Query Rewriting Based on User's Profile Knowledge

Dimitre Kostadinov Mokrane Bouzeghoub Stéphane Lopes

Laboratoire PRiSM - Université de Versailles
45, avenue des Etats-Unis
78035, Versailles cedex, France
{firstname.lastname}@prism.uvsq.fr

ABSTRACT

Query personalization was introduced as an advanced mechanism to allow the reformulation of database queries and adapt them to the user's domain of interest and preferences. Domain of interest and preferences are captured into a profile which is explicitly defined by the user or derived from his or her logged interactions with the information system. A substantial effort was done in recent years to provide reformulation techniques which enrich user queries prior to their execution. These approaches are often limited to a single data source. However, query personalization is much more crucial in a context where data sources are numerous and distributed, i.e. a mediation or P2P systems, as the user is generally overloaded with massive results in response to his or her queries. To deal with such a context of distributed data sources, we have studied two naive approaches and proposed an advanced approach, based on a schema-free profile. A comparison of the three approaches is performed, through two metrics we have defined. The limitations of the two first approaches are shown and the improvements provided by the third approach are demonstrated through the experiment.

Key words: data access personalization, user profile, query reformulation, query rewriting

1. INTRODUCTION

The access to relevant information, adapted to the preferences and the context of the user is a challenge in modern information systems, in particular when data is massively distributed and potentially redundant. Generally, querying databases with conventional DBMSs leads to an informational overload which makes the user unable to distinguish relevant information from secondary one or even from the noise. Besides, another drawback of these systems is their incapacity to discriminate between users according to their domains of interest, their preferences or their querying contexts, and to deliver them relevant results according to their respective profiles. These limits can easily be observed in distributed databases, P2P and mediation systems which, for a given query, deliver the same massive and redundant results, regardless of the user who issued this query and of the context in which it was issued.

Access models of conventional database systems are conceived for a standard use, with a closed world assumption, where the user knows not only the database schema but also the exact information he wants: his queries are exact expressions of his needs. Distributed and data integration systems generalized this approach to multiple data sources, without considering their specificity such as evolution of data sources, their temporary or permanent unavailability, the relative quality of their redundant data, etc. Whatever the system architecture is, queries are treated similarly for all users and the returned results are considered fully precise (i.e. they exactly match the query conditions) and complete (i.e. considering closed world assumption).

However, these access models do not satisfy a wide spectrum of users who have new requirements such as taking into account their geographical location, the media used for the expression of their queries, their preferences in terms of data quality, data presentation, data security and so on. Additionally, with the multiplicity of connected databases, in particular in large scaled P2P architectures, users do not generally specify an exact need but an intention which must be refined and interpreted according to the targeted data sources. Then, new access practices have emerged, pushing database systems to behave closer to information retrieval systems.

In this perspective, several new access models have been proposed: approximate queries [6], ranked que-

ries [19], preference queries [17], context aware queries [11] and so on; all of them aiming, at some extent, to provide the user with more and more relevant data. Most of these works are based on query languages extension, as in PreferenceSQL [13] or SQL/f [2]. These models are interesting steps toward an adaptive access model; however they always force users to completely specify their queries, ignoring invariant elements which constitute their own specificity. Clearly, there is no distinction between a user profile and a user query.

The concept of user profile has been introduced in many works in the database field, with various goals such as pre-selection of data sources [20], delivery of data with more or less freshness [4], or reformulation of queries using user profiles prior to their execution [16]. The first approach addressed data source selection according to their quality measures which should match as much as possible to the requirements given in the user profile. The second approach explored the capacity of a distributed system, based on the content of its memory cache, to deliver data using a user preference between query response time and data freshness. The third approach exploited preferences given in the user profile, as weighted predicates, to enrich user queries during their compilation. Applied during design time [20], compile time [16][25] or execution time [4], these approaches constitute a step forward in a profile-based access to databases, and as such constitute a good background to a general personalized access model.

Data personalization has also been addressed in other domains such as information retrieval and human-computer interaction. In information retrieval systems [22], the user is fully involved in the query evaluation which is conducted as a stepwise refinement process where the user can decide at each step which data he likes and which data he dislikes. The personalization is then considered as a machine learning process based on user feedback. In human-computer interaction [8], user profiles generally define user expertise with respect to the application domain in order to provide them with appropriate interfaces and dialogs. Data delivery modalities, graphical metaphors and level of expertise of the dialog constitute the main personalization issues in this context.

The work presented in this paper follows the line of this state of the art, in particular the query reformulation approach using user profiles. But in contrast to previous work, our approach applies to the context of data integration systems, e.g. a mediation system. Indeed, we consider the problem of query personalization as more crucial in large scaled distributed systems, where users have generally fewer knowledge on data source descriptions, than in a single database system. The multiplicity of data sources, possibly redundant or replicated, their relative quality with respect to user needs and the information overload returned by corresponding access models, constitute strong obstacles to the delivery of relevant data adapted to the preferences of each user. In this context, the primary goal of a personalized access model (PAM) is to identify relevant data sources and to reformulate user queries over these data sources. Query reformulation consists in reinterpreting the user intention, expressed in his initial query, in a more complete query, considering at the same time the user profile and the descriptions of the data sources. Query reformulation aggregates two complementary tasks: query rewriting which determines relevant data sources and query enrichment which integrates the user profile.

Our general requirements for a PAM to a set of integrated data sources are the followings:

- We make a clear distinction between user profiles and user queries: a profile is a user model which specifies the user domain of interest and the most general preferences which distinguish this user from the others, while a query is an on-demand user need which is evaluated with respect to a certain profile. All queries issued by the same user are evaluated with respect to his specific profile. The same query issued by different users may return different results as it is evaluated using different profiles.
- A profile is defined by a set of attributes, possibly organized into multidimensional entities [3], whose values can be user-defined or dynamically derived from user behavior. The user profile is schema-free, it does not depend on any database schema as it is supposed to characterize the user domain of interest and preferences.
- The goal of a PAM is to obtain as much as relevant results for the user. In the context of database systems, this turns into augmenting user queries with more restrictive predicates. As a consequence, personalized queries obtain, generally, a

restricted number of results compared to non personalized queries.

We limit our scope to relational data sources and consider their semantic links with the global schema through the Local As View (LAV) approach which is best adapted to a rapid evolution of data sources. In this context, we assume having the necessary metadata defining the global schema as well as the data sources.

On the basis of two known algorithms, carrying out respectively query rewriting [9] and query enrichment [15], we propose three query reformulation techniques. Two of them are simple compositions of the previously mentioned algorithms, while the third one is a new query rewriting algorithm driven by the user profile.

The three approaches are compared using two metrics, *coverage* and *precision*, adapted from measurements commonly used in information retrieval systems. Our experiment shows that the third approach is well suited to both the selection of the best data sources and to the use of the best part of the user profile. Thus, this approach constitutes a good basis for a new PAM to any distributed set of data sources whose access is based on query rewriting.

Section 2 presents some background definitions on an example which will be used along the paper as a basis to illustrate our approach. Principles of query rewriting and query enrichment are also explained in this section. Section 3 describes intuitively two naive reformulation approaches combining these techniques, as well as our advanced approach. Two evaluation metrics are introduced in section 4. Section 5 details our profile-based query rewriting approach and presents the experiments. Section 6 concludes the article with further research.

2. BACKGROUND AND EXAMPLE

As mentioned before, query rewriting and query enrichment are fundamental techniques to respectively select data sources and improve user queries prior to their execution. This section will recall the principles of two algorithms we have chosen as a basement for our work, namely the rewriting algorithm of [9] and the enrichment algorithm of [15]. Before, we give a simple example of a data integration system which will serve to illustrate motivations and principles of these algorithms as well as our approach.

2.1 Motivating Example

Our data integration system deals with travels, transports and hotels that a user can book for his trips. The virtual schema of this system is composed of the relations presented in Example 1. Primary keys are underlined and foreign keys are in italic.

Example 1: Virtual schema (S_v)

```
TRAVEL(vid, price, departure, arrival, nbDays, departDate,
       departTime, visitType, tripType, tid, hid)
TRANSPORT(tid, mean, wayType, comfort)
HOTEL(hid, nbStars, name, region, city, restaurant)
```

The instances of this schema are computed using six data sources: WORLDHOTELS (information about hotels), PLANETRANSPORT (flights information), SNCF (travels by French trains), RIDEEVERYWHERE (travel with different transport means), PROMOHOLYDAYS (promotional travels from Paris) and LYONHOLYDAYS (travels from Lyon). Each data source is described by a LAV mediation query expressed in a Datalog-like language (Example 2).

The left side of each definition, called *head*, corresponds to the source schema and the right side contains one *atom* for each mediation relation which is used for the source definition, plus possibly selection predicates over these relations. To clarify the presentation, the atoms in the right side of a user query will be called *subgoals*. The variables of the head are called *distinguished* variables and all other variables are *existential* variables. We assume that selection predicates are of the form $<var\ \theta\ val>$ where *var* is a variable which appears in some atom, θ is an arithmetic operator and *val* is a constant from the domain of *var*.

Example 2 Data source definitions

```
S1:WORLDHOTELS(hid,nbStars,name,region,city,restaurant)        :-
    HOTEL(hid, nbStars, name, region, city, restaurant).
S2:PLANETRANSPORT(tid,departure,arrival,departDate,departTime,
          mean, wayType, comfort)                              :-
    TRANSPORT(tid, mean, wayType, comfort),
    TRAVEL(vid, price,departure, arrival, nbDays, departDate,
          departTime, visitType, tripType, tid, hid),
    mean = 'plane'.
S3:SNCF(tid,departure,arrival, departDate, departTime,
          mean, wayType, comfort)                              :-
    TRANSPORT(tid, mean, wayType, comfort),
    TRAVEL(vid, price, departure, arrival, nbDays, departDate,
          departTime, visitType, tripType, tid, hid),
    mean='train'.
S4:RIDEEVERYWHERE(tid, departure, arrival, departDate,
          departTime, mean, wayType, comfort)                  :-
    TRANSPORT(tid, mean, wayType, comfort),
    TRAVEL(vid, price, departure, arrival, nbDays, departDate,
          departTime, visitType, tripType, tid, hid).
```

```
S5:PROMOHOLYDAYS(vid, price, departure, arrival, nbDays, depart
                 Date, departTime, visitType, tripType,
                 mean, name, nbStars, restaurant, tid)     :-
    TRANSPORT(tid, mean, wayType, comfort),
    TRAVEL(vid, price, departure, arrival, nbDays, departDate,
           departTime, visitType, tripType, tid, hid),
    HOTEL(hid, nbStars, name, region, arrival, restaurant),
    price<950, departure='Paris'.
S6:LYONHOLYDAYS(vid,price,departure,arrival,nbDays,departDate,
                 departTime, visitType, tripType, mean,name,
                 nbStars, restaurant, tid, hid)          :-
    TRANSPORT(tid, mean, wayType, comfort),
    TRAVEL(vid, price, departure, arrival, nbDays, departDate,
                 departTime, visitType, tripType, tid, hid),
    HOTEL(hid, nbStars, name, region, arrival, restaurant),
    price<950, departure='Lyon'.
```

As an example of user preferences, consider a traveller who likes holydays of *more than 7 days* and usually takes *comfortable direct flights* from *Toulouse* or in the worst case from *Paris* and prefers *circuit* trips with *at least 3 stars* hotels situated in the *city centre*. We will see later in the following sections how this profile can be represented and used depending on the reformulation approaches. Finally, as an example of a user query, we consider the one which searches for 4 days trips in Madrid (Example 3).

Example 3: Initial query (Q_u)

```
SELECT V.vid, V.price, V.departure, T.mean, T.comfort
FROM TRAVEL V, TRANSPORT T
WHERE V.tid=T.tid AND V.arrival='Madrid'
                  AND V.nbDays=4;
```

This example will be used along the paper to illustrate base algorithms as well as our reformulation approaches.

2.2 Query Rewriting

The query rewriting process consists in transforming the user query expressed on the virtual schema so that it can be evaluated on the data sources. It aims to determine contributive data sources for query execution and to use their definitions to reformulate the query [5]. This section presents the query rewriting process in a LAV context. There are two main classes of rewriting algorithms: inverse-rules-based algorithms [7] and bucket-based algorithms [18].

The inverse-rules algorithm [7] [21] constructs a set of rules which invert the source definitions. For each source definition, a rule is created for each of its atoms. This rule specifies how to compute tuples for the virtual relation, represented by the atom, from tuples of the source. The rewritings of a query are then constructed by expanding the query with all possible combinations of inverse rules.

According to [9], the rewritings produced by the inverse-rules algorithm described in [7], are not appropriate for query evaluation for two reasons. First, applying the inverse rules to the extension of a source definition may invert some of the useful operations done to produce the source data. Second, irrelevant sources can be accessed in order to compute the query results.

The bucket-based algorithm proceeds in two steps [18]: (i) create a bucket for each subgoal of the query which contains source definitions which are relevant to answering the particular subgoal and (ii) construct the candidate rewritings by keeping only rewritings contained in the query. A query Q is *contained* in another query Q' if for any database D, the set of tuples returned by the evaluation of Q over D is a subset of the tuples returned by the execution of Q' over D. Each candidate rewriting is a conjunctive query obtained by taking one element from each bucket.

The main problem with this algorithm is that it considers each atom of the source definition in isolation when it looks for relevant sources. Doing so leads to the missing of some important interactions between the atoms in the source definitions [9]. In particular, it misses the fact that if a source will be used to rewrite some query subgoal, then there might be other query subgoals which have to be covered by the same source. This fact is due to join variables in the query, which cannot be covered using other source definitions. These problems are solved by the MiniCon algorithm [9] which takes into account the interactions between the variables in the subgoals of the query in order to find relevant data sources.

The first step of the MiniCon algorithm begins by searching for correspondences between the subgoals of the query and the atoms of the source definitions. Once it finds a correspondence between a subgoal *g* of the query Q and an atom *g'* of a source S_i, the algorithm considers the join predicates of Q in order to find the minimal subset of subgoals in Q which have to be mapped to atoms in S_i given that *g* will be mapped to *g'*. In other words, the algorithm checks that each join predicate in the mapped subgoals of the query is either satisfied by the source definition or is expressed on distinguished variables in the source definitions. In the latter case, the join can be processed with other sources. We say that an atom g' *covers* a query subgoal g if g is mapped to g'. The map-

ping information between a query and a source definition is called a MiniCon Descriptor (MCD). A MCD contains information about mappings, denoted by φ, between variables in Q and variables in S_i as well as the subgoals in Q which are covered by atoms in S_i. When computing the MCDs, the algorithm verifies that (i) distinguished variables in the query are mapped into distinguished variables in the sources, (ii) for each mapping, there are not conflicting predicates between the query and the sources and (iii) each query predicate involving the mapping *can be rewritten*. Two predicates are *conflicting* if they cannot be satisfied simultaneously (for example « price<10 » and « price>15 »). A query predicate can be *rewritten* with respect to a given source (MCD) if it is either contained in this source definition or if there exists a distinguished variable on which this predicate can be expressed.

The second step of the MiniCon algorithm consists in combining the MCDs in order to compute the candidate rewritings. The MCDs generated by the MiniCon algorithm can be combined without checking whether the join predicates in the query can be satisfied or not by the sources. In order to form a candidate rewriting, a set of MCDs must satisfy the following conditions: (i) they are not redundant, (ii) they do not contain conflicting predicates expressed on variables mapped to the same query variable and (iii) they cover all query subgoals.

The MCDs which can be generated for the query Q_u from Example 3 and the source definitions in Example 2 are presented in Table 1.

The source WORLDHOTELS does not contain any atom which can be mapped to a subgoal in Q_u, so no MCD is created for this source. For each source containing information about transport (PLANETRANSPORT, SNCF and RIDEEVERYWHERE), one MCD can be created which covers the query subgoal TRANSPORT. Finally the last two sources (PROMOHOLYDAYS and LYONHOLYDAYS) can cover the first query subgoal (TRAVEL). Consequently, the rewriting expression of Q_u is a union of six candidate rewritings resulting from the combination of the generated MCDs. Example 4 presents the candidate rewriting obtained from the MCDs of the source SNCF and PROMOHOLYDAYS.

Table 1. MCDs for Q_u

SOURCE	MAPPINGS	SUB-GOALS
PLANE TRANSPORT	tid→tid, mean→mean, wayType → wayType,comfort→comfort	2
SNCF	tid→tid, mean→mean, wayType → wayType, comfort→comfort	2
RIDEEVERY WHERE	tid→tid, mean→mean, wayType →wayType, comfort→comfort	2
PROMOHOLY DAYS	vid→vid,price→price, departure → departure, arrival →arrival, nbDays→nbDays, departDate → departDate, depart-Time→departTime, visitType → visitType, tripType → tripType, tid→tid, hid→hid	1
LYONHOLY DAYS	vid→vid, price→price, departure → departure, arrival→arrival, nbDays → nbDays, departDate → departDate, departTime→departTime, visitType → visitType, tripType → trip-Type, tid→tid, hid→hid	1

Example 4: A candidate rewriting of Q_u

```
RW(Qu)(vid, price, departure, mean, comfort):-
    SNCF(tid, departure, arrival, departDate,
          departTime, mean, wayType, comfort),
    PROMOHOLYDAYS(vid, price, departure, arrival, nbDays,
                  departDate,  departTime,  visitType,
                  tripType, mean, name, nbStars,
                  restaurant, tid),
    arrival='Madrid',nbDays=4.
```

2.3 Query Enrichment

Query enrichment is a process which exploits the user profile to enhance expressiveness of his query by integrating specific knowledge collected in his profile. This technique, commonly used in information retrieval systems, is very recent in the database domain.

An interesting method defined recently is the one of Koutrika and Ioannidis [15][16]. To some extent, this method appears as very close to the view mechanism. Indeed, the user profile is defined as a list of disjunctive predicates, including selections and joins. Predicates are associated with weights, ranging within [0,1] interval and representing their relative importance with respect to the user preferences. Among this ordered set of predicates, the user can also specify another type of preference: the number of predicates he wants to be taken into account in the enrichment process. Given such a profile, the query enrichment process consists in reformulating the initial user query by adding predicates from this profile. To go

further and to emphasise the specificity of the approach, we illustrate the process by an example.

In [15], the user profile may contain both selection predicates and join predicates. Each predicate is characterized by a weight, ranging from 0 to 1, which defines its relative importance with respect to other predicates of the profile. Predicates are ordered according to their decreasing weights. In the context of the database schema of Example 1, the informal preferences of the traveller of section 2.1 lead to the profile given in Example 5. The query enrichment is processed in two steps: (i) relevant predicates selection and (ii) integration of the relevant predicates to the query.

Example 5: User Profile as defined in [15] (P_k)

```
{  TRAVEL.tid = TRANSPORT.tid        1.0    (a)
   TRAVEL.hid = HOTEL.hid            1.0    (b)
   TRAVEL.nbDays > 7                 1.0    (c)
   TRAVEL.departure = 'Toulouse'     0.8    (d)
   TRANSPORT.mean = 'plane'          0.7    (e)
   TRANSPORT.wayType = 'direct'      0.6    (f)
   HOTEL.nbStars > 3                 0.5    (g)
   TRAVEL.tripType <>'circuit'       0.5    (h)
   TRANSPORT.comfort > 2             0.4    (i)
   TRAVEL.departure = 'Paris'        0.3    (j)
   HOTEL.region = 'centre city'      0.2    (k)  }
```

Relevant predicates are those related to the query and not conflicting with it. A profile predicate is *related* to a query if it is expressed on a relation of the query or if there is a join path between one query relation and the relation on which the predicate is expressed. Only join predicates from the user profile are used to find the join paths. For example the predicate (g) (« HOTEL.nbStars > 3 ») from P_k can be related to Q_u using the join predicate (b) (« TRAVEL.hid = HOTEL.hid »). In this case, if (g) is used to enrich Q_u, the virtual relation HOTEL is also added to Q_u. A profile predicate is conflicting with a given query if its conjunction with this query leads to empty results. For example the predicate « TRAVEL.nbDays > 7 » is conflicting with Q_u because Q_u contains the predicate « nbDays = 4 » and both predicates cannot be satisfied simultaneously.

The second step of the query enrichment process is the integration of relevant predicates to the initial query. This is done using three preference parameters given by the user: (i) the number *K* of high weighted predicates taken from the user profile, (ii) *M* which is the number of predicates among the *K* predicates selected above which are mandatory and (iii) the number of the remaining (K-M) predicates which at least must be satisfied by each result.

The second step of the query enrichment process can be summarized with the following steps: (i) make all mandatory predicates as one conjunctive clause, (ii) make a conjunctive clause with each combination of L predicates among the remaining (K-M), (iii) make a disjunctive clause of all these combinations, and finally (iv) add these new clauses as conjunctive sub queries to the initial user query.

Consider the initial query Q_u in Example 3, the user profile P_k in Example 5 and set parameters K=6, M=3 and L=2. In the first step, the 6 predicates with the highest weights not conflicting with Q_u are selected; this excludes the predicate (c). Among the selected predicates {d, e, f, g, h, i}, the first 3 (d, e and f) are considered mandatory and are added to Q_u as well as the disjunction of the conjunctions of 2 predicates among the remaining ones (g, h and i). The resulting query enrichment Q_{u+} is shown in Example 6. Notice that the predicate (g) is expressed on the virtual relation HOTEL which is not present in the initial query. This relation is added using the join predicate (b) which relates it to the relation TRAVEL of the initial query.

Example 6: Enrichment of the initial query (Q_{u+})

```
SELECT V.vid, V.price, V.departure, T.mean, T.comfort
FROM TRAVEL V, TRANSPORT T, HOTEL H
WHERE V.tid=T.tid AND V.arrival='Madrid'
      AND V.nbDays=4
      AND V.hid=H.hid AND V.departure='Toulouse'
      AND T.mean='plane' AND T.wayType='direct'
      AND ((H.nbStars>3 AND V.tripType<>'circuit')
           OR (H.nbStars>3 AND T.comfort>2)
           OR(V.tripType<>'circuit' AND T.comfort>2))
```

The algorithms of query rewriting and query enrichment can be combined to achieve a personalized access to a multi-source data integration system. To this end, the next section presents three approaches of query reformulation: two naive ones which consist in composing in different orders the rewriting and enrichment algorithms; the third approach which is a new reformulation approach we defined over the rewriting algorithm using the user profile.

3. QUERY REFORMULATION: AN INTUITIVE PRESENTATION

Query enrichment process and query rewriting process have different goals. The first one takes into account the user profile to enrich the initial query and,

thus, increases the relevance of the obtained results. The second one translates the query to access the real data sources. To provide the user with a personalized access to a set of distributed data sources, both approaches are necessary. Let $\mathcal{R}$ and $\mathcal{E}$ be respectively the rewriting and the enrichment process, the problem is to decide in which order the two processes should be applied ($\mathcal{E}(\mathcal{R})$ or $\mathcal{R}(\mathcal{E})$), that is: is it better to first select data sources and then enrich the resulting rewritings, or is it better to first enrich the user query and then rewrite the enriched query? We designate these two approaches as naive ones because it is a simple composition of existing algorithms.

An alternative approach is a hybrid one using the basic ideas of the two previous algorithms but providing a more integrated reformulation principle. This approach, called profile-based query rewriting (or $\mathcal{R}/\mathcal{P}$ for short), constitutes our contribution which will be evaluated with respect to the two naive approaches.

This section gives an intuitive understanding of the three query reformulation approaches.

3.1 Enrichment-Rewriting Approach: $\mathcal{R}(\mathcal{E})$

The first naive reformulation approach consists in enriching the initial query with the user profile before computing the candidate rewritings of the obtained enrichment. The goal of this approach is to increase as much as possible the relevance of the query by applying the enrichment algorithm first and to take into account as much profile predicates not conflicting with the initial query as possible.

Due to the fact that in the $\mathcal{R}(\mathcal{E})$ approach, the enrichment process is done without considering the data source definitions, it may occur that some of the predicates which have been added to the query are useless or impossible to be rewritten. A profile predicate is *useless* if it is already included in the source definitions of all query rewritings.

As mentioned in the previous section, in some cases, only the Top K profile predicates are used in the query enrichment process. Consequently, if there are useless predicates among the Top K ones, it is desirable to replace them so more profile predicates are taken into account without modifying the parameters of the enrichment process.

In the worst case, enriching the initial query can lead to an empty set of rewritings. This occurs if a predicate in the user profile, which has been used to enrich the initial query, cannot be rewritten. Consider the enriched query in Example 6 which contains the predicate « TRAVEL.departure='Toulouse' ». This predicate is conflicting with both sources which can cover the query subgoal TRAVEL (i.e. PROMOHOLYDAY and LYONHOLYDAY); thus resulting in an empty set of rewritings for the enriched query. A possible solution to these problems is to take into account data source definitions before query enrichment process.

3.2 Rewriting-Enrichment Approach: $\mathcal{E}(\mathcal{R})$

The main idea of the rewriting-enrichment approach is to process the rewriting of the user query prior to the enrichment of these rewritings.

The profile predicates which can be used to enrich the query rewritings are selected using the mappings between the query variables and the source variables, computed during the rewriting process. The following interactions may exist between a profile predicate p and a query rewriting RW:

- p is conflicting with RW if p is conflicting with the source definitions of RW,
- p is satisfied by RW if p is included in the source definitions of RW,
- p can be expressed on RW if there is a distinguished source attribute, involved in the mappings φ, on which p can be expressed.

Consider the candidate rewritings of the initial query Q_u of Example 3, the user profile P_k of Example 5 and the preference values K=5, M=2 and L=2 for the enrichment algorithm. The set of profile predicates selected for the enrichment process may differ from one candidate rewriting to another due to the sources involved in their definitions. For example, the profile predicate (e) is conflicting with the definition of the source SNCF (S_3) and cannot be used to enrich the candidate rewritings based on this source while it can enrich the candidate rewritings based on RIDEEVERYWHERE (S_4). Notice that predicate (e) is also satisfied by the candidate rewritings based on PLANETRANSPORT and is not relevant for their enrichment.

For the rest of the profile predicates, (c) is conflicting with all candidate rewritings because it is in conflict with the initial query and consequently is not selected for the enrichment process. In the same way, the

predicate (d) is conflicting with all candidate rewritings and must be ignored. The predicate (g) and (k) are expressed on attributes not involved in the mappings (i.e. cannot be used for the enrichment of the candidate rewritings). In summary, the five predicates {e, f, h, i, j} will be used to enrich the two candidate rewritings based on S_4, and the set of predicates {f, h, i, j} will be used for the candidate rewritings based on S_2 and S_3.

The advantage of the $\mathcal{E}(\mathcal{R})$ approach is its ability to consider in the enrichment phase only predicates of the user profile which are not conflicting and which are not already contained in the source definitions. It has a major disadvantage: the rewriting process has defined once for all the target data sources; consequently, only the profile predicate concerning these sources can be used. An example of profile predicate in P_k which cannot be used to enrich the candidate rewritings of Q_u is the predicate (g).

3.3 Profile-Based Query Rewriting: $\mathcal{R}/\mathcal{P}$

The third query reformulation approach is an improvement of the two previous ones. Instead of separating the two processes, the enrichment process is spread over the different phases of the rewriting process. Starting from the rewriting algorithm [9], the idea is to produce only candidate rewritings which are of some relevance with respect to the user profile, i.e. rewritings which can later integrate some part of the user profile.

The user profile is schema-free in our approach, so it does not contain any join predicate. The user can exploit its profile on any real or virtual database schema, provided it exists at least a partial matching between this schema and the user profile. We say that the user profile has to be first interpreted before it can be used with a particular schema. The profile *interpretation* is the process of matching the attributes of the user profile P_u, denoted by $\mathcal{A}(P_u)$, against those of the schema S, denoted $\mathcal{A}(S)$. The result of this matching is a set of mappings between semantically equivalent attributes. This set of mappings can be seen as a binary relation $\mathcal{M}$ between attributes of the user profile and those of the schema, defined as follows:

$\mathcal{M} \subseteq \mathcal{A}(P_u) \times \mathcal{A}(S)$ and $(a, R.b) \in \mathcal{M}$ if it exists a mapping between the attribute $a \in \mathcal{A}(P_u)$ and the attribute b of the relation R such that $R.b \in \mathcal{A}(S)$. The attribute a is said to be *bound* to the relation R through the interpretation $\mathcal{M}$ and noted $\mathcal{M}\{R.b \rightarrow a\}S_v$. By extension, any predicate $p \in P_u$, involving the attribute a is said to be *bound* to the relation R *through* $\mathcal{M}$. The problem of computing the interpretation $\mathcal{M}$ is out of the range of this paper. Consequently, in this paper, we consider only interpreted profiles.

Example 7 gives an interpretation of the traveller's profile through the virtual schema S_v of Example 1. It is almost similar to the profile definition of Example 5, except the joins which are not represented. Additionally, although not considered in this paper, the mappings composing this interpretation can be characterized by the semantic distance between the pair of attributes (considered equal to 1 to simplify our example).

Example 7: Interpreted user profile (P_1)

$\mathcal{M}$ {
TRAVEL.nbDays→nbDays>7 1.0 (c)
TRAVEL.departure→departure='Toulouse' 0.8 (d)
TRANSPORT.mean→mean='plane' 0.7 (e)
TRANSPORT.wayType→wayType = 'direct' 0.6 (f)
HOTEL.nbStars→nbStars > 3 0.5 (g)
TRAVEL.tripType→tripType <>'circuit' 0.5 (h)
TRANSPORT.comfort→comfort > 2 0.4 (i)
TRAVEL.departure→departure = 'Paris' 0.3 (j)
HOTEL.region→region = 'city centre' 0.2 (k)
}S_v

Given the interpretation of the user profile, the rewriting algorithm is extended first with a pre-processing phase which expands the user query scope over the virtual schema, using the user profile. Second, it introduces pruning rules which eliminate rewritings that are irrelevant with respect to the user profile. Third, the enrichment is done on the basis of selected rewritings (i.e. the selected data sources). The whole query reformulation process we propose is shown in Figure 1. It is composed of four steps: query expansion, relevant sources identification, sources combination and final enrichment.

The first step, *query expansion*, consists in augmenting the scope of the user query over the virtual schema in such a way that it can better integrate the user profile predicates. The virtual relations, to which some profile predicates are bound and which are not in the user query, are added to the FROM clause of the user query. The expansion of the user query with these additional relations prepares the integration of some predicates of the user profile into the user

query. Obviously, adding these new relations to the user query implies adding the joins which relate them to the initial query relations. The search of these joins is not trivial and will be developed later in section 5.

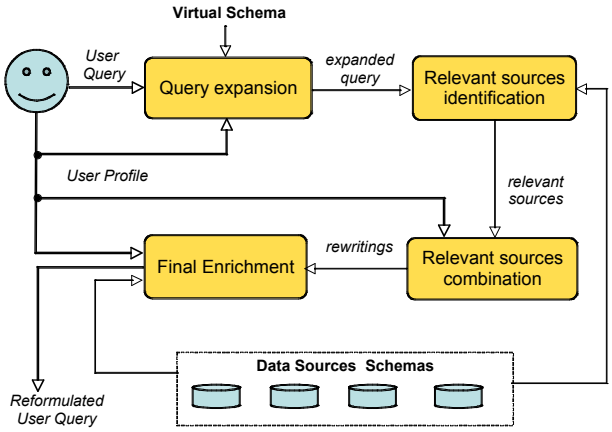

Figure 1. Query reformulation process

The second step, *relevant sources identification*, is the same as the first part of the MiniCon algorithm summarized in section 2.2. It roughly consists in selecting source relations which will be used for the rewriting of the user query. This is done by computing the descriptors (called MCD in MiniCon algorithm) of these selected sources. This step does not directly exploit the user profile but operates on the whole set of relations of the user query, including those resulted from the expansion step.

The third step, *relevant sources combination*, combines the previously selected source relations represented by the MCDs, to generate candidate rewritings which are worth to be efficiently enriched by the last step of the algorithm. Rewritings which are worth to be enriched are those which can integrate a certain number of profile predicates, determined by a metric we will define.

The fourth step elaborates the *final enrichment* of the user query, possibly using the approach in [15], as we have done in our prototype, or another enrichment technique exploiting the outputs of the previous steps.

Compared to the naive query reformulation approaches, the $\mathcal{R}/\mathcal{P}$ approach (as the $\mathcal{R}(\mathcal{E})$ approach) is able to consider during the enrichment step, all profile predicates not conflicting with the initial query. This is due to the query expansion which is assumed to enlarge the query scope with all virtual relations on which at least one profile predicate is expressed. The difference between the $\mathcal{R}(\mathcal{E})$ and the $\mathcal{R}/\mathcal{P}$ approaches is the ability of $\mathcal{R}/\mathcal{P}$ to detect profile predicates which cannot be rewritten. In summary, the hybrid approach has the advantages of both naive query reformulation approaches, but not their limitations: all and only useful profile predicates are taken into account for query enrichment.

The complete algorithm of this approach is detailed in section 5. Steps 1 and 3 of the algorithm use a metric, called *coverage*, which is defined in section 4. In order to evaluate and compare this approach with the naive ones, we introduce another metric, called *precision*, which is also presented in the following section.

4. EVALUATION METRICS

This section introduces two metrics which evaluate, for a given query, the proportion of profile predicates used during the enrichment phase and the real utility of these predicates. The first provides the *coverage* of the approach while the second estimates its *precision*.

These two metrics are usually used to measure the results of a query execution and to compare them to ideal results expected by the user. In our case, we aim to evaluate reformulation approaches during the compile time. Consequently, we are much more concerned by the quality of query definition than by query execution results. Obviously, these qualitative measures are estimations which may or not be confirmed by other measures done on execution results. But this latter goal is outside the scope of this paper. The following paragraphs give a formal definition of these metrics.

4.1 Coverage metric

For a given query reformulation approach, the *coverage* measures the proportion of profile predicates which are *usable* by this approach. A profile predicate is *usable* by a reformulation approach if it can be used in the enrichment phase of this approach.

The weights of the predicates of the user profile are more an estimation of the user preferences, made at the time of the profile definition, than exact values. To limit the possible imperfections of their definition, we propose to partition the profile into a set of groups such as the predicates in a group have approximately the same importance for the user.

Let GR be the partitioning of a user profile P_u into N groups $\{GR_1, \ldots, GR_N\}$, $(GR_1 \cup \ldots \cup GR_N = P_u$ and

$\forall i,j \in 1..N, i \neq j, GR_i \cap GR_j = \varnothing$). The importance I_i of a group GR_i is a function of the weights of its predicates (it is more important to satisfy the predicates of greater weight) and of the number of its predicates (it is more interesting to satisfy 5 predicates out of 10 than 1 out of 2). Let IN_i and IW_i be the relative importance of a predicate group GR_i, computed respectively according to the number and the weights of its predicates. In our evaluation, we chose as function for IN_i the percentage of predicates in each group compared to the total number of predicates in the user profile. The function for IW_i is the normalized average of the weights of the predicates of each group:

$$IN_i = \frac{|GR_i|}{|Pu|} \text{ and } IW_i = \frac{AVG(GR_i)}{\sum_{j=i}^{N} AVG(GR_j)} \text{ with } AVG(GR_i) = \frac{\sum_{p \in GR_i} w(p)}{|GR_i|}$$

where w(p) is the weight of the predicate p. Using these two parameters, we can define the importance I_i of a group as the balanced sum of IN_i and IW_i:

$$I_i = \frac{\alpha \times IN_i + \beta \times IW_i}{\alpha + \beta},$$

where α and β are constants which express the relative importance of the number of predicates and of their weights. For instance, if the weight of the predicates in a group is two times more important than their number, then $\alpha=1$ and $\beta=2$.

Let H_1 and H_2 be two sets of predicates and let $C(H_1, H_2)$ be the *coverage* of the predicates of H_1 with respect to the predicates of H_2: $C(H_1, H_2) = \frac{|H_1 \cap H_2|}{|H_1|}$.

Let *UP(P_u, Q_u, F)* be the set of *usable predicates* obtained from a profile P_u by a reformulation approach *F* for a query Q_u. The predicates in $UP(P_u, Q_u, F)$ are expressed on attributes present in relations of Q_u and are not conflicting with the predicates of Q_u.

Definition 1 (weighted coverage): Let GR be a partition of the user profile P_u.The *weighted coverage*, denoted by $WC(P_u, Q_u, F)$, of P_u by a set $UP(P_u, Q_u, F)$ of usable predicates for a reformulation approach F is equal to the balanced sum of the coverages of the groups of GR by $UP(P_u, Q_u, F)$:

$$WC(P_u, Q_u, F) = \sum_{i=1}^{N} I_i C(GR_i, UP(P_u, Q_u, F))$$

4.2 Precision metric

The *precision* for a given reformulation approach F measures its capacity to take into account only profile predicates which can be rewritten and which influence the final result of the query execution. A predicate can *influence* the result of a query execution either by adding a restriction to some query rewritings, or by eliminating candidate rewritings. The first case occurs if the predicate is not already contained in the source definitions of at least one candidate rewriting. In the latter case, the predicate eliminates at least one contributive source to the rewriting of the query.

A predicate *p*, from a user profile P_u, is considered to be *really useful* for the enrichment process of a query Q_u, if it satisfies the following conditions: (i) p can be used to enrich Q_u and (ii) p can be rewritten and (iii) p influences the query execution result. The set of all really useful predicates of a user profile P_u according to a query Q_u is denoted by *RUP(P_u, Q_u)*. All other predicates of P_u which are not conflicting with Q_u are considered *useless*.

Let *PUP(Pu, Qu, F)* be the set of profile predicates from P_u, *potentially useful* for the enrichment process of a query Q_u by a reformulation approach F. Thus, PUP(Pu, Qu, F) represents the usable predicates $UP(P_u, Q_u, F)$ without the useless predicates that F is able to detect.

Definition 2 (precision): The precision, denoted by $PR(P_u, Q_u, F)$, of P_u, used by a reformulation approach F for the enrichment of a query Q_u is equal to the percentage of really useful predicates among the set of predicates considered to be useful by this approach:

$$PR(P_u, Q_u, F) = \frac{|RUP(P_u, Q_u) \cap PUP(P_u, Q_u, F)|}{|PUP(P_u, Q_u, F)|}$$

The coverage and the precision, previously defined, allows to evaluate the query reformulation approaches and to show in a more objective way their characteristics.

The following section formally defines the advanced query reformulation approach and discusses the results obtained by the evaluation of the query reformulation approaches, using the two evaluation metrics.

5. FORMAL PRESENTATION OF THE PROFILE-BASED QUERY REWRITING $\mathcal{R}/\mathcal{P}$

This section gives a detailed description of the profile-based query rewriting which constitutes our main contribution to query personalization.

5.1 Query Expansion

Query expansion is the process of extending the user query, defined on the virtual schema, using the scope, over the same virtual schema, of the user profile. Extending the query with respect to a given profile scope consists in adding to the query the most relevant relations of the profile scope.

Given a virtual schema S_v and a user profile P_u, the scope represents the set of relations of S_v on which at least one attribute of P_u is bound (see section 3.3 for the definition of a binding). The mappings $\mathcal{M}$ between the user profile P_u and the virtual schema S_v allow defining the *scope* of P_u over S_v as follows:

$scope(P_u, S_v)=\{R \mid R\in S_v \wedge \exists a\in \mathcal{A}(P_u), \exists b\in \mathcal{A}(R), (a,R.b) \in \mathcal{M}\}$.

Let us define the graph $G_{Sv}=(V_{Sv}, E_{Sv})$ where V_{Sv} is the set of relations of the virtual schema S_v and E_{Sv} is the set of joins between these relations. In term of relations, the user query Q_u can be seen as a graph $G_{Qu} = (V_{Qu}, E_{Qu})$ where V_{Qu} is the set of relations of the FROM clause of Q_u, and E_{Qu} is the set of joins used in Q_u. Adding a relation R to Q_u requires finding a join path $J_R \subseteq E_{Sv}$ between a relation of V_{Qu} and R. The graph representing the expanded query is thus $(V_{Qu} \cup \{R\}, E_{Qu} \cup J_R)$.

Let RS be the *relevant scope* of P_u, i.e. the subset of $scope(P_u, S_v)$ containing the most relevant virtual relations. The graph representing the expanded query Q_e is:

$$G_{Qe} = (V_{Qu} \cup RS, E_{Qu} \cup \bigcup_{R\in RS} J_R)$$

Computing this graph consists in two main steps: (i) virtual relation selection and (ii) virtual relation integration.

5.1.1 Virtual relation selection

The first step of query expansion consists in choosing in the profile scope the virtual relations which will be added to the initial query. The selection of these virtual relations is achieved according to two criteria: the length of the join paths and the weights of the bound predicates.

Let p be a profile predicate and $R_p \in scope(P_u, S_v)$ a relation bound to p. Adding p to Q_u leads to adding R_p to Q_u. Let J_{Rp} be a shortest join path in G_{Sv} between R_p and one of the relations in Q_u. If more than one shortest path exist, one of them is randomly chosen. According to [15], the weight of a profile predicate decreases when the length of J_{Rp} increases. We denote by $nw(p, Q_u, S_v)$ the *new weight* of p with respect to a query Q_u and a virtual schema S_v:

$nw(p,Q_u,S_v) = \lambda^{|J_{Rp}|}\, w(p)$, where $\lambda\in[0,1]$ is a constant specifying the decreasing rate and w(p) is the initial weight of the profile predicate.

The new weight of a predicate is decreasing faster when the value of λ is smaller. When λ is equal to 0, only profile predicates expressed on query relations are considered. If λ is equal to 1, the new weight of a predicate remains equal to its initial weight regardless of the length of J_{Rp}.

For example, consider the initial query Q_u in Example 3, the user profile P_1 in Example 7, the virtual schema S_v in Example 1and let λ be equal to 0.9. The joins in S_v are: « TRAVEL.tid = TRANSPORT.tid » and « TRAVEL.hid = HOTEL.hid ». Predicate (g) is bound to the relation HOTEL. The join path J_{HOTEL}={TRAVEL.hid = HOTEL.hid} allows to add HOTEL to Q_u. Its length is 1 and thus $nw((g), Q_u, S_v) = 0.9^1\times0.5 = 0.45$.

Once the weights of the predicates in a user profile P_u are updated, the relevance of each virtual relation $R \in scope(P_u, S_v)$, denoted by $rel(R, P_u)$, is computed. Notice that $rel(R, P_u)=0$ if $R \notin scope(P_u, S_v)$. The relevance of a virtual relation is a scoring function defined on the weights of the profile predicates which are bound to this relation. This function allows sorting the set of virtual relations with respect to their relevance. There are many scoring functions which can be used. We use here the *coverage* metric presented in section 4.1.

Finally the relevant virtual relations RS which will be added to the query are chosen according to a given criterion $\gamma(R)$ which is based on their relevance. This selection criterion can be defined in several ways, for example by specifying a relevance threshold or the number of Top K relations.

The algorithm for virtual relation selection is given in Figure 2.

Algorithm SelectVirtualRelation

```
Input:  Qu: initial query
        Sv: virtual schema
        Pu: user profile
        γ(R): criterion for virtual relation selection
Output: RS: Relevant scope of Pu over Sv
Begin
   Compute the updated profile P'u = {p | p ∈ Pu ∧
                                    w(p) = nw(p, Qu, Sv)}
   For each R ∈ scope(Pu, Sv)
      Compute the relevance rel(R, P'u)
   Return RS = {R | R ∈ scope(Pu, Sv) ∧ γ(R)}
End
```

Figure 2. Virtual relation selection algorithm

5.1.2 Virtual relation integration

Once the relevant virtual relations are selected, it remains to integrate them into the user query. However, each relation added to the query will make the rewriting process more time consuming. Indeed, for each relation we want to add, several other intermediate relations might also be inserted to allow the join between this relation and those of the query. Thus, it is important to minimize the number of added relations which is not an obvious task. Notice that a virtual relation of the profile scope can be added to the initial query even if it has not been selected for the query expansion.

Minimizing the number of new virtual relations is close to the Steiner Tree Problem (STP) [12]. The STP is defined for a non directed graph G = (V, E) where each edge e ∈ E is associated with a positive weight. The STP consists in finding the minimal cost tree which connects a subset W of nodes (W ⊆ V). The STP is known to be NP-complete, so finding the optimal solution is difficult and time consuming.

In our context, the graph G is G_{Sv} and W = $V_{Qu} \cup RS$. All edges have the same weight equal to 1. The main difference between the STP and the virtual relations integration problem is that in the latter one, the semantics of the initial query has to be preserved. Consequently, the join predicates of the initial query form a non reducible subgraph which has to be part of the solution. More details about the discovery of join paths which connect a set of relations can be found in [23].

During query compilation, the cost of computing an optimal solution is prohibitive. Thus, we use the Minimum Cost Paths Heuristic (MPH) [24] to add the relevant profile scope to the query. MPH is an iterative algorithm where each step consists in adding to the partial solution the shortest path to the closest node not yet added. The complexity of the MPH is polynomial in $O(mn^2)$ where n is the number of nodes in G and m is the number of nodes in W.

When using the MPH, it might be possible that for some iteration, there is more than one shortest path with smallest cardinality. In this case the relevancies of the virtual relations of the paths are used to decide which path has to be integrated first. Remark that only virtual relations from the profile scope which are not selected for query expansion have to be taken into account. This is due to the fact that all virtual relations which are not in the profile scope have a relevance of 0. If more than one path has the same relevance, one is randomly selected.

Figure **3** shows the virtual relations integration algorithm. This is an iterative algorithm which takes as input the set of selected virtual relations RS. At each step, the algorithm computes the set of shortest paths having the smallest cardinality between the query relations and the virtual relations in RS. If there are more than one path, the most relevant one, according to the relevance of the virtual relations which compose it, is integrated to the query. The virtual relation for which the path is constructed is removed from RS. The algorithm ends when there are no other selected virtual relations.

Algorithm IntegrateVirtualRelation

```
Input:  Qu: initial query
        Sv: virtual schema
        Pu: user profile with actualized weights
        RS: selected virtual relations
Output: Expanded query
Begin
   While RS ≠ ∅
      Compute the shortest paths J between the query
        relations and the virtual relations RS
      MinSP = { sp | sp ∈ J ∧ |sp| = min_{sp∈J}(|sp|) }
      Select one path sp ∈ MinSP with respect to the
        relevance of the virtual relations
      Expand Qu with sp
      Remove from RS the selected virtual relation
      for which sp is constructed
   endWhile
   Return Qu
End
```

Figure 3. Virtual relations integration algorithm

For the initial query in Example 3 there is only one virtual relation which can be added thus resulting in the expanded query of Example 8.

Example 8: Expanded query

```
Qe =
SELECT vid, price, V.departure, T.mean, T.comfort
FROM TRAVEL V, TRANSPORT T, HOTEL H
WHERE V.tid = T.tid AND V.hid = H.hid
                    AND V.arrival='Madrid'
                    AND V.nbDays=4;
```

In this section, we presented query expansion. Selection of relevant sources is achieved by computing the MCDs of the expanded initial query using the first step of the MiniCon algorithm (see section 2.2) [9]. The next section describes how to combine these MCDs to generate query rewritings.

5.2 Finding Relevant Query rewritings

Adding virtual relations to the user query allows the use of some of the profile predicates bound to these relations. The subset of profile predicates bound to the relations of the expanded query is called *useful profile.*

To build query rewritings, the MCDs have to be combined. This step has to take into account the useful profile to produce only relevant candidate rewritings, i.e. those which can be enriched by a sufficient number of profile predicates.

The main idea of our proposal is to explore the search space in a level wise manner which allows to prune non relevant MCD combinations as soon as possible. To achieve this goal, we adapted the well known *Apriori* algorithm [10]. This algorithm was introduced and successfully used for mining frequent item sets for association rules [1].

The *Apriori* algorithm employs an iterative approach to find the set of "solutions", i.e. subsets which satisfy a property. It is called a level wise algorithm because it explores the search space in a breadth first way and solutions of size i are used to explore candidates of size i+1. Indeed, candidates of size i+1 are obtained by combining solutions of size i. To benefit from the efficiency of *Apriori* algorithm, the desired property has to be *anti-monotone*: if a set cannot pass the test, all its supersets will fail the same test as well. Such a property is used during candidate generation to prune the set of candidates.

In our context, the sets are combinations of MCDs and the elements are the MCDs. A candidate solution (a combination of MCDs) is pruned if one of the following conditions holds: (i) the candidate solution forms a rewriting, (ii) there is at least one redundant MCD in the candidate solutions, (iii) there are two MCDs in which source definitions are conflicting, (iv) the score of the MCDs does not satisfy a given threshold. Notice that, contrarily to *Apriori*, the solutions we search for are part of the pruned elements, i.e. solutions are subsets of the so-called *negative border*.

The first two conditions ensure that the produced candidate rewritings are not redundant. When a combination of MCDs forms a candidate rewriting, all query subgoals are covered and any new MCD will always be redundant. Candidate rewritings are stored in order to be returned by the algorithm as result of the rewriting process.

The third condition eliminates a candidate if it contains conflicting predicates expressed on attributes mapped to the same query attributes.

For the fourth pruning condition, we need a monotonically increasing scoring function. We use the fact that when a new MCD is added to a combination of MCDs, the set of profile predicates which are *excluded* by the MCDs is increasing (due to the fact that only conjunctive queries are considered). The *excluded* profile predicates for a combination of MCDs is the union of the profile predicates *excluded* by each MCD separately. A profile predicate is *excluded* by an MCD if (i) it is conflicting with the source definitions of the MCD or (ii) it cannot be expressed on the source of the MCD chosen to cover the virtual relation to which the predicate is bound. The relevance of the excluded profile predicates by a combination of MCDs is called a *penalty*. This function is anti-monotone as needed by *Apriori* algorithm. Consequently, if a combination of MCDs is irrelevant (i.e. its penalty is greater than a given threshold), then all candidates containing this combination will be irrelevant.

The algorithm for the combination of MCDs is shown in Figure 4. It begins by computing the penalties of the MCDs so that those which do not satisfy the penalty threshold are pruned. The algorithm uses the *AprioriGen* function to generate combinations of MCDs

[1]. At each level, the redundant combinations are pruned before computing the penalties. Then, combinations which do not satisfy the penalty threshold and candidate rewriting are pruned. The algorithm ends when there are no more combinations to explore.

Algorithm CombineMCDs

```
Input:  MCD: set of generated MCDs
        Pu: user profile
        penalty({mcdi},P) : penalty function for a set
                            of MCDs
        ρ: penalty threshold
Output: Relevant candidate rewriting CR
Begin
   i=1
   Li = {m ∈ MCD | penalty(m, Pu) ≤ ρ }
   While Li ≠ ∅
      Ci+1 = AprioriGen(Li)
      For each e ∈ Ci+1
         if( e is not redundant ∧
             e is not conflicting ∧
             penalty(e, Pu) ≤ ρ )
               if( e is a rewriting )
                     CR = CR ∪ {e}
               else
                     Li+1 = Li+1 ∪ {e}
      i = i+1
   endWhile
   Return CR
End
```

Figure 4. MCDs combination algorithm

Table 2. MCDs for Q_e

MCD ID	SOURCE	SUB-GOALS
A	WORLDHOTELS	3
B	PROMOHOLYDAYS	1,3
C	LYONHOLYDAYS	1
D	LYONHOLYDAYS	3
E	SNCF	2
F	PLANETRANSPORT	2
G	RIDEEVERYWHERE	2

Consider the expanded query Q_e in Example 8 and the user profile P_1 in Example 7. The penalty function used hereafter is the weighted coverage of the excluded predicates (see section 4.1). The predicate groups of P_1 used to compute the penalty are {c,d}, {e,f,g,h}, {i,j,k}. Assume that the weights of the profile predicates are independent of the number of joins necessary to relate them to the query (i.e λ=1) and let the penalty threshold be set to 0.5. The MCDs generated for Q_e are presented in Table 2. The source PROMOHOLYDAYS must be used to cover both subgoals HOTEL and TRAVEL because of the join on the variable "hid" which is existential in its definition.

Figure 5 presents the execution of the combination algorithm applied to the set of MCDs in Table 2. In order to make the figure clearer, the edges between nodes of different levels are not represented. Nodes in level *i* represent combinations of *i* MCDs and the numbers above the nodes are penalty thresholds. In the first level of Figure 5, the penalty of each MCD is computed and only relevant MCDs are combined. In our example, all initial MCDs satisfy the penalty threshold. Consequently, all possible combinations of two MCDs between seven have to be checked. The redundant combinations such as AB or EF are pruned and the penalties of the rest of the combinations are computed. Then, combinations which do not satisfy the penalty threshold (for example BE) and candidate rewriting (BF and BG) are pruned. Finally, all combination generated in the third level are candidate rewritings and the algorithm stops. In this example, at the third level, we have to check only two MCDs combinations instead of the thirty five possible combinations of three MCDs among seven.

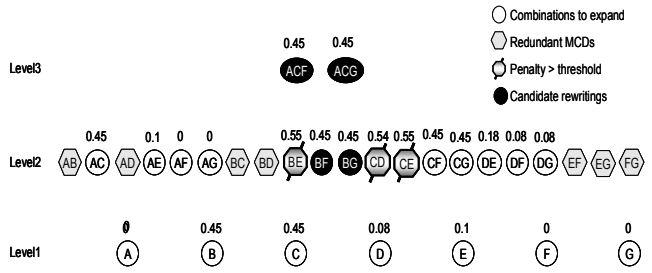

Figure 5. Example of MCDs combination

The result of the hybrid query reformulation approach in this example is a set of query rewritings which can be enriched by at least 50% of the useful profile predicates.

Actually, the last phase of the reformulation algorithm is done using the algorithm presented in [15] to which we have provided all the necessary and relevant profile knowledge to make an effective query enrichment of all the produced rewritings.

5.3 Experiments

In order to evaluate the query reformulation approaches, we used a small but representative test bed composed of the virtual schema of Example 1, the source definitions of Example 2, and a sample of 4 user profiles and 10 queries. These profiles and que-

ries were constructed by varying the number of predicates they contain and the virtual relations to which these predicates are bound. More details on the tests carried out can be found in [14]. The values of coverage and precision obtained by the reformulation approaches are presented in tables 3 and 4.

Table 3. Coverage of profile predicates

Profiles	P1			P2			P3			P4		
Queries	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$
Q1	0,63	0,82	0,45	0,91	0,91	0,91	0,86	0,86	0,86	0,90	0,90	0,60
Q2	0,82	1,00	0,36	1,00	1,00	0,77	1,00	1,00	0,67	1,00	1,00	0,43
Q3	0,82	1,00	0,82	0,73	0,73	0,73	0,86	0,86	0,86	1,00	1,00	1,00
Q4	0,72	0,91	0,54	0,91	0,91	0,91	0,91	0,91	0,91	0,90	0,90	0,60
Q5	0,82	1,00	0,36	0,73	0,73	0,50	0,86	0,86	0,52	1,00	1,00	0,43
Q6	0,82	0,82	0,64	0,91	0,91	0,91	0,71	0,71	0,71	1,00	1,00	0,62
Q7	0,82	0,82	0,36	0,73	0,73	0,50	0,71	0,71	0,38	1,00	1,00	0,43
Q8	0,73	0,73	0,46	0,83	0,83	0,59	0,91	0,91	0,52	0,76	0,76	0,49
Q9	0,82	1,00	0,54	1,00	1,00	0,77	1,00	1,00	0,67	0,85	0,85	0,59
Q10	0,72	0,72	0,72	0,66	0,66	0,66	0,62	0,62	0,62	0,90	0,90	0,90

Table 4. Precision of profile predicates

Profiles	P1			P2			P3			P4		
Queries	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$	$\mathcal{R}/\mathcal{P}$	$\mathcal{R}(\mathcal{E})$	$\mathcal{E}(\mathcal{R})$
Q1	1,00	0,88	1,00	1,00	0,88	1,00	1,00	1,00	1,00	1,00	0,90	1,00
Q2	1,00	0,89	1,00	1,00	0,90	1,00	1,00	1,00	1,00	1,00	0,91	1,00
Q3	1,00	0,89	1,00	1,00	0,88	1,00	1,00	1,00	1,00	1,00	0,91	1,00
Q4	1,00	0,88	1,00	1,00	0,88	1,00	1,00	1,00	1,00	1,00	0,89	1,00
Q5	1,00	0,89	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
Q6	1,00	1,00	1,00	1,00	0,86	1,00	1,00	1,00	1,00	1,00	0,90	1,00
Q7	1,00	1,00	1,00	1,00	0,86	1,00	1,00	1,00	1,00	1,00	0,90	1,00
Q8	1,00	1,00	1,00	1,00	0,89	1,00	1,00	1,00	1,00	1,00	0,89	1,00
Q9	1,00	0,88	1,00	1,00	0,89	1,00	1,00	1,00	1,00	1,00	0,89	1,00
Q10	1,00	1,00	1,00	1,00	0,75	1,00	1,00	1,00	1,00	1,00	0,88	1,00

The results materialize the intuitive analysis which we had done in section 3. In the majority of the cases, the $\mathcal{R}(\mathcal{E})$ and the $\mathcal{R}/\mathcal{P}$ approaches have the same profile coverage. The unique situation, where the values of their coverage differ, occurs when the profile contains predicates which cannot be rewritten (for example the predicate (d) in P_1 of Example 7). In fact both approaches consider all profile predicates not conflicting with the initial query but $\mathcal{R}/\mathcal{P}$ works better as it detects profile predicates which cannot be rewritten. Consequently, the $\mathcal{R}/\mathcal{P}$ approach has the best coverage considering only rewritable profile predicates. The coverage obtained by the $\mathcal{E}(\mathcal{R})$ approach is at most equal to that of the $\mathcal{R}/\mathcal{P}$ approach. This is due mainly to the fact that in the $\mathcal{E}(\mathcal{R})$ approach the rewriting process has defined once for all the target data sources; consequently, only the profile predicate concerning these sources can be used.

According to her respective precisions, the reformulation approaches $\mathcal{E}(\mathcal{R})$ and $\mathcal{R}/\mathcal{P}$ have the same 100% precision. Indeed, in this case, enrichment is made on the candidate rewritings of the initial query for $\mathcal{E}(\mathcal{R})$ (or expanded query for $\mathcal{R}/\mathcal{P}$) which contain the predicates of the source definitions. All the needed information for the identification of the useful predicates (predicates of the initial query, user profile predicates and predicates of the source definitions) is thus available. In contrast to $\mathcal{E}(\mathcal{R})$ and $\mathcal{R}/\mathcal{P}$, the $\mathcal{R}(\mathcal{E})$ approach does not allow to take into account the source definitions in the choice of the profile predicates for the enrichment process. There are two types of useless profile predicates not detected by the $\mathcal{R}(\mathcal{E})$ approach: (i) predicates which are satisfied by all candidate rewritings and (ii) predicates which cannot be rewritten.

In conclusion, the $\mathcal{R}(\mathcal{E})$ approach has the best profile coverage but can use useless predicates for enrichment. Its critical point is the use of predicates which cannot be rewritten. In contrast to the $\mathcal{R}(\mathcal{E})$ approach, the $\mathcal{E}(\mathcal{R})$ approach takes into account only useful predicates, but does not allow to use all of them. This approach misses any predicate which cannot be expressed on the candidate rewritings of the initial query and does not allow to take into account the predicates which are either conflicting or satisfied by all candidate rewritings. Finally, the $\mathcal{R}/\mathcal{P}$ approach has the best coverage according to the rewritable profile predicates and has the best possible precision. It considers profile predicates which are either conflicting or satisfied by all candidate rewritings, by assigning a greater penalty score to candidate rewritings which do not satisfy those predicates.

6. CONCLUSION

We proposed here a query rewriting algorithm which takes into account information about user preferences stored in his profile. This work is well adapted in a context where data sources are distributed such as in a mediation system. Moreover, two metrics were defined to quantitatively evaluate this algorithm with respect to the part of the user profile it can use and to the relative importance of this part. Based on these metrics, an experiment showed the interest of this algorithm compared to two naive approaches. This work has to be extended to several directions. First of all, we are currently performing experiments to validate the approach by considering the results of the execution of the rewritings given by our algo-

rithm. Other experiments have to be done to measure the execution times of the algorithm. Our preliminary results are promising, showing that the personalization cost is acceptable. Apart from the experiments, much work remains to be done in considering the impact of other type of preferences on the rewriting process.

ACKNOWLEDGEMENT

This research was partially supported by the French Ministry of Research and New Technolologies under the ACI program devoted to Data Masses (ACI-MD), project #MD-33.

7. REFERENCES

[1] Agrawal, R., and Srikant, R. Fast algorithms for mining association rules. In *Proceedings of the International Conference on Very Large Data-Bases (VLDB)*, Santiago, Chile, 1994.

[2] Bosc, P., Pivert O. SQLf : A Relational Database Language for Fuzzy Querying. IEEE Transactions on Fuzzy Systems, vol. 3, No 1, 1-17, 1995.

[3] Bouzeghoub, M., Kostadinov D. Personnalisation de l'information : aperçu de l'état de l'art et définition d'un modèle flexible de profils. Dans *les actes de la seconde COnférence en Recherche d'Informations et Applications (CORIA)*, Grenoble, France, 2005.

[4] Bright, L., Raschid L. Using Latency-Recency Profiles for Data Delivery on the Web. In *Proceedings of the 28th Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[5] Calvanese, D., Lembo D., Lenerini M. *Survey on methods for query rewriting and query answering using views.* Technical report, University of Rome, Roma, Italy, 2001.

[6] Chakrabarti, K., Garofalakis M., Rastogi R., Shim K. Approximate Query Processing Using Wavelets. In *Proceedings of 26th International Conference on Very Large Data Bases,* Cairo, Egypt, 2000.

[7] Duschka, O., Genesereth M. Answering Recursive Queries Using Views. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Conference on Principles of Database Systems, PODS,* Tucson, AZ, 1997.

[8] Eisenstein, J., Puerta A. Adaptation in Automated User-Interface Design. In *Proceedings of the International Conference on Intelligent User Interfaces*, LA, USA, 2000.

[9] Halevy, A., Pottinger R, MiniCon: A scalable algorithm for answering queries using views. Very Large Data Bases Journal, Vol. 10, 182-198, 2001.

[10] Han, J., and Kamber, M. Data Mining: Concepts and Techniques. 2nd ed. Morgan Kaufmann, 2006.

[11] Heer J., Newberger A., Beckmann C., Hong J. liquid: Context-Aware Distributed Queries. UbiComp 2003: Ubiquitous Computing Journal, 2003.

[12] Hwang, F. K., Richards, D. S., and Winter, P. The Steiner Tree Problem. Elsevier, North-Holland, 1992.

[13] Kießling W. Foundations of Preferences in Database Systems. In *Proceedings of the 28th Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[14] Kostadinov D., Bouzeghoub M., Lopes S. *Accès personnalisé à des sources de données multiples.* Technical Report, Laboratoire PRiSM, Université de Versailles, France, 2006.

[15] Koutrika, G., Ioannidis Y. E. Personalization of Queries in Database Systems. In *Proceedings of the 20th International Conference on Data Engineering*, Boston, Massachusetts, USA, April, 2004.

[16] Koutrika, G., Ioannidis Y. E. Personalized Queries under a Generalized Preference Model. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, Tokyo, Japan, April 5-8, 2005.

[17] Lacroix, M., Lavency P. Preference:Putting More Knowledge into Queries. In *Proceeding of the 13th Very Large Data Bases Conference (VLDB)*, Brighton, 1987.

[18] Levy, A. Y., Rajaraman A., Ordille J. J. Querying Heterogeneous Information Sources Using

Source Descriptions. In *Proceedings of the 22nd Very Large Data Bases Conference*, Bombay, India, 1996.

[19] Li, C., Chang K., Ilyas I., Song S. RankSQL: Query Algebra and Optimization for Relational Top-k Queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Baltimore, USA, 2005.

[20] Naumann, F., Freytag J.C., Spiliopoulou M. Quality Driven Source Selection Using Data Envelope Analysis. In *Proceedings of the MIT Conference on Information Quality (IQ'98)*, Cambridge, USA, 1998.

[21] Qian, X. Query folding. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, New Orleans, Louisiana, February 26 - March 1, 1996.

[22] Shearin, S., Lieberman H. Intelligent Profiling by Example. In *Proceedings of Conference on Intelligent User Interfaces,* ACM Press, Santa Fe, NM, USA, 2001.

[23] Soukane, A. *Génération automatique de requêtes de médiation avec prise en compte des besoins des utilisateurs dans un environnement hétérogène*. PhD Thesis, Université de Versailles-Saint-Quentin-en-Yvelines, December, 2005.

[24] Takahashi, H., Matsuyama A. An approximate solution for the Steiner problem in graphs. Mathematica Japonica, 24, 1980, 573-577.

[25] Vidal, M.E., Raschid L., Marquez N., Cardenas M., Wu Y. Query Rewriting in the Semantic Web. In *Proceedings of the 22nd International Conference on Data Engineering Workshops, ICDE 2006*, Atlanta, GA, USA, 2006.