patterns of the queries) of the former log.
2) Predicting candidate operations by using the patternized current session and the set of patternized log sessions and a match function between the pattern of the current session and the set of patternized sessions.
3) Computing the candidate recommendations by combining one query of the current session and the candidate operations.
4) Ranking the candidate recommendations.

The following subsections detail these steps.

A. Patternizing the query log

Initially, each session of the former log is split into patterns of queries. To this end, this step uses a query patternizing function (see algorithm 1). This function outputs a set of sets of query patterns, that we call a set of session patterns. The principle is straightforward: For each session of the log, replace in the session each query with its query pattern using extractQueryPattern (see algorithm 2) and detail the operations allowing to pass from a query to the next in the session (see extractOp function, detailed in algorithm 3). The patternized current session can be computed as well.

Algorithm 1 PatternizingQueryLog(L, C_L)

Require:
L: The former query log, as a set of sessions,
C_L: The cube on which the log queries have been launched.
Ensure: a set of session patterns (SP)

$V \leftarrow \emptyset$ // *for the operations*
$E \leftarrow \emptyset$ // *for the query patterns*
$SP \leftarrow \langle E, V \rangle$
for each session $s_i \in L$ **do**
  for each tuple of queries $\langle q_j, q_{j+1} \rangle \in s_i$ **do**
    $SP.E \leftarrow SP.E \cup extractQueryPattern(q_j, C_L)$
    // extractQueryPattern: A function extracting the pattern of a given query.
    $SP.E \leftarrow SP.E \cup extractQueryPattern(q_{j+1}, C_L)$
    $SP.V \leftarrow SP.V \cup extractOp(q_j, q_{j+1}, C_L)$
    // extractOp: A function extracting the operation allowing to pass from a query to another in a given session.
  end for
end for
return SP

Algorithm 2 shows how query patterns can be computed while algorithm 3 details how operations between two queries are identified.

Extracting query patterns (see algorithm 2) consists in extracting the references (set of attribute values) of a given query and returning the corresponding level name on each dimension.

OLAP operators	Operation descriptions
$Display(F_{NEW}, f_i(M_{NEW}), \{D_{SHOW}, H_{SHOW}, P_{SHOW}\}) = q_{RES}$	Displaying a fact F_{NEW} and its aggregated measure $f_i(M_{NEW})$ according to the dimensions $\{D_{SHOW}\}$. Non-specified displaying levels are the extremity levels ALL.
$Pivot(q_{SRC}, D_{CUR}, D_{NEW}, H_{NEW}, P_{NEW}) = q_{RES}$	Changing an analysis axis D_{CUR} by a new dimension D_{NEW} at level P_{NEW}
$DrillDown(q_{SRC}, D_{CUR}, P_{NEW}) = q_{RES}$	Displaying the data with a finer level of detail P_{NEW}
$RollUp(q_{SRC}, D_{CUR}, P_{CUR}) = q_{RES}$	Displaying the data with a coarser level of detail P_{CUR}

TABLE II. OLAP ALGEBRA.

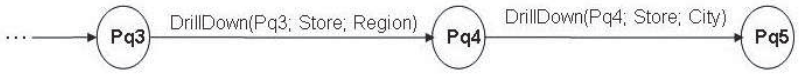

Fig. 3. Example of graph session.

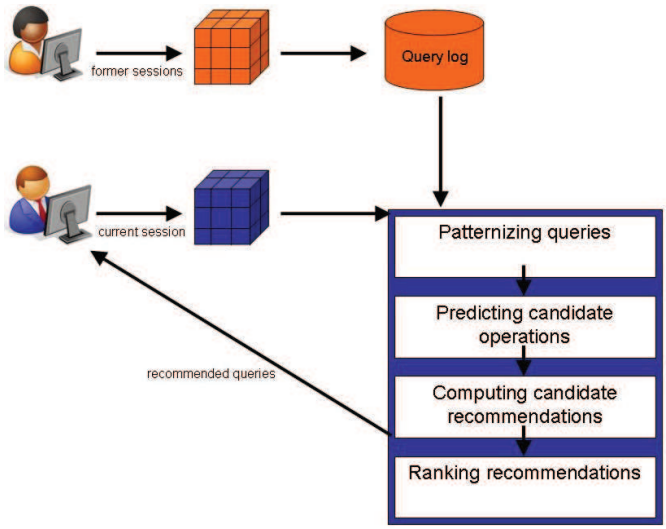

Fig. 4. Overview of the cold-start process.

Algorithm 2 extractQueryPattern(q, C)

Require:
q: A given query,
C: The cube on which the query q has been launched.
Ensure: a list of cube level names (QP)

$QP \leftarrow \emptyset$
for each dimension D_i of C **do**
$QP \leftarrow QP \cup getLevelName(getReferences(q, C))$
end for
return QP

The operation extraction (algorithm 3) consists in extracting the references (a set of attribute values) of two given queries, the corresponding level numbers on each dimension and finding the operation that allows navigating from one query to the other. We focus on common OLAP operations: slice-and-dice, drill-down, roll-up, switch and pivot. Note that slice-and-dice and switch operations do not influence our approach as these operations affect only values. Our approach operates at a higher level; it manipulates the cube structure and not its values.

More precisely, for two given queries (q_1, q_2) and for a each dimension of the cube C :

- if the references (sets of attributes) of each query are located on the same hierarchical level of the dimension, the system considers that the operation to pass from a query to the other one is $identity$,
- if the references of only one query are located at the highest level of the hierarchy ($'ALL'$ level) then the operation is a $pivot$,
- if the references of the first query launched are located on a higher level than the references of the second query launched then the operation is a $DrillDown$,
- in all other cases, the operation is a $RollUp$.

When the system detects a $pivot$, it is about determine on which dimension the $pivot$ takes place.

Algorithm 3 extractOp(q_1, q_2, C)

Require:
q_1, q_2: Two given queries,
C: The cube on which the queries q_1 and q_2 have been launched.
Ensure: a list of OLAP operations (OP)

$OP, Pivot \leftarrow \emptyset$
$Niv_1, Niv_2, height \leftarrow 0$
for each dimension D_i of C **do**
  $Niv_1 \leftarrow getLevelNum(getReferences(q_1, C))$
  $Niv_2 \leftarrow getLevelNum(getReferences(q_2, C))$
  $height \leftarrow$ height of the hierarchy of dimension D_i
  if $Niv_1 - Niv_2 == 0$ **then**
    $OP \leftarrow OP \cup$ "$Identity$"
  end if
  if $(Niv_1 \neq height \wedge Niv_2 == height) \vee (Niv_1 == height \wedge Niv_2 \neq height)$ **then**
    $Pivot \leftarrow Pivot \cup D_i$
  end if
  if $Niv_1 - Niv_2 > 0$ **then**
    $OP \leftarrow OP \cup$ "$Drilldown$"
  end if
  if $Niv_1 - Niv_2 < 0$ **then**
    $OP \leftarrow OP \cup$ "$RollUp$"
  end if
end for
if $|Pivot| > 1$ **then**
  for each tuple of dimension $\langle D_i, D_{i+1} \rangle$ of $Pivot$ **do**
    $OP[D_i] \leftarrow$ "$Pivot($" $+ D_i +$ "$,$" $+ D_{i+1} +$ "$)$"
    $OP[D_{i+1}] \leftarrow$ "$Pivot($" $+ D_i +$ "$,$" $+ D_{i+1} +$ "$)$"
  end for
end if
return OP

The function $getReferences(q, C)$ outputs for each dimension of C the attribute values of the query q. For example[1], the references of query q_1 of the motivating example are $\langle Quantity, All^{Vehicle}, US, 2011 \rangle$. The function $getLevelName$ (resp. $getLevelNum$) extracts, for each set of attribute values on each dimension of the cube C, the name (resp. the rank in the hierarchy - 0 for the finest granularity) of the corresponding level in the hierarchy of the dimension. For q_1, we obtain $Year$ as level name for the attribute value 2011 and 2 which is the level number of $Year$ in the TIME hierarchy of C (for $Month$ it would have been 1 or 0 if it was $Date$).

For example, the query q_3, having for references $\langle Quantity, All^{Vehicle}, France, 2007 \rangle$ (see Table I), of the motivating example can be patternized as: $\langle Quantity, All^{Vehicle}, Country, Year \rangle$. Thus, the session of the motivating example containing q_3, q_4 and q_5, can be patternized as illustrated on Figure IV-A.

[1]Note that for readability, query references, query patterns and operations are displayed, in this section, as an ordered 4-uples where each tuple corresponds, in the right order, to measures, then the Vehicle dimension, then the Store dimension and then the Time dimension. For instance, the query reference $\langle m, a, b, c \rangle$ indicates that this query deals with the measure m, with attribute value a on Vehicle dimension, b on Store dimension and c on Time dimension. The query pattern $\langle m, pa, pb, pc \rangle$ indicates that this query pattern deals with the measure m, with levels pa on Vehicle dimension, pb on Store dimension and pc on Time dimension. And the operation $\langle o_1, o_2, o_3, o_4 \rangle$ indicates that there are operations o_1 on measures, o_2 on Vehicle dimension, o_3 on Store dimension and o_4 on Time dimension.

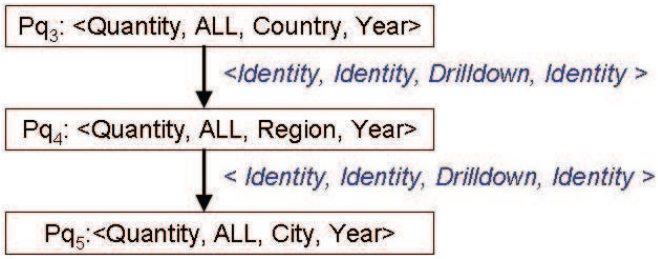

Fig. 5. Pattern of the session of the motivating example.

B. Predicting candidate operations

The previous step produced the set of patternized sessions. Now, using this set and the sequence of queries of the current session, a set of candidate operations is built by algorithm 4: In addition to the set of patternized sessions and the current session, this algorithm uses a matching function: Match. This matching is used to find a set of patternized sessions of the log that match the current patternized session. The algorithm proceeds as follows: First, the patternized current session is obtained (with the PatternizingQueryLog function as the current session can be seen as a log containing only one session). Then, Match is used to search, among the set of patternized sessions, which session matches the patternized current session. This function outputs a set of session pairs indicating which patternized sessions match the patternized current session along with the position of the matching. From these pairs, the algorithm extracts a set of candidate operations which are the operations that allow passing from the query pattern at the returned position in the matching session pattern to the next one in the candidate session pattern. This set of operations is returned as an answer. Note that our goal is that this set cannot be empty and that such an algorithm has a complexity of $O(n[(w+1)^2+1]+w)$ where n is the number of sessions into the log and w is the max length (number of queries) of a given session.

Algorithm 4 PredictingCandidateOperations(s_c, C, SP, Match)

Require:
s_c: The current session,
C: The cube on which the session s_c has been launched,
SP: The set of patternized sessions, as output by the previous step,
Match: A function returning candidate session patterns.
Ensure: a set of candidate operations ($Cand$)

$M, Cand \leftarrow \emptyset$
$P_{s_c} \leftarrow$ the pattern of the current session s_c
$M \leftarrow Match(P_{s_c}, SP)$
if $M \neq \emptyset$ **then**
  for each $m = \langle p, pos \rangle \in M$ **do**
    //where $p = \langle e, v \rangle$ is a session pattern
    $Cand \leftarrow Cand \cup p.v(p.e[pos], p.e[pos+1])$
  end for
end if
return $Cand$

For example, the pattern of the session s_1 in the motivating example that contains q_3, q_4 and q_5 is illustrated in Figure IV-A. The pattern of the current session s_c contains the pattern of q_1: $\langle Quantity, All^{Vehicle}, Country, Year\rangle$, the pattern of q_2: $\langle Quantity, All^{Vehicle}, State, Year\rangle$ and the operations to transform q_1 into q_2: $\langle Identity, Identity, Drilldown, Identity\rangle$. For example, suppose that the Match function returns the pattern of s_1: P_{s_1} matching with the pattern of s_c: P_{s_c} on position 2. The algorithm 4 then returns $\langle Identity, Identity, Drilldown, Identity\rangle$ as candidate operation. Indeed, this drill-down operation changes the query q_4 at position 2 in session s_1 into the query q_5 at position 3 in s_1.

Match function

This Match function has been studied in [18] and [10] where two sequence matching functions are detailed (each being based on an edit distance and approximate string matching) and combined with two member similarity measures (the shortest path and the Hamming distance). The approximate String matching technique used consists in removing, at each iteration, the last element of the sequence and in comparing it with the current sequence by using an edit distance. Note that the number of calculation of the edit distance can be exponential. While if only the edit distance is processed (i.e. Levenshtein distance), the number of calculations is smaller, as it is processed only once for each sequence. Thus, the Levenshtein distance was used for comparing two sessions. This distance is combined with the hamming distance (EdH) or with the shortest path ($EdSP$) for comparing members. [18]'s experiments show that EdH and $EdSP$ perform similarly in terms of recall/precision but [10]'s experiments show that $EdSP$ is slower than EdH. As we are in a cold-Start recommender system problem context, the system has to be fast and efficient (as much as possible). This is why EdH is used: the Levenshtein distance combined with the Hamming distance.

C. Computing candidate recommendations

The previous steps produce a set of candidate operations. Following this, for every candidate operation a new query is computed by applying these candidate operations to the last query of the current session. Recently, a multidimensional algebra [21] could guarantee that the applied operations will have a sense. In [10], five possibilities were considered: (i) the last query of the current session, (ii) the successor of a given query of the current session, (iii) the union of the current session queries, (iv) the intersection of the current session queries, and (v) the medoid [2] of the current session queries. Possibility (v) is time consuming when sessions are long (due to the calculation of the medoid for a big number of queries). Possibility (iv) can result in an empty query and possibility (iii) can create a query whose result would correspond to the entire data cube. By analogy with the web [29] and in our cold-start context, we suppose that intermediary steps are useless and that an analytical session focuses on the goal of the session. Thus, for us, the better possibility is to apply the operations to the last query of the current query to compute the candidate recommendations.

[2]The medoid of a set of queries belongs to this set and is the query that minimizes the distances with the other queries of the set.

In our example, there is a candidate operation: $\langle Identity, Identity, Drilldown, Identity\rangle$. The computation of a new query by applying this operation (a drill-down on the STORE dimension) to the last query q_2 (quantities of vehicles sold in Texas in 2011) of the current session s_c returns the query identified in Table I by q_{pred} (quantities of vehicles sold in Texas cities in 2011).

D. Ranking the candidate recommendations

From the previously computed set of recommendations, the idea is to select the most suitable recommendation. We consider that we cannot have a satisfaction criterion expressed by the user. To this end, a query ranking, that orders the candidate recommendations, is difficult to propose.

For now, the solutions that could be considered are:

- Ranking the candidates according to how close to the last query of the current session they are.
- Ranking the candidates according to their number of occurrences in the logs.
- Ranking the candidates according to the position of the candidate operation in the patternized log, i.e. the most recently launched operations in the log will be used to compute the first queries in the set of recommended queries.

First, we know neither the density of the log nor its contents. Thus, ranking the candidates according to their number of occurrences in the logs can be difficult, if, for example, each candidate appears the same number of times. Still for timing reasons, ranking the candidates according to how close to the last query of the current session they are can be time consuming. For these reasons, the candidates are ranked according to the position of the candidate operation in the patternized log.

In the end, our proposition of a cold-start recommender system, that will be implemented in our future work, consists in :

1) Patternizing queries of the former log by using our $PatternizingQueryLog$ algorithm.
2) Predicting candidate operations by using the patternized current session and the set of patternized log sessions and the EdH match function between the pattern of the current session and the set of patternized sessions, in our $PredictingCandidateOperations$ algorithm
3) Computing the candidate recommendations by combining the last query of the current session and the candidate operations.
4) Ranking the candidates according to the position of the candidate operation in the patternized log.

V. Conclusion and discussion

In this paper, we exposed the limits of existing recommender systems when exploring data cubes on new systems. In order to overcome these limitations we propose to develop a process for predicting queries that will be based on user behavior analysis during their sessions (these sessions corresponding to sequences of queries) and on the behaviors during former analysis sessions on another system.

This predictive approach raises 4 challenges: (1) extract the behavior of users by patternizing their current queries and from query logs of former systems; (2) predicting candidate OLAP manipulation operations by matching the current user query pattern with a set of patternized sessions (sets of query sequences); (3) computing candidate recommendations, i.e. selecting the relevant operations between the predicted ones; and finally, as there may be several recommended operations, (4) ranking these candidate recommendations. This cold-start recommender system is used until the system has gathered enough user manipulation data in order to run a more appropriate standard recommender system. This allows the user to have recommendations although the system is bran new.

As this is preliminary work we intend, as future work, to perform experiments and scale them up. This will allow us to test thoroughly our algorithms and to answer questions such as: how to match the best patternized query (if such query exists) and if numerous patternized queries are returned how to order them to avoid confusing the user with too many suggestions. Moreover, we intend to take into account more complex OLAP operators such as NEST and Drill-Across. Finally, we also wish to apply our algorithms on more complex multidimensional schemas, that is, new schemas that would have more differences with the schemas of the former systems (i.e. composed of different dimensions or facts).

References

[1] G. Colliat, "Olap, relational, and multidimensional database systems," *SIGMOD Record*, vol. 25, no. 3, pp. 64–69, 1996.

[2] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[3] P. Lyman, H. R. Varian, P. Charles, N. Good, L. L. Jordan, and J. Pal., "How much information ?" Available at http:www2.sims.berkeley.eduresearchprojectshow-much-info-2003, 2003.

[4] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[5] R. A. Baeza-Yates, "Query intent prediction and recommendation," in *RecSys*, 2010, pp. 5–6.

[6] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis, "Query recommendations for interactive database exploration," in *SSDBM*, 2009, pp. 3–18.

[7] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu, "A case for a collaborative query management system," in *CIDR*, 2009.

[8] K. Stefanidis, M. Drosou, and E. Pitoura, ""you may also like" results in relational databases," in *Proc. of the 3nd International Workshop on Personalized Access, Profile Management and Context Awareness: Databases (PersDB 2009), in conjunction with the VLDB 2009 Conference*, 2009.

[9] P. Marcel and E. Negre, "A survey of query recommendation techniques for datawarehouse exploration," in *EDA*, 2011.

[10] E. Negre, "Exploration collaborative de cubes de donnes," Ph.D. dissertation, Université Franois-Rabelais de Tours, 2009.

[11] H. Jerbi, F. Ravat, O. Teste, and G. Zurfluh, "Preference-based recommendations for olap analysis," in *DaWaK*, 2009, pp. 467–478.

[12] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent, "A personalization framework for olap queries," in *DOLAP*, 2005, pp. 9–18.

[13] M. Golfarelli, S. Rizzi, and P. Biondi, "myolap: An approach to express and evaluate olap preferences," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 7, pp. 1050–1064, 2011.

[14] S. Sarawagi, "User-adaptive exploration of multidimensional data," in *VLDB*, 2000, pp. 307–316.

[15] V. Cariou, J. Cubillé, C. Derquenne, S. Goutier, F. Guisnel, and H. Klajnmic, "Built-in indicators to discover interesting drill paths in a cube," in *DaWaK*, 2008, pp. 33–44.

[16] C. Sapia, "On modeling and predicting query behavior in olap systems," in *DMDW*, 1999, p. 2.

[17] X. Yang, C. M. Procopiuc, and D. Srivastava, "Recommending join queries via query log analysis," in *ICDE*, 2009, pp. 964–975.

[18] A. Giacometti, P. Marcel, and E. Negre, "A framework for recommending olap queries," in *DOLAP*, 2008, pp. 73–80.

[19] ——, "Recommending multidimensional queries," in *DaWaK*, 2009, pp. 453–466.

[20] A. Giacometti, P. Marcel, E. Negre, and A. Soulet, "Query recommendations for olap discovery-driven analysis," *IJDWM*, vol. 7, no. 2, pp. 1–25, 2011.

[21] O. Romero, P. Marcel, A. Abelló, V. Peralta, and L. Bellatreche, "Describing analytical sessions using a multidimensional algebra," in *Proceedings of the 13th international conference on Data warehousing and knowledge discovery*, ser. DaWaK'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 224–239. [Online]. Available: http://dl.acm.org/citation.cfm?id=2033616.2033639

[22] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Generative models for cold-start recommendations," in *IN PROCEEDINGS OF THE 2001 SIGIR WORKSHOP ON RECOMMENDER SYSTEMS*, 2001.

[23] M. A. Kłopotek, "Approaches to cold start in recommender systems," in *Proceedings of 10th International Conference on Artificial Intelligence*, ser. Proceedings of Artificial Intelligence Studies, M. A. Kłopotek and J. Tchórzewski, Eds., vol. 5, no. 28, Siedlce, Poland, Sep. 18-19, 2008, pp. 29–33.

[24] N. Golbandi, Y. Koren, and R. Lempel, "Adaptive bootstrapping of recommender systems using decision trees," in *WSDM*, 2011, pp. 595–604.

[25] A. M. Rashid, G. Karypis, and J. Riedl, "Learning preferences of new users in recommender systems: an information theoretic approach," *SIGKDD Explorations*, vol. 10, no. 2, pp. 90–100, 2008.

[26] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, "Methods and metrics for cold-start recommendations," in *SIGIR*, 2002, pp. 253–260.

[27] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh, "Graphical querying of multidimensional databases," in *ADBIS*, 2007, pp. 298–313.

[28] T. Sboui, M. Salehi, and Y. Bédard, "A systematic approach for managing the risk related to semantic interoperability between geospatial datacubes," *IJAEIS*, vol. 1, no. 2, pp. 20–41, 2010.

[29] R. W. White, "Studying the use of popular destinations to enhance web search interaction," in *ACM SIGIR '07. ACM*. Press, 2007, pp. 159–166.