Similarity Between Queries in a Mediator

Alain Bidault, Christine Froidevaux and Brigitte Safar[1]

Abstract. Answering queries on remote and heterogeneous sources is a complex task. Mediators provide techniques that exploit a domain ontology. To propose a cooperative approach to repair queries without answers, we introduce the notion of *solution* which is a query close to the user's query. We describe how ontologies can be used to evaluate the *similarity* of two predicates and then of two queries. The work presented has been developed in the PICSEL[2] project. Examples come from the domain of tourism, the experimentation domain chosen for this project.

1 Introduction

In recent years, several information integration systems (Information Manifold [10], PICSEL [5], SIMS [2]) have been developed, based on a *mediator* architecture which provides a *uniform* query interface to multiple and possibly heterogeneous data sources. Users put queries in terms of a set of relations designed to capture the semantics of a given application domain. Those relations are *virtual* in the sense that their instances are not directly available. Thus, answering a query means translating a user query into queries that refer directly to the relevant sources, which needs a set of *source descriptions*.

The most important advantage of a mediator is that it enables users to focus on specifying their demand, by freeing them from having to find the relevant sources and possibly to combine data from multiple sources to obtain answers.

The need for a *cooperative* query answering process is especially crucial because users do not know the contents of the data sources that are available. In particular, it may happen that the user's query, while being meaningful w.r.t the domain model, has no answer because its translation leads to specialized query plans (constructed by the mediator) that violate the constraints specifying the actual contents of the sources. It is important to *explain* him why his query fails (e.g., the user asked for hotels located in UK and hotels provided by the sources connected to the mediator are located in Spain). In addition, it is very useful to offer him a new query, called a *solution,* which is semantically close to the initial one and which can be answered (e.g., Bed&Breakfasts in UK).

In this paper, we consider the problem of repairing queries without any answer, due to a lack of sources, taking a logical framework to represent the domain and the sources.

Our contribution is twofold. First, we show in section 3 how to generate predefined queries, starting from the sources' description. Second, we present in section 4 a method to calculate similarity between two predicates and then between two queries. We begin with the specification of our logical framework, called the *domain theory*.

[1] L.R.I., UMR 8623 C.N.R.S & University of Paris-Sud Bâtiment 490, 91405, Orsay Cedex, France {bidault, chris, safar}@lri.fr

[2] PICSEL is supported by France Telecom R&D under contract number 97 1B 378.

2 Representation of Domain, Sources and Queries

2.1 Domain Knowledge

The knowledge domain is expressed by means of a declarative representation of object classes (Country, Flight, Stay, Travel...) and of relations among these classes. It is described using atoms of the form $p(\bar{X})$, where p is a relation name and $\bar{X}$ a tuple of variables or constants. We distinguish some unary relations, called *concepts*. They represent object classes relevant to the application domain. $C(x)$ is called an *atom-concept* if C is a concept.
The Semantics used are standard 1st order logic semantics (see [3]).

The domain knowledge $(\mathcal{D}, \mathcal{C})$ contains two components:

• A set $\mathcal{D}$ of **rules** of the form: $p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})$, where $p_i(\bar{X}_i), q(\bar{Y})$ are atoms, with $\bar{Y} \subseteq \cup_{i \in [1..n]} \bar{X}_i$.
The rules in $\mathcal{D}_h \subseteq \mathcal{D}$ describe a *hierarchy* between the domain concepts and are of the form: $C_1(x) \Rightarrow C_2(x)$, where C_1 and C_2 are concepts. Concepts and relations not appearing in rules as conclusions are called *base relations*. Relations that are not concepts are called *nc-predicates*.

• A set $\mathcal{C}$ of **constraints**, $c : l_1(\bar{X}_1) \wedge \ldots \wedge l_n(\bar{X}_n) \Rightarrow \bot$ where $l_1(\bar{X}_1), \ldots, l_n(\bar{X}_n)$ are literals, with at most 1 negative.
$\mathcal{C}_t \subseteq \mathcal{C}$ describes, for every nc-predicate, which concept **types** each of its arguments. $c_t : p(\bar{X}) \wedge \neg C(x_i) \Rightarrow \bot$, with $x_i \in \bar{X}$.

2.2 Description of the Sources

The contents of a source $\mathcal{S}_i$ are represented by means of a vocabulary $\mathcal{V}$ containing as many local relations v_{ij}, called *views*, as we know the source $\mathcal{S}_i$ gives instances of domain relations. The description of sources in terms of views contains two components:
A logical set of implications $\mathcal{D}_v \subseteq \mathcal{D}$, linking each view to a domain relation, $v_i(\bar{X}) \Rightarrow p(\bar{X})$, and **a set of constraints** $\mathcal{C}_v \subseteq \mathcal{C}$ characterizing the view instances: $l_1(\bar{X}_1) \wedge \ldots \wedge l_n(\bar{X}_n) \Rightarrow \bot$, where $l_1 \ldots l_n$ are base relations and/or view names or their negation.

2.3 Queries and Rewritings

We perform conjunctive queries of the form: $Q(\bar{X}) : p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n)$, where p_i is a predicate of $\mathcal{D} \cup \mathcal{C}$ that is not a view, and where $\bar{X} = \bar{X}_1 \cup \ldots \cup \bar{X}_n$ represents data expected by the user.
In a mediator approach we do not access the data source contents but only an abstract description of them. Answering a query consists in searching the different rewritings of it in terms of *views* of $\mathcal{V}$. A rewriting of a query $Q(\bar{X})$ is a query that logically implies $Q(\bar{X})$, using the domain theory $\mathcal{T} = \mathcal{D} \cup \mathcal{C}$. In the following, $\mathcal{T}$ always denotes the domain theory.

Definition 2.1: *Let $Q(\bar{X})$ and $Q_{\mathcal{R}}$ be two conjunctive queries and $\mathcal{T}$ be the domain theory.*

- $Q_{\mathcal{R}}(\bar{X})$ *is a* **rewriting** *of* $Q(\bar{X})$ *iff*
  $Q_{\mathcal{R}}(\bar{X}) \cup \mathcal{T}$ *is satisfiable and* $Q_{\mathcal{R}}(\bar{X}), \mathcal{T} \models Q(\bar{X})$
- $Q_{\mathcal{R}}(\bar{X})$ *is a* **terminal rewriting** *of* $Q(\bar{X})$ *iff*
$Q_{\mathcal{R}}(\bar{X})$ *is a rewriting of* $Q(\bar{X})$ *and a conjunction of atom-views.*

The set of the terminal rewritings constitutes the set of the specialized query plans that permits to obtain the answers of the initial query.

2.4 Position of the Problem

In our context of the mediator PICSEL [5], we use an algorithm implemented in java to get the terminal rewritings. It consists in expanding the initial query. An expansion step is a backward chaining step on the rules of $\mathcal{T}$.
These rewritings are the nodes of a tree rooted by Q. While developing the tree, the satisfiability of each node $Q_{\mathcal{R}}$ with $\mathcal{T}$ is tested.

Definition 2.2: *A* **Success-Query** *is a query with at least one terminal rewriting. A Query is a* **Failing Query** *iff it is not a Success-Query for* $\mathcal{T}$.

When all the leaves are unsatisfiable with $\mathcal{T}$, i.e., when there is no terminal rewriting of the user's query, we have to propose him other satisfiable queries.
Unsatisfiable queries have already been studied in [3]. In this paper, we restrict our study to **Failing Queries, satisfiable with** $\mathcal{T}$. These queries cannot be rewritten in terms of views for lack of satisfactory sources.
In the rest of the paper, we will define, for the user's query Q, the notion of **Solution** for Q. Godfrey [6] shows that looking for all the Failing Subqueries and finding all their satisfiable generalisations are NP-hard problems, which can become polynomial when exhaustiveness is not required. Instead of having a complete but expensive process, we prefer an approximate method, based on a set of *Predefined Success-Queries*. Thus, the solutions proposed to the user will be queries of this set, evaluated as being close enough to Q.
Now, we show how to build a base of queries starting from the description of the sources; then we explain how to evaluate a similarity between two predicates and between two queries.

3 Initialization of the Base

Solutions are issued from a base of predefined success-queries $\mathcal{B}_{pq}$ initialized from the descriptions of the sources. A Success-Query is built from one same source which provides instances of its atoms: thus, we make sure that at least this source can entirely rewrite the query. Thanks to the mediator approach, we can realize such an initialization by widely exploiting the description of the sources.
We briefly present here the initialization of $\mathcal{B}_{pq}$. In this paper, we do not tackle the problem of its updating (e.g., by taking into account the number of instances of queries after an effective consultation of the sources, or their relevance for the user -validation step-).
Furthermore, we propose a static approach that disregards the fact that the sources contents quickly evolve. To overcome the potential and punctual lack of answers for one source we aim to define queries that can be rewritten in different sources.

Potential Number of Rewritings of a Predicate
To favor rewriting of the queries in many different sources, we build queries from the predicates that can be rewritten in a maximal number of views, called *most common-used (c-u) predicates*.

Definition 3.1: *Let* N_1 *and* N_2 *be the numbers of views that can be used to rewrite two predicates* p_1 *and* p_2. p_1 *is* **more common-used** *than* p_2 *iff* $N_1 \geq N_2$.

ex: Consider the descriptions $(\mathcal{D}_v, \mathcal{C}_v)$ of the sources S_1 that provide hotels in luxurious places or in Sahara and S_2 providing gîtes in Corsica:
$\mathbf{r_1} : V_{S11}(x) \Rightarrow Hotel(x)$ $\quad$ $\mathbf{r_2} : V_{S12}(x,y) \Rightarrow Located(x,y)$
$\mathbf{r_3} : V_{S13}(x) \Rightarrow Sahara(x)$ $\quad$ $\mathbf{r_4} : V_{S14}(x,y) \Rightarrow Located(x,y)$
$\mathbf{c_1} : V_{S12}(x,y) \wedge \neg V_{S11}(x) \Rightarrow \bot$ $\quad$ $\mathbf{c_2} : V_{S12}(x,y) \wedge \neg V_{S13}(y) \Rightarrow \bot$
$\mathbf{c_3} : V_{S14}(x,y) \wedge \neg V_{S11}(x) \Rightarrow \bot$
$\mathbf{c_4} : V_{S14}(x,y) \wedge \neg LuxuriousPlace(y) \Rightarrow \bot$

$\mathbf{r_5} : V_{S21}(x) \Rightarrow Gîte(x)$ $\quad$ $\mathbf{r_6} : V_{S22}(x,y) \Rightarrow Located(x,y)$
$\mathbf{c_5} : V_{S22}(x,y) \wedge \neg V_{S21}(x) \Rightarrow \bot$
$\mathbf{c_6} : V_{S21}(x) \wedge Located(x,y) \wedge \neg Corsica(y) \Rightarrow \bot$

Assume that $\mathcal{D}_h$ contains the two rules $Hotel(x) \Rightarrow ResidencePlace(x)$, $Gîte(x) \Rightarrow ResidencePlace(x)$ and that there are only S_1 and S_2. Located ($N_1 = 3$) is more c-u than hotel or gîte ($N_2 = 1$), but ResidencePlace ($N_3 = 2$) would be more c-u than Sahara ($N_4 = 1$).

Queries Quota for a Source
Once the most c-u predicates have been defined, we have to specify the number of queries to generate for some fixed source. As available sources on the domain (ex: *eurostar, thalys, dégriftour*) are of unequal sizes in their contents and description, the quota for a source S depends on the number of views proposed by S.
Let N_B be the number of queries of the predefined base we expect; a **global quota** per view q_B is defined as a ratio $q_B = N_B/N_vs$, with N_vs the number of views among all the sources.
ex: Now, suppose that we have 3,200 sources having 5 views per source. If N_B is equal to $4,000$, $q_B = 4,000/(3,200 \times 5) = 0.25$.
The **quota** q_S for each source S is equal to $n_vs \times q_B$, with n_vs the number of views of S.

3.1 Construction of Predefined Queries

The q_S predefined queries of a source S are built starting from a set P_S that contains the predicates for which a source S gives instances. As the number of queries to generate is limited, first, we generate P_{Su}, a subset of P_S, that contains the most c-u predicates of the source, and second, we specialize these subqueries, according to the constraints of the source. To define how many predicates have to be selected, for S, a function f(q_s) is introduced and returns the average number of views needed in order to generate q_s queries.
It may be noted that a predicate can be rewritten by different views from a same source S. This number is called **weight** of the predicate for S. The number of predicates in P_{Su} is increased iteratively, picking the next most c-u predicate of P_S, until the number of views f(q_s) is outnumbered by the sum of their weights.

Definition 3.2: *Let* S *be a source,* q_S *its quota and* P_S *be the set of predicates for which it gives instances. Let* $P_{Su} \subseteq P_S$. P_{Su} *is a set of* **quite common-used** *predicates of* S, *if* p_k *is one of the least common-used predicates of* P_{Su} *and:*
- for all p_j *of* $P_S - P_{Su}$, p_k *is more c-u than* p_j*;*
$- \sum_{p_i \in P_{Su} - \{p_k\}} weight(p_i) < f(q_S) \leq \sum_{p_i \in P_{Su}} weight(p_i)$

ex: Suppose that we have $f(q_{S1}) = 3$ and an order over predicates, calculated among all the sources, such that *Located* is more c-u than *Hotel* which is more c-u than *Sahara*.
We have: P_{S_1u} = {Located, Hotel}. Indeed, weight(Located) = 2, weight(Hotel) = weight(Sahara) = 1. If we add Sahara to P_{S_1u}, sum of the weights = 4 and if we remove Hotel, the sum would go down to 2.

Due to a lack of place, we only present the main steps of the algorithm to generate the predefined queries for a source S:

1. Determinate the set of predicates P_{Su} for S.
   ex: Rules r_1, r_2 and r_4 process predicates of $P_{S_1 u}$ and correspond to the set of views $\mathcal{E}_v = \{V_{S11}, V_{S12}, V_{S14}\}$.
2. Generate subqueries that only contains predicate from P_{Su}, studying constraints of S to determine the substitutions.
   ex: As c_1 only takes into account views from $\mathcal{E}_v$, Q_1 = Located(x, y) $\wedge$ Hotel(x) is generated.
   Furthermore, the view V_{S14} is not used in a constraint with other views from $\mathcal{E}_v$; the unique atom conjunction Q_2 = Located(x, y) is generated.
3. Saturate the constraints of S to specialize the subqueries obtained in 2, and still allowing them to be rewritten by S.
   ex: Constraints that contain all their atoms in Q_1, except for a predicate, are processed. The constraint c_2 adds the predicate Sahara in Q_1.
   Q'_1 = Located(x, y) $\wedge$ Hotel(x) $\wedge$ Sahara(y)
   From Q_2 we get Q'_2 = Located(x, y) $\wedge$ Hotel(x) $\wedge$ LuxuriousPlace(y).
4. There are 2 possible situations:
   - the number of queries outnumbers the quota of S: we suppress some of these queries;
   - q_S is not reached: we add the next more c-u predicate of P_S to P_{Su} and go back to step 2.

Initialization of the base is performed by a prototype implemented in java. It permits to generate from a set of sources an appropriate number of queries.

3.2 Organisation of the Queries

Face with a failing-query Q_u, we need to find quickly, among the predefined queries, solutions for Q_u. We present here the indexing method concisely. It is used to cluster the queries in order to avoid searching all over the base $\mathcal{B}_{pq}$.
In general, cluster analysis deals with a set is defined given a set $\mathcal{E}$ of N_B samples in a sample space E containing $\mathcal{E}$.
We use the *k-medoids* method [8] to cluster the queries relative to a special thematic, which is an adaptation for the 1st order logic of the *k-means* algorithm [4] introduced in the propositional case. The k clusters minimize the overall sum of squared distances between instances and clusters' centers. Indeed, this method optimizes an existing clustering and approximates actual cluster centers by taking existing samples, i.e., in our approach, predefined queries from $\mathcal{B}_{pq}$. This method fits to an easy updating process of the base. In our case, methods described in section 5 are used to estimate the distances.
Thus, we select clusters of $\mathcal{B}_{pq}$ that maximize the similarity between Q_u and their center, to compare their queries with Q_u.

4 Similarity Between Predicates

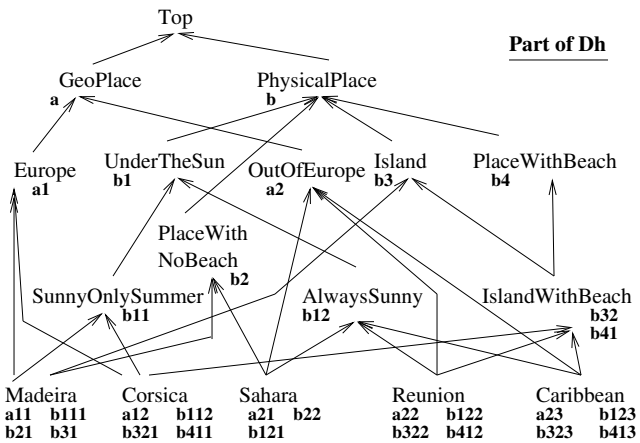
This section shows how to order predicates w.r.t. a central predicate, thanks to a similarity measure between concepts of a hierarchy.

4.1 Intuition

The calculation of similarity is based on the position of the concept in the hierarchy and is done to help the user to find a solution.
Thus, our calculation must verify the following assumptions:
- as each concept contains **characteristics (cics)**, the more characteristics two concepts share, the closer they are considered.
- as specialisations of a concept give instances of this concept and are taken into account in the expansion step, they have to be considered closer to the concept than its subsumers are.

In order to respect these assumptions, we suppose that any edge of the hierarchy displays a characteristic, that are divided up in **defined characteristics** and **not-defined characteristics** for a concept.

Definition 4.1: *The set of* **defined characteristics** *of a concept C is the set of the occurrences of edges that constitutes the differents paths $Ch_1^\top$, ...,$Ch_k^\top$ from the root Top ($\top$) of the hierarchy to C.*



Let P_h be the depth of the hierarchy. The number of cics of a concept is $k \times P_h$, where k is the number of paths $Ch_i^\top$.
It amounts to considering that each path $Ch_i^\top$ has been extended by a virtual path $Ch_i^\perp$ from C to $\perp$, such that the length of any path issued from the concatenation of $Ch_i^\top$ and $Ch_i^\perp$ is equal to P_h.
The set of these virtual paths represents the set of **not-defined characteristics** of C and is used in the computation of the similarity.

ex: *IslandWithBeach (Iwb)* has 2 paths from $\top$, the one with *Island (Isl)*, $Ch^\top_{Isl}$ the other one with *PlaceWithBeach (Pwb)*, $Ch^\top_{Pwb}$. Both are 3 edges long, thus *Iwb* has 6 defined cics. Each path is extended by a virtual path such that the full path is 4 cics long ($P_h = 4$); thus, *Iwb* has 2 not-defined cics (length($Ch^\perp_{Isl}$) + length($Ch^\perp_{Pwb} = 1 + 1$), and 8 cics ($6 + 2 = 2 \times 4$).

In practical terms, paths from $\top$ to each concept C are represented by an intrinsic numeration of all the concepts of the hierarchy.

4.2 Numbers and Identifier of a Concept

Definition 4.2: *Let $\mathcal{D}_h$ be a hierarchy, a* **number** nb *for a concept is a unique string for $\mathcal{D}_h$, defined on an alphabet of n symbols. The number of its characters $|nb|$ is called* **length**.

The chosen alphabet has to be wide enough to take into account the maximal number of sons that a concept can have in $\mathcal{D}_h$. In our examples, n = 36 $(0, 1, .., 9, a, b, .., z)$ seems to be satisfactory.
A concept can have many numbers if it has many fathers, or if at least one of its fathers has many numbers.

Definition 4.3: *The* **identifier** *of a concept C, $id(C)$ is the set of its numbers.*

All the sons of a concept C inherit the identifier of this concept and for each son, each number of id(C) inherited is extended on the right by a distinct character. The method used to numerate the hierarchy is inspired by topological sorting (the hierarchy is a DAG rooted in Top). We only impose the unicity of each number.

ex: The three concepts Corsica, Reunion and Caribbean inherite of IslandWithBeach [b32, b41]. Here, numbers are extended by characters '1' for Corsica, '2' for Reunion and '3' for Caribbean.

Numbers of the id of C are assigned to a unique path from $\top$ to C. A number $c_1...c_n$ is assigned to a path being n edges long such that the ith edge links the concept that has in its id the number $c_1...c_{i-1}$ to the concept with $c_1...c_{i-1}c_i$. The sum of the lengths of the numbers of a concept identifier is equal to the number of its defined cics.

4.3 Note of Similarity

We define the **note of similarity** of a concept C' **centered on** a concept C, noted $N_{C'\to C}$, to order concepts C' from the closest to the furthest of C. The similarity between a pair of identical objects is 1, and when there is no commonality between these two objects, their similarity is 0. Notes take into account: (i) the common characteristics of the common subsumers of C and C', which requires to compare all the numbers of the identifiers of C and C', (ii) the characteristics of C' that C does not have.

We use the following notations: the identifier of the central concept C is noted $id(C)$ = [n_1, ..., n_k] and $id(C')$ = [m_1, ..., m_l]. $+$ represents the string concatenation symbol. For 2 numbers, n_α and m_β, with $\alpha \in [1..k]$ and $\beta \in [1..l]$, we note $com_{\alpha\beta}$ their longest common prefix. We have the following concatenations: $n_\alpha = com_{\alpha\beta} + end_{n_\alpha}$ and $m_\beta = com_{\alpha\beta} + end_{m_\beta}$.

The calculation of the note consists in three stages:
1) We define the **similarity ratio of** m_β **centered on** n_α, with negative values forced to 0:

$$R_{m_\beta\to n_\alpha} = ((|com_{\alpha\beta}| + P_h - |n_\alpha|)/P_h) - 0.002 \times |end_{m_\beta}|.$$

2) Then, for each number n_α, we define a **similarity ratio of** C' **centered on** n_α, $Rmax_{C'\to n_\alpha}$, that measures the maximal similarity of the different numbers of C' centered on n_α:

$$Rmax_{C'\to n_\alpha} = \text{MAX}\{R_{m_\beta\to n_\alpha}, \beta \in [1..l]\}.$$

3) The **note of similarity of** C' **centered on** C is forced to 0 when there are no common defined cics, e.g., for *Top* centered on any concept, and otherwise is equal to:

$$N_{C'\to C} = \text{AVG}\{Rmax_{C'\to n_\alpha}, \alpha \in [1..k]\}.$$

4.4 Examples and Justifications

ex: Here is the calculation of the note $N_{Sahara\to Reunion}$ between the concept C' = *Sahara* and the central concept C = *Reunion*, with $P_h = 4$.
id(*Sahara*) = [a21, b22, b121] id(*Reunion*) = [a22, b122, b322, b412]
$R_{a21\to a22} = (|a2| + 4 - |a22|)/4 - 0.002 \times |1| = 0.748$
$R_{b22\to a22} = (0 + 4 - 3)/4 - 0.002 \times 3 = 0.244$ $R_{b121\to a22} = 0.242$
$Rmax_{Sahara\to a22} = 0.748$ $Rmax_{Sahara\to b122} = 0.748$
$Rmax_{Sahara\to b322} = 0.246$ $Rmax_{Sahara\to b412} = 0.246$
$N_{Sahara\to Reunion}$ = (0.748 + 0.748 + 0.246 + 0.246) / 4 = 0.497

• In the definition of $R_{m_\beta\to n_\alpha}$, $|com_{\alpha\beta}|$ represents how many common defined cics the numbers n_α and m_β have. Keeping for any n_α, the maximum of these ratios is equivalent to consider the longest common prefix of the m_βs with this n_α, that is a least common subsumer (lcs) for C' and C, according to the cics expressed in n_α.
Indeed, for 2 concepts C_1 and C_2, numbers corresponding to the lcs of C_1 and C_2 are obtained by keeping the set of the longest prefixes com_{12} to C_1 and C_2. Then, we just have to find the concepts identified by **at least one** of the com_{12}s.

ex: The maximal common prefixes to the numbers of *Sahara* and *Reunion* are $\{a2\}, \{b12\}$, they correspond to the numbers of the concepts *OutOfEurope* and *AlwaysSunny*, i.e., their 2 lcs.

• The number $|com_{\alpha\beta}|$ is then increased by the number of *not-defined* cics of n_α, $P_h - |n_\alpha|$. Indeed, not-defined cics of n_α are common to any of its subsumers.
• The division by P_h corresponds to a normalization stage that contributes to obtain ratios between 0 and 1.
• The first step of the calculation captures intuition (i) that similarity is related to the number of common cics of the lcs.
• According to intuition (ii), cics of C' that C does not have are taken into account by introducing a surcharge that depends on the length of the suffix of m_β, $|end_{m_\beta}|$.

ex: *PlaceWithBeach* has 4 specializations: first *IslandWithBeach*, and then *Corsica*, *Reunion* and *Caribbean*:
$N_{IslandWithBeach\to PlaceWithBeach}$ = (4/4 - 0.002) /1 = 0.998.
$N_{Cors.\to PWB} = N_{Reun.\to PWB} = N_{Carib.\to PWB}$ = 1 - 0.004 = 0.996.

4.5 Similarity Between Nc-Predicates

We are now interested in predicates that are not concepts. The calculation of similarity is based on the number of their arguments and on the constraints that type them. When there is no constraint for an argument, the note 0 is assigned to this argument.

Definition 4.4: *Let $p_1(x_1, ..., x_n)$, $p_2(y_1, ..., y_m)$ be nc-predicates. Let $C_i, i \in [1..n]$ (resp. $C'_j, j \in [1..m]$) be the concepts that type the arguments x_i (resp. y_j). The* **note of similarity of** p_2 **centered on** p_1 *is:* $N_{p_2\to p_1} = \text{AVG}_{i\in[1..n]}\{\text{MAX}_{j\in[1..m]}\{N_{C'_j\to C_i}\}\}$.

ex: Consider the 5 following constraints of $\mathcal{C}_t$ for the predicates p_1 = EquipAss that associates an Equipment to a ResidencePlace and p_2 = DistEquip that expresses the distance of an Equipment to a fixed GeoPlace: DistEquip(x,y,z) $\wedge \neg$ GeoPlace(x) $\Rightarrow \perp$
DistEquip(x,y,z) $\wedge \neg$ Equip(y) $\Rightarrow \perp$ DistEquip(x,y,z) $\wedge \neg$ Number(z) $\Rightarrow \perp$
EquipAss(x,y) $\wedge \neg$ ResPlace(x) $\Rightarrow \perp$ EquipAss(x,y) $\wedge \neg$ Equip(y) $\Rightarrow \perp$

Suppose we have the notes of similarity shown on the table, one C_i per row and one C'_j per column. Using this table, where the best notes are in bold font, we get $N_{DistEquip\to EquipAss} = (1 + 0.4)/2 = 0.7$.

EqtAs \ DistEq	GeoPlace	Equip	Number
ResPlace	**0.4**	0.3	0
Equip	0.1	**1**	0.2

5 Similarity Between Queries

We present in this section how to compare any query Q_p of $\mathcal{E}$ to Q_u. Similarity between queries is based on proximity between predicates, but can be reduced to a calculation of similarity between 2 conjunctions of concepts, for concepts that refer to the same objects (same variables). Thus, it must also take into account: (i) the difference between the names of the variables from the user's query Q_u and from predefined queries Q_p, (ii) links between variables.

ex: Due to a lack of place, see [3] for details and full examples. The user asks for a stay in an hotel in Sahara and a stay in Corsica, from june to august:
$Q_u = Stay(hu1, bu, eu, su, pu1) \wedge Stay(hu2, bu, eu, cu, pu2) \wedge Hotel(hu1) \wedge june(bu) \wedge august(eu) \wedge Sahara(su) \wedge Corsica(cu)$
Q_{p1} and Q_{p2} have the same predicates, but Q_{p2} requires a unique hotel:
$Q_{p1} = Stay(\mathbf{h1}, \mathbf{b}, \mathbf{e}, m, p1) \wedge Stay(\mathbf{h2}, \mathbf{b}, \mathbf{e}, m, p2) \wedge luxhotel(\mathbf{h1}) \wedge hotelwithswpool(\mathbf{h2}) \wedge june(\mathbf{b}) \wedge july(\mathbf{e}) \wedge Madeira(m)$
$Q_{p2} = Stay(\mathbf{h}, \mathbf{b1}, \mathbf{e1}, m, p1) \wedge Stay(\mathbf{h}, \mathbf{b2}, \mathbf{e2}, m, p2) \wedge luxhotel(\mathbf{h}) \wedge hotelwithswpool(\mathbf{h}) \wedge june(\mathbf{b1}) \wedge july(\mathbf{b2}) \wedge Madeira(m)$

Substitutions We do not want to calculate neither all the substitutions on the queries, nor notes of similarity for each substitution to get the best note. Therefore, we define heuristics to choose a substitution within the candidates.

Algorithm:
1. Unfold the queries Q_p and Q_u to get conjunctions of concepts Qu_p and Qu_u, by replacing the nc-predicates with their types.
2. Compare the concepts of the predefined query Qu_p with Qu_u's ones; this comparison is directed in order to evaluate which atoms from Qu_u have to be associated to atoms of Qu_p.
3. Obtain Qu'_p by applying the substitutions that match the two closest predicates in Qu_p and Qu_u, starting by the first predicate with the best note of similarity, stopping when all the atoms from Qu_p are processed.

The note of Q_p centered on Q_u is the average among the notes obtained for the atoms of Qu'_p, centered to the atoms of Qu_u. This time, notes are calculated in the other direction as we want to find which queries Q_p from the base $\mathcal{B}_{pq}$ are the closest ones to Q_u.
A first note is calculated comparing the atoms that refer to the same objects. To remedy the fact that only 1 substitution is chosen, a second note is worked out for every atom of Qu_u. This note only takes into account the name of the concepts, disregarding the arguments, but performs a surcharge [3].

ex: $N_{Q_{p1} \to Q_u}$ = 0.778 $N_{Q_{p2} \to Q_u}$ = 0.740. The notes are different because Qu'_{p1} contains an atom, $ResPlace(h2)$, that Qu'_{p2} does not. This atom arises in the unfolding step as the first argument of *Stay* differs in the two occurrences in Q_{p1}.
Not to consider the variables would lead to consider Q_{p1} and Q_{p2} as a unique query, whereas they express 2 different objectives.
Different thresholds can be defined explicitly (by the user) or not (by default); they may be, for example, the minimal note that determines whether a predefined query is close to the user's query, i.e., is a **solution** for Q_u, or the number of solutions to present to the user.

6 Comparison with Other Similarity Measures

In many proposals, similarity measures between two concepts C_1 and C_2 in a taxonomy are distance based (e.g., using the length of the shortest path from C_1 to C_2, through one of their least common subsumers). But these measures (i) are symmetrical and (ii) respect the triangle inequality, properties that do not fit our intuitions in the context of semantic similarity. Concerning (i) we proved in [3] that, to have specializations closer than subsumers, the similarity measure cannot be symmetrical. For example, with a rule of $\mathcal{D}_h$: *Reunion* $\Rightarrow$ *OutOfEurope*, $N_{Reunion \to OutOfEurope} > N_{OutOfEurope \to Reunion}$ because *OutOfEurope* must be further than any specialization of *Reunion*. Note that all the other measures presented here are symmetrical. Concerning (ii) specifying that the distance between two concepts (e.g., old dog and young cat) must be shorter than the sum of the two distances with a third concept (e.g., young dog): **d**(o.d., y.d.) + **d**(y.d., y.c.) $\geq$ **d**(o.d, y.c.) may be counter-intuitive.
In some other proposals [9, 11] the measure is based on the notion of shared information and uses a function of probability of encountering an instance of a concept in a corpus, data that we do not have.
We now study properties introduced by Lin in [9] on similarity measures: (iii) their maximal value is equal to 1, (iv) they increase with commonality, (v) they decrease with difference and (vi) if two concepts can be viewed from two independent perspectives and if their similarity can be computed separately from each perspective, their overall similarity is a weighted average of the similarities computed from every perspective. Resnik's measure [11] does not verify (iii)(v)(vi). The work done by Wu & Palmer [12] is very closed to ours; it takes into account the deepest *lcs* in the hierarchy and bases its calculation on the depth of this lcs and its distance to the concepts: $depth(lcs) \times 2$ / $(dist_{lcs}(C_1) + dist_{lcs}(C_2) + depth(lcs) \times 2)$. However, their measure, as all the processes based on a hierarchy, requires to scan the whole hierarchy for any calculation while our concepts identification system avoids such an expensive process. Another important problem with their measure is that, for example, the concept *OutOfEurope* would be closer to *Reunion* than *Caribbean* is, which is counter-intuitive. Their measure, Lin's approach and our notes satisfy properties (iii) to (vi). In addition, our measure and Lin's process can be applied for ordinal values and in order to find common radicals in the context of word similarity. In this context, we could generate the identifiers of the words using only their letters, without requiring any taxonomy.
Our computation of similarity between two queries can be compared with Andreasen's work [1] in the context of document retrieval, but restricted to conjunctions of concepts (instead of predicates). The metric on clauses of Hutchinson [7] is based on the often used *Hausdorff metric*, computed from a table of $n \times m$ values, if the 2 clauses (or concepts) have n and m characteristics. Only one of these values is chosen, i.e., 1 *path* for predicates, 1 atom for clauses. Furthermore, all the substitutions of the initial clause are processed in order to select one of them, whereas we avoid this calculation and we take into account all the *lcs* of the concepts.

7 Conclusion and Perspectives

Our objective is to help a user of a mediator system to reformulate his query when it fails. For it we have introduced the notion of solution. This paper has presented in a formal framework an algorithm that generates a base of predefined queries specified from a description of the sources. Moreover, we have introduced the notion of characteristics for a concept in order to define similarities between two predicates and therefore between two queries. An algorithm to calculate these solutions has been proposed. It is implemented in Java.
Several experimentations still have to be done to determine the size of the base of predefined queries and the number of clusters. We also have to precise in an interactive way the minimal similarity note that allows us to conclude that two queries are closed enough.
Concerning the update of the base, proposals issued from Case Base Reasoning community will be investigated.

REFERENCES

[1] T. Andreasen, J.F. Nilsson and H.E. Thomsen. Ontology-Based Querying, FQAS'2000, pp.15-26, Warsaw, Poland.
[2] Y. Arens, C. A. Knoblock, W.-M. Shen. Query reformulation for dynamic information integration, In *J. Intelligent Information Systems*, Kluwer, 6(2), pp. 99-130, juin 1996.
[3] A. Bidault, C. Froidevaux, B.Safar. Repairing Queries in a Mediator Approach, In *Proc. of ECAI 2000,* pp. 406-410, 2000.
[4] Bock, Diday. Analysis of Symbolic Data, Springer-Verlag Ed 2000.
[5] F. Goasdoué, V. Lattès, M.-C. Rousset. The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project, In *Int. J. of Cooperative Inf. Syst.* (IJCIS), 1999.
[6] P. Godfrey. Minimization in Cooperative Response to Failing Databases Queries, In *Int. J. of Cooperative Inf. Syst.*, IJCIS'97, 6(2), pp. 95-149.
[7] A. Hutchinson. Metrics on Terms and Clauses, In ECML'97, pp. 138.
[8] M. Kirsten, S. Wrobel. Extending k-means Clustering to 1st-Order Representations, In *Inductive Logic Prog.*, LNAI 1866, pp. 112, 2000.
[9] D. Lin. An Information-Theoretic Definition of Similarity, In *Proc. of the Int. Conf. on Machine Learning (ICML-98)*, pp. 296-304, 1998.
[10] A. Y. Levy, A. Rajamaran, J. Ordille. Query-answering algorithms for information agents, In *Proc. of AAAI conf.*, pp. 40-47, 1996.
[11] P. Resnik. Semantic Similarity in a Taxonomy: An Information-based Measure and its Application to problem of Ambiguity in Natural Language, In *J. of Artificial Intelligence Research*, (11) pp. 95-130, 1999.
[12] Z. Wu and M. Palmer. Verb Semantics and Lexical Selection, In *Proc. of 32nd Meeting of the Ass. for Computational Linguistics*, 1994.