Error Propagation in Distributed Databases

O. Haase, A. Henrich

Praktische Informatik, Fachbereich Elektrotechnik und Informatik
Universität Siegen, D-57068 Siegen, Germany
Email: {haase,henrich}@informatik.uni-siegen.de

Abstract

An interesting research area for distributed database systems is inaccessibility. Even if part of the database is inaccessible, a query should be evaluated reasonably. This means, that appropriate rules for propagation and minimization of errors induced by inaccessibility are needed. Furthermore, the result of a query in a distributed database system must be regarded to be potentially vague.

In this paper we present a hybrid representation for vague sets, that consists of (1) an enumerating part, which contains the elements we could access during query processing, and (2) a descriptive part, which describes the relevant elements we could not access. Further we introduce propagation rules, which can be used to minimize the vagueness of a query result represented in this hybrid way. Thereby, we propose different levels of accuracy for the descriptive part.

As far as we know this is the first approach using a descriptive part to qualify the missing elements of a result. An example for the usefulness of the descriptive part arises, if we have to calculate the intersection of two subqueries. Here, the descriptive part of the first operand can be used to check for each element of the second operand, whether it should be part of the first operand, and vice versa.

1 Introduction

One main characteristic of distributed database management systems is that due to network or machine failure, the environment may become partitioned into sub-environments that cannot communicate with each other. In this case, the sub-environments should remain operable, in particular queries to the database should be processed in an appropriate way. To this end, inaccessibility has to be concerned during query processing. On the one hand, this means that an expressive error report has to be assigned to the computed result in order to clarify the reasons for failure to the user. On the other hand, adequate error propagation rules have to be used that keep the error as small as possible during query processing.

In this paper, we focus on the problem of error propagation, aiming towards error minimization.

In: N. Pissinou, A. Silberschatz, E.K. Park, K. Makki (Hrsg.), Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM '95), ACM Press, S. 387-394, Baltimore, Maryland, USA, December 1995

With respect to a query, evaluated on a database with inaccessible parts, the database can be logically divided into three parts: (1) those elements, which surely fulfill the query, (2) those which surely do not fulfill the query, and (3) those, which may or may not fulfill the query, depending on the inaccessible parts of the database – e.g. because the element itself is located on the inaccessible parts of the database or because a condition for an element located on the accessible parts of the database cannot be evaluated because this would require access to inaccessible parts of the database.

We can represent these three sets by the 'egg-yolk'-model [CG94]. The egg yolk represents the set of surely contained elements, the egg-white the set of possibly contained elements, and the outside represents the set of surely not contained elements. This gives a graphical impression of the vague sets we deal with in this paper.

Obviously in a distributed database taking inaccessibility into account, not only the final result of a query, but also all intermediate results must be regarded to be potentially vague. Hence, we have to define rules to combine vague sets by extensions of the usual set operators like $\cap$, $\cup$ and $\setminus$, and furthermore, we need rules to handle the typical query language (QL) operators like e.g. selection.

To this end, we will show in section 4 how to propagate the vagueness expressed by vague sets through the different operations. We do so using the characteristic function κ_V to describe a vague set V, with $\kappa_V(i) = 1$ for all elements i in the egg yolk, $\kappa_V(i) = u$ for all elements i in the egg-white, and $\kappa_V(i) = 0$ for all elements i outside the egg. If we use the values 1, 0 and u in the sense of a three valued logic, this representation is extremely handy in order to state the propagation rules and relations on vague sets and therefore useful to define the semantics of a QL taking inaccessibility into account. Furthermore, as we will see in section 5, this formalism allows to check the relation to the result of a query for a given element i of the database directly simply by calculating $\kappa_V(i)$.

Nevertheless, the characteristic function given above is, in fact, just another representation for the stated query itself and when we put a query to a database, we are indeed not interested in yet another descriptive representation of our query, rather we look for an enumerating one, which states the result explicitly.

A convenient representation of this enumerating result would e.g. be, to hold a vague set V as a pair of sets, namely the lower bound V_y and the upper bound V_e; where V_y contains the elements i with $\kappa_V(i) = 1$, i.e. the **y**olk in the 'egg-yolk'-model, and V_e contains the elements i with $\kappa_V(i) \neq 0$,

i.e. the whole egg.

Unfortunately most elements i with $\kappa_V(i) = u$ have value u just because they are inaccessible. In these cases we are not able to give the complete set V_e explicitly. We therefore represent V_e in a hybrid way: Those elements we can access are added to the enumerating part $\widehat{V}_e$ of V_e and the others are described by the function δ_V which states the relation to the vague set V for all elements of the database not contained in $\widehat{V}_e$, i.e. we represent V_e as a pair $(\widehat{V}_e, \delta_V)$.

The descriptive part (δ_V) of a vague set V can be used to further improve $\widehat{V}_e$ during query processing, i.e. to check for additional elements whether they can be inserted into $\widehat{V}_e$. This can, e.g. be done when we combine V with a second vague set W. Here we can check for each element j of $\widehat{W}_e$, which is not element of $\widehat{V}_e$, whether it can be added to $\widehat{V}_e$.

It has to be mentioned, that κ_V itself is a possible choice for δ_V, but we are indeed not forced to describe the missing elements of a vague set V by the function κ_V itself. Instead, we can use a less restrictive descriptive function in order to reduce the effort when applying the descriptive function. The price we may have to pay is a more vague[1] – but nevertheless correct – result.

The paper is organized as follows: Section 2 gives a short survey of related work. Section 3 introduces the data model and the query language we use as example environment. In section 4 we propose the propagation rules for vague sets by means of the characteristic function κ. In section 5 the techniques to revise vague sets represented in the proposed hybrid way during query processing are described. Section 6 concludes the paper.

2 Related Work

In the context of relational databases, much work has been done in order to deal with the problems resulting from null values. To be able not only to express that a value is not available, but to distinguish different reasons of being not available, one has to use different null values. Gessert proposes the use of a four valued logic [Ges90, Ges91] to distinguish 'values not known' and 'values not applicable'.

A more sophisticated approach is to describe unknown values ('values not known') by fuzzy sets, i.e. a set of values plus their probability of being the missing one. A survey of the work related to fuzzy sets and further references can be found in [DPY93], while the usage of fuzzy sets for the processing of uncertain queries is collected by Motro in a comparing article [Mot90].

Lipski [Lip79] distinguishes the internal and the external interpretation of a query. The external one corresponds to the answer given if the real world would be completely known. The internal one in turn is the one given if there is some information missing. Missing information always concerns atomic attribute values. An unknown atomic value is represented by a set of possible values. Hence, results are internal represented as a set of objects surely fulfilling the query (the inner limit), and a set of objects possibly fulling the query (the outer limit); these sets correspond to our explicit parts V_y and $\widehat{V}_e$. Because Lipski investigates missing information at the level of atomic attributes, he does not need to concern QL operations, like e.g. selection, on the sets described above. He is only interested in the usual set operations. Besides, in his context, the upper bound of a result set can always be stated explicitly; a fact which is not true in the case of inaccessibility in distributed systems.

Wong [Won82] extends Lipski's approach by statistical information about the distribution of attribute values. The result of a query is the set of records, which fulfill it with an adjustable probability α. If α tends towards 0, Lipski's outer limit is computed; if $\alpha = 1$, Lipski's inner limit is computed. But as Lipski before, Wong assumes accessibility of all records, and therefore has no need for a hybrid representation.

Morrissey [Mor90] uses, as Lipski before, two sets to represent the possible result elements. But for the set of possible elements, he uses fuzzy sets. This means that every possible element has its value of possibility assigned, what allows a ranking of these elements. But Morrissey too does not consider incomplete upper bounds, other operations than the usual set operations, and relations on uncertain sets.

Imieliński and Lipski [IL84] define conditions that must be fulfilled by the semantics of a relational algebra, in order to allow the correct computation of a query involving unknown values. A query is computed correctly, in this sense, if all possibilities for the meaning of the unknown values are contained in the result. Imieliński and Lipski define so-called representation systems, that consist of a subset of the usual relational algebra, the class of allowed unknown values, and the extension of the used relational operators to these unknown values. The representation systems in turn guarantee the correct computation described above. But this work strictly refers to the relational data model instead of an object oriented model, and it does not take inaccessibility into account; thus there is no need to use a descriptive part for the missing items.

Cohn and Gotts use the so-called 'egg-yolk'-representation for geographical regions with undetermined boundaries [CG94]. This corresponds to the graphical impression of vague sets used in this paper. But because of their application area Cohn and Gotts neither use a hybrid representation nor do they consider the QL operators considered in our paper.

3 Example Environment

Although our considerations neither depend on a special query language nor on a given data model, we introduce a concrete environment as a basis, which will be used for the examples throughout the paper. To this end we refer to the data model of PCTE, the ISO and ECMA standard for an open repository [ECM93, WJ93], and to NTT, an algebraic set-oriented query language for PCTE [Haa95].

PCTE has a semantic data model, based on the entity relationship model. An object base consists of objects, which are connected by links. Objects are typed. The object type hierarchy builds a directed acyclic graph with one root, the object type *object*. In the usual way, an object of a type *ot* automatically is also of all types that lie on a path between *object* and *ot*. Object types determine the set of applied attributes, which are atomic, and the set of applicable link types.

A link type specifies, among other things, a list of attribute types called key attributes, a set of non-key attribute types, a set of allowed destination object types and its reverse link type; that is because links normally form bi-directional relationships by having a reverse link.

Since the PCTE-OMS is designed as a distributed OMS, objects and links are stored on segments and there is no guarantee that all segments are accessible at a given time.

[1] A formal definition of what is meant by 'more vague' will be given in section 4.

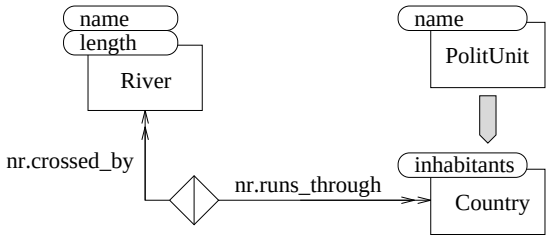

Figure 1: Example schema

In figure 1 small part of a PCTE schema is given in the typical PCTE schema diagram style. It consists of three object types **River**, **PolitUnit** and **Country** with the attributes **name**, **length** and **inhabitants**. Subtype relationships between object types are shown by broad shaded arrows, being directed from the supertype to the subtype. Additionally a pair of link types **runs_through** from **River** to **Country** and **crossed_by** from **Country** to **River** is given. The double arrowheads on the end of the links indicate that the links have cardinality *many*. Both links have a natural key named **nr**. The number used as key for a link of type **runs_through** should indicate the order in which the countries are crossed by the river. Hence, the country associated by the link `1.runs_through` is the country where the river rises.

The **n**on-navigating **t**ool **t**ongue NTT is an algebraic approach for a set-oriented query language for the PCTE data model. We use it here to show the effects of our error propagation rules, therefore we describe only those aspects of the language which are essential for this paper. NTT operates on vague sets of objects, which in turn are typed.

Two ways to specify a start set for further query formulation are provided by NTT. One way are user defined input sets, which are passed to the API of NTT. The other way is the operator **ext**, that computes the extension of an object type. This extension consists of all objects in the object base, which are of the given type. E.g. the NTT query $\mathcal{Q}$: `ext(River);` would compute all rivers in the whole object base. Both facilities to define start sets are handled homogeneously, and need not be distinguished in the following.

Other operators always get a vague object set as input and return a vague object set as output. Totally, there are five such QL-operators (besides the usual set operations *union, intersection*, and *difference*). In this paper we consider only two of them.

The first one is the **select** operator. It returns all objects of the input set, which fulfill a given predicate. Predicates can be logically combined, they can be quantified, and they can contain further subqueries. The query $\mathcal{R}$:

```
select(ext(River), length > 1000);
```

would compute all rivers longer than 1000 km.

The second operator is **relatives**. It allows the set-oriented navigation from an input object set via a given regular link name to a result object set. The result contains all objects that can be reached from at least one input object via at least one link that matches the regular link name. A regular link name is a PCTE link name, which may contain wild cards for the key attributes. A string key can contain the wild cards '`*`' and '`?`', with the meaning 'arbitrary many' and 'exactly one' symbol, respectively. A natural key can be an exact number or an interval, where the upper bound can be missing. Therefore the query $\mathcal{S}$:

```
relatives(select(ext(River), length > 1000),
          1.runs_through);
```

would return all countries, where at least one river longer than 1000 km rises. And finally, the query $\mathcal{T}$:

```
select(relatives(select(ext(River),
                        length > 1000),
                 1.runs_through),
       inhabitants < 50000000);
```

computes all countries with more than 50 million inhabitants, where at least one river longer than 1000 km rises.

4 Propagation Rules

When we talk about 'set-oriented' query languages, we actually mean 'collection-oriented'. Query languages typically support three different kinds of collections: *sets*, *multisets* and *lists*. As mentioned above, if we take inaccessibility and other sources of incorrectness into account, we get vague collections. Due to space limitations we can only deal with vague sets here.

4.1 Vague Sets

Obviously, a vague set can be envisaged as a set of sets where every element is a set, which could be a correct answer to the computed query.

In the following we write $S \in \mathbf{Set}(T)$, if S is a set over the foundation set T.

We describe a vague set V by two sets V_y and V_e with the property $V_y \subseteq V_e$. If V is a vague set over the foundation set T, we denote $V \in \widetilde{\mathbf{Set}}(T)$. Each $A \in \mathbf{Set}(T)$, that lies between V_y and V_e, is contained in V. This is expressed by the definition: $A \in V \overset{\text{def}}{\Longleftrightarrow} V_y \subseteq A \subseteq V_e$.

The representation by a lower bound V_y and an upper bound V_e is equivalent to the description of a vague set by its characteristic function κ_V. This function defines for each $i \in T$, if i is in the egg yolk, in the egg-white, or outside:

$$\kappa_V : T \to \{1, 0, u\}$$
$$\kappa_V(i) = \begin{cases} 1 \text{ , if } i \in V_y \\ 0 \text{ , if } i \notin V_e \\ u \text{ , otherwise} \end{cases} \quad (1)$$

Representing vague sets by their characteristic function κ enables us to express our error propagation rules on top of a three valued logic in a short, easy and straightforward way, as we will see in the following.

4.2 Base Operations on Vague Sets

Now we adapt the usual set operations to vague sets. The situation is as follows: As input we have some vague sets, each of them representing a set of possible correct sets. And we are looking for a vague set as result, which represents the set of all possible result sets, with the property of the result vague set to be as *strict* as possible. A vague set V is stricter than another vague set W ($V \prec W$), iff (1) $V_e \setminus V_y \subset W_e \setminus W_y$ (i.e. if in V the number of uncertain elements is less than in W) and (2) $V_y \supseteq W_y$ and (3) $V_e \subseteq W_e$. A vague set V is called *exact*, iff there is no vague set W, that is stricter than V (i.e. if $V_y = V_e$).

e_1	e_2	$\neg_3 e_1$	$e_1 \vee_3 e_2$	$e_1 \wedge_3 e_2$	$e_1 \Rightarrow_3 e_2$	$e_1 \Leftrightarrow_3 e_2$
0	0	1	0	0	1	1
0	u	1	u	0	1	u
0	1	1	1	0	1	0
u	0	u	u	0	u	u
u	u	u	u	u	u	u
u	1	u	1	u	1	u
1	0	0	1	0	0	0
1	u	0	1	u	u	u
1	1	0	1	1	1	1

Table 1: Value table for three valued logical junctions

$$\forall_3 i : P(i) \equiv \begin{cases} 1 \text{ , if } \forall i : P(i) \equiv 1 \\ 0 \text{ , if } \exists i : P(i) \equiv 0 \\ u \text{ , otherwise} \end{cases}$$

$$\exists_3 i : P(i) \equiv \begin{cases} 1 \text{ , if } \exists i : P(i) \equiv 1 \\ 0 \text{ , if } \forall i : P(i) \equiv 0 \\ u \text{ , otherwise} \end{cases}$$

Table 2: Value table for three valued logical quantifiers

In order to state the characteristic functions for the vague set operations, we use the three valued logic given in table 1 [2].

If $\circ_3 \in \{\cup_3, \cap_3, \setminus_3\}$, and $X = V \circ_3 W$, the fact, that κ_X describes the strictest vague set containing all sets C for which there is a set $A \in V$ and a set $B \in W$ with $C = A \circ B$, is equivalent to the conjunction of the two conditions:

$$\forall i : \quad \kappa_X(i) \equiv 1 \Leftrightarrow \forall A \in V, B \in W : i \in A \circ B \qquad (2)$$

$$\forall i : \quad \kappa_X(i) \equiv 0 \Leftrightarrow \forall A \in V, B \in W : i \notin A \circ B \qquad (3)$$

Hence, in the following it will be sufficient to show that conditions (2) and (3) hold in order to proof that the result of an operator is as strict as possible.

4.2.1 Vague Set Union

The characteristic function κ of the union of two vague sets can be expressed as the three valued disjunction of the characteristic functions of the two vague input sets:

$$X = V \cup_3 W \Rightarrow \kappa_X(i) = \kappa_V(i) \vee_3 \kappa_W(i) \qquad (4)$$

We proof conditions (2) and (3) as follows:

1. condition (2):
$$\begin{aligned} & \kappa_X(i) \equiv 1 \\ \Leftrightarrow \quad & \kappa_V(i) \vee_3 \kappa_W(i) \equiv 1 \\ \Leftrightarrow \quad & \kappa_V(i) \equiv 1 \vee \kappa_W(i) \equiv 1 \\ \Leftrightarrow \quad & (\forall A \in V : i \in A) \vee (\forall B \in W : i \in B) \\ \Leftrightarrow \quad & \forall A \in V, B \in W : i \in A \cup B \end{aligned}$$

2. condition (3):
$$\begin{aligned} & \kappa_X(i) \equiv 0 \\ \Leftrightarrow \quad & \kappa_V(i) \vee_3 \kappa_W(i) \equiv 0 \\ \Leftrightarrow \quad & \kappa_V(i) \equiv 0 \wedge \kappa_W(i) \equiv 0 \\ \Leftrightarrow \quad & (\forall A \in V : i \notin A) \wedge (\forall B \in W : i \notin B) \\ \Leftrightarrow \quad & \forall A \in V, B \in W : i \notin A \cup B \end{aligned}$$

□

4.2.2 Vague Set Intersection

The characteristic function κ of the intersection of two vague sets can be expressed as the three valued conjunction of the characteristic functions of the two input vague sets:

$$X = V \cap_3 W \Rightarrow \kappa_X(i) = \kappa_V(i) \wedge_3 \kappa_W(i) \qquad (5)$$

Proof of condition (2) and (3): analogous to vague set union.

[2] In the following we will use the index '3', whenever we refer to operations on vague sets or to three valued logical junctions, respectively. If an operator or logical junction is written without index, we always refer to the exact or two valued case, respectively.

4.2.3 Vague Set Difference

The characteristic function κ of the difference of two vague sets can be expressed as the following three valued expression:

$$X = V \setminus_3 W \Rightarrow \kappa_X(i) = \kappa_V(i) \wedge_3 \neg_3 \kappa_W(i) \qquad (6)$$

Proof of condition (2) and (3): analogous to vague set union.

4.3 Relations on Vague Sets

Query languages often use relations on sets to express some selection criterion. Therefore, we have to adapt these set relations to vague set relations.

Because a vague set is actually representing a set of possible sets, relations on vague sets have to be interpreted in a three valued manner. If all possible sets in any of the operands fulfill the given relation, we interpret it as '1'. If all possible sets in any of the operands do not fulfill the given relation, we interpret it as '0'. And if there are some possible sets that fulfill the relations, while others do not, we interpret it as 'u'.

4.3.1 Equality of Two Vague Sets

Two vague sets V and W are only equal, if they are both exact and represent the same set. They are not equal, if there is an element i that is contained in every possible set of V (i.e. $i \in V_y$), and that is not contained in every possible set of W (i.e. $i \notin W_e$), or vice versa. In all other cases, the value of the equality is unknown.

In order to state the following rules, table 2 gives the evaluation rules for three valued quantified predicates.

The conditions for the equality of two vague sets are expressed by the following logical formula:

$$(V =_3 W) \equiv (\forall_3 i : \kappa_V(i) \Leftrightarrow_3 \kappa_W(i))$$

4.3.2 Subset Relation

A vague set V is a subset of a vague set W, if all elements i, that are contained in any possible set of V (i.e. $i \in V_e$) are also contained in every possible set of W (i.e. $i \in W_y$). V is no subset of W, if there is an element i, that is contained in every possible set of V (i.e. $i \in V_y$), and that is not contained in every possible set of W (i.e. $i \notin W_e$). In all other cases, the value of the subset relation is unknown.

$$(V \subseteq_3 W) \equiv (\forall_3 i : \kappa_V(i) \Rightarrow_3 \kappa_W(i))$$

4.4 Query Language Operations

The usual set operators adapted to vague sets above, are part of most query languages; nevertheless they are no QL speciality. Now we focus on typical QL operators. To obtain a high degree of generality, we keep these operators as abstract as possible, and illustrate them by giving a concrete example in NTT. We treat two kinds of operators: selection and a so-called abstract operator.

4.4.1 Selection

When we deal with vagueness and three valued logic, a selection is a mapping

$$\sigma_3 : \widetilde{\mathbf{Set}}(T) \times (T \to \{1, 0, u\}) \to \widetilde{\mathbf{Set}}(T)$$
$$(\sigma_3(V,P))_y = \{i \in V_y | P(i) \equiv 1\},$$
$$(\sigma_3(V,P))_e = \{i \in V_e | P(i) \not\equiv 0\}$$

A selection uses a vague set V and a predicate P as input and returns a vague set $\sigma_3(V,P)$ as output. Again, the corresponding propagation rule can be formulated short and straightforward in terms of the characteristic function κ:

$$W = \sigma_3(V,P) \Rightarrow \kappa_W(i) = \kappa_V(i) \wedge_3 P(i) \qquad (7)$$

We do not need to proof the fact, that κ_W is the strictest function describing all possible results of a selection, because selection can be regarded as a particular abstract operator, as we will see in the next section.

4.4.2 Abstract Operator

No other operator is as common as selection; every QL uses selection, as a separate operator like the relational algebra or NTT, or implicit, for example integrated in the `where`–clause of a query in SQL, the ODMG proposal OQL [Cat93] or P-OQL [Hen95].

But concerning a lot of existing QL's, we think there is one principle behind several important operators: namely that a set operator actually is defined via its effect on the single elements of the set, and the result is built as the union of the results for the single elements.

We exploit this general principle to define an *abstract operator* ω, containing all these operators as special cases:

The effect on a single element can be represented as a mapping $F : T_1 \to \mathbf{Set}(T_2)$, which is used as an input to the abstract operator. In the case without vagueness, ω is defined as:

$$\omega : \mathbf{Set}(T_1) \times (T_1 \to \mathbf{Set}(T_2)) \to \mathbf{Set}(T_2)$$
$$\omega(A,F) = \{i \in T_2 | \exists j \in A : i \in F(j)\}$$

If we take into account, that both the input set and the mapping can be vague, we obtain the abstract operator ω_3[3]:

$$\omega_3 : \widetilde{\mathbf{Set}}(T_1) \times (T_1 \to \widetilde{\mathbf{Set}}(T_2)) \to \widetilde{\mathbf{Set}}(T_2)$$
$$(\omega_3(V,L))_y = \{i \mid \exists j \in V_y : i \in (L(j))_y\}$$
$$(\omega_3(V,L))_e = \{i \mid \exists j \in V_e : i \in (L(j))_e\}$$

A three valued abstract operator uses a vague set V and a mapping L as input and returns a vague set $\omega_3(V,L)$ as output.

[3] An example for a vague mapping L is the predicate of a selection, if we envisage selection as a particular abstract operator, as will be done at the end of the section.

The characteristic function κ of the result vague set is:

$$W = \omega_3(V,L) \Rightarrow \kappa_W(i) = \exists_3 j : \kappa_V(j) \wedge_3 \kappa_{L(j)}(i) \qquad (8)$$

Due to space limitations, we cannot give the proof for the strictness of the abstract operator, here. In principle, it can be done analogous to the strictness proofs in section 4.2, but it is somewhat lengthy. Hence, we refer to [HH95].

Next we give two examples that show the usage of the abstract operator:

First, we can represent selection as an abstract operator. In this case, obviously the input foundation set T_1 is the same as the output foundation set T_2. Let $W = \sigma_3(V,P)$. Here, L is given by:

$$\kappa_{L(j)}(i) = \begin{cases} P(i), \text{ if } i = j \\ 0 \quad , \text{ otherwise} \end{cases} \qquad (9)$$

As given in formula (8), $\kappa_W(i)$ is

$$\kappa_W(i) = \exists_3 j : \kappa_V(j) \wedge_3 \kappa_{L(j)}(i)$$

And with the definition in (9) we obtain

$$\kappa_W(i) \equiv \kappa_V(i) \wedge_3 P(i),$$

which is exactly the characteristic function of the selection, given in formula (7).

Our second example is the set-oriented navigation operator of NTT, called **relatives**, as introduced in section 3. Let $W = \mathbf{relatives}(V, RL)$, where V is a vague object set and RL is a regular link name. In order to state L for this example, we need two conventions: (1) The notation $j \xrightarrow{l} i$ means, that a link with name l connecting j and i exists in the database. (2) We use the regular link name RL as a synonym for the set consisting of all link names matching RL itself. Then, L can be given by

$$\kappa_{L(j)}(i) = \exists l \in RL : j \xrightarrow{l} i$$

This can be inserted into formula (8), to obtain the characteristic function for the **relatives**–operator:

$$\kappa_W(i) = \exists j : \kappa_V(j) \wedge \exists l \in RL : j \xrightarrow{l} i$$

4.5 Predicates

When we express selection criteria via first order logic, we must evaluate them three valued to deal with vagueness. The value tables of the logical junctions are listed in table 1. The rules for the evaluation of three valued quantified predicates are given in table 2.

5 Hybrid Representation of Vague Sets

In this section we show how the propagation rules stated in section 4 can be applied to vague sets given in the hybrid representation.

In order to calculate $X = V \circ_3 W$ we propose a three step approach:

1. We use δ_W to test for each element of $\widehat{V}_e \setminus \widehat{W}_e$ whether it should be in $\widehat{W}_e$, and vice versa.

2. We compute X_y and $\widehat{X}_e$ by means of V_y, W_y, $\widehat{V}_e$ and $\widehat{W}_e$. To this end, we use formula (1) to calculate the value $\kappa_V(i)$, resp. $\kappa_W(i)$, for each element $i \in \widehat{V}_e$ [4], resp. $\widehat{W}_e$. This is permissible although we actually use $\widehat{V}_e$, resp. $\widehat{W}_e$, instead of V_e, resp. W_e, because we yield the correct result for the considered elements and the elements in $V_e \backslash \widehat{V}_e$, resp. $W_e \backslash \widehat{W}_e$, are covered by the descriptive part of the hybrid representation treated under point 3. Thereafter, we apply propagation rule (4), (5) or (6) respectively.

3. We compute δ_X by means of δ_V and δ_W, which means that we apply propagation rule (4), (5) or (6) resp. to δ_V and δ_W.

The three step approach shows the interaction of the two parts of our hybrid representation.

It remains to mention that applying a QL operator like e.g. selection or the abstract operator ω_3 to a vague set V, step 2 and 3 are quite analogous. Step 1 must be dropped, because there is no other vague set beside V for which the elements can be checked against δ_V.

As we have seen in the introduction, one possible choice for δ_V is the characteristic function κ_V itself. On the other hand, the most trivial choices for the descriptive function are $\delta_V(i) = 0$ and $\delta_V(i) = u$ to state that $\widehat{V}_e$ is complete, resp. incomplete. In the remainder of this section we will show that there is a wide range of possible choices for δ_V in between. Possible levels of accuracy for δ_V will be presented in section 5.2 and in section 5.3; beforehand the influence of type information on the quality of the descriptive function is discussed in section 5.1.

5.1 Typed vs. Type-Free QL

The trivial descriptive function $\delta_V(i) = u$ not really describes V, but says, that all elements may be in V. *All* elements actually means all elements $i \in T$, where T is the foundation set of V. The smaller the foundation set T, the stricter V is described by the descriptive function. The means to restrict the foundation set is, in the context of a QL, to use typing. There is not only the distinction between typing and no typing, but one can distinguish different granularities of typing:

If a QL is type free, we know nothing about the missing elements of an incomplete vague set. They can be atomic or complex values, or even objects of the database.

On a coarse-grained level, typing distinguishes between different atomic and complex values and database objects. An incomplete vague set built over natural values, for example, misses some natural values, and nothing else.

On a fine-grained level, QL's type values as above, and additionally distinguish different types for the database objects. In this case, if a vague set of objects is incomplete, we know the missing objects to be of the corresponding object type.

NTT, for example, knows for each operator an inference rule to determine the object type for the result vague set, dependent on the types of the input vague sets. Here, we do not propose these rules; instead we apply them to our example query $\mathcal{T}$, as introduced in section 3, and explain their meaning. The innermost expression $\mathcal{Q} = \mathbf{ext}(\text{River})$ has the object type **River**. After the selection, the type remains the same, still **River**. According to figure 1, navigation via links of type **runs_through** reaches objects of type **Country**; therefore the subquery $\mathcal{S}$ has the type **Country**. Again, the type remains the same after selection, what means that the complete query $\mathcal{T}$ has the object type **Country**; i.e. if it were computed incomplete, we knew that only objects of that type could be missing.

Assume, the result of our query above[5] would be the vague set T^0 – note that we use an upper index n to denote the n-th revision of something – with $T^0_y = \{Rwanda, Canada, Mongolia, Switzerland\}$ and $T^0_e = (T^0_y, \delta^0_T(i) = u)$, i.e. $\widehat{T}^0_e$ is incomplete. Further assume, we would like to build the intersection X of T and a second vague set U, with $U_y = \{Rwanda, Canada, Australia, Portugal, Singapore, Vatican, China, Alaska\}$, and $U_e = (U_y, \delta_U = 0)$, i.e. U is an exact set.

If we assume coarse-grained typing, we only knew from both T and U to contain objects of the database instead of values. Therefore we must the whole database assume to be T_e. This means in particular, that all elements of $\widehat{U}_e$ are also in $\widehat{T}^1_e$. Using our three step approach to apply propagation rule (5), we obtain the first vague result set X^0 with $X^0_y = \{Rwanda, Canada\}$ and $X^0_e = (\{Rwanda, Canada, Australia, Portugal, Singapore, Vatican, China, Alaska\}, \delta^0_X(i) = 0)$.

If we assume fine-grained typing, we know that T has the object type **Country**. Therefore we know that T_e can only contain objects of type **Country**, which leads to the new revision T^2 with $\widehat{T}^2_e = \{Rwanda, Canada, Mongolia, Switzerland, Australia, Portugal, Singapore, Vatican, China\}$, because *Alaska* does not have the object type **Country** and cannot be contained in T_e. T_y and δ_T remain unchanged. In this situation, X^1_y and δ^1_X are the same as above, but $\widehat{X}^1_e$ results to $\{Rwanda, Canada, Australia, Portugal, Singapore, Vatican, China\}$.

And, indeed, X^1 calculated exploiting type information is stricter than X^0.

5.2 Locality Information

Until now we have only concerned type information to improve the quality of a vague set. Now, we will propose more sophisticated variants of descriptive functions.

If we know, which segments of the database were not accessible, when we built a vague set Y, we can derive an improved descriptive function δ'_Y from any other descriptive function δ_Y. To this end, we first check an element i against the locality criterion. If i resides on the accessible part, δ'_Y can be set to 0, because if it belongs to the result of the query, it would already be contained in $\widehat{Y}_e$:

$$\delta'_Y(i) = \begin{cases} \delta_Y(i) & \text{, if } i \text{ resides on an inaccessible segment} \\ 0 & \text{, otherwise} \end{cases}$$

Unfortunately, if a **relatives** operator is applied to a vague set W with an incomplete $\widehat{W}_e$, we cannot use this improvement. The reason is, that we cannot follow links originating from unknown objects, which in turn could refer to objects residing on accessible segments. On the other hand, the operator **ext**, like all operators using no navigation, allows the application of the locality improvement.

[4] Remember that $V_y \subseteq \widehat{V}_e$.

[5] In the following we use Latin capital letters to denote the result of the query with the corresponding calligraphic letter.

5.3 Back Tracing

The information we used so far to describe a vague set – type and locality information – was independent of the structure of the query.

On the other hand, in section 4 we have stated the rules to compute the characteristic function κ for a query. These rules can also be applied to the descriptive function δ for the unknown part of the result of an operation. We have already used propagation rule (5) in this way in the example given at the beginning of section 5. Hence, the application of these rules allows us to compute the best possible descriptive function δ [6].

Unfortunately the descriptive function δ computed in this way can become complicated for nontrivial queries, and if we take into account, that δ may be used during query processing to check for a lot of elements their relation to the query, it might be interesting to use a less strict descriptive function δ which can be checked with less effort. Of course, the price one has to pay for this, might be a less strict result.

One way to achieve such a less strict – and less expensive to compute – descriptive function δ is to use the trivial descriptive function $\delta(i) = u$ for some subqueries.

Now let us switch our point of view and take a look at the query from the top, to check whether an element i is part of the result of a query. Then applying the descriptive function derived by our propagation rules implicitly means to trace back the query top-down operator by operator and to check whether i could have been in the result. Furthermore using the trivial descriptive function $\delta(i) = u$ for a subquery means to stop back tracing at this subquery.

For the vague set base operations the use of the trivial descriptive function $\delta(i) = u$ for one operand simplifies the descriptive function of the result vague set. As an example, we consider the propagation rule for vague set union. If $X = V \cup_3 W$ and $\delta_V(i) = u$, we obtain

$$\delta_X(i) = \delta_V(i) \vee_3 \delta_W(i) \equiv u \vee_3 \delta_W(i) \equiv \begin{cases} 1 \text{ , if } \delta_W \equiv 1 \\ u \text{ , otherwise} \end{cases}$$

In the following we show how the use of the trivial descriptive function $\delta(i) = u$ for the operand of a QL operator influences the corresponding propagation rules.

5.3.1 Selection

If $Z = \sigma_3(Y, P)$ and $\delta_Y(i) = u$, we obtain:

$$\delta_Z(i) = \delta_Y(i) \wedge_3 P(i) \equiv u \wedge_3 P(i) \equiv \begin{cases} 0 \text{ , if } P(i) \equiv 0 \\ u \text{ , otherwise} \end{cases}$$

This rule has an intuitive interpretation: If we do not know the vague set used as the operand for the selection, an element fulfilling the predicate could be in the result; an element not fulfilling the predicate cannot be in the result, independent of the fact, whether it was in the operand vague set or not.

Let us return to our running example: we assume both vague sets T and U as given above, and we still use type information.

Because $\mathcal{T} = \sigma_3(\mathcal{S}, P)$ with $P =$ `inhabitants < 50000000`, we get the descriptive function

$$\delta_T^3(i) = \begin{cases} u \text{ , if inhabitants}(i) < 50000000 \\ 0 \text{ , otherwise} \end{cases} ,$$

[6] Best in this case means strictest, and by the strictness proofs in section 4 we have shown that the propagation rules yield the strictest possible function.

if we want to stop back tracing at subquery $\mathcal{S}$.

We can now test those elements i of $\widehat{U}_e \setminus \widehat{T}_e^0$ with type **Country**, against δ_T^3 to see, if i should also be in $\widehat{T}_e^3$. These are the elements *Australia*, *Portugal*, *Singapore*, *Vatican* and *China*. Except *China*, for which $\delta_T^3 \equiv 0$, all other elements have the value u, and are therefore in $\widehat{T}_e^3$, which results to $\widehat{T}_e^3 = \{$*Rwanda*, *Canada*, *Mongolia*, *Switzerland*, *Australia*, *Portugal*, *Singapore*, *Vatican*$\}$, while T_y and δ_T remain unchanged.

The new vague result set is given by $X_y^2 = \{$*Rwanda*, *Canada*$\}$ and $X_e^2 = (\{$*Rwanda*, *Canada*, *Australia*, *Portugal*, *Singapore*, *Vatican*$\}, \delta_X^2(i) = 0)$.

5.3.2 Abstract Operator

If $W = \omega_3(V, L)$ and $\delta_V(i) = u$, we obtain

$$\begin{aligned} \delta_W(i) &= \exists_3 j : \delta_V(i) \wedge_3 \delta_{L(j)}(i) \\ &\equiv \begin{cases} u \text{ , if } \exists j : \delta_{L(j)}(i) \not\equiv 0 \\ 0 \text{ , otherwise} \end{cases} \end{aligned}$$

This means, if there is an element j for which i could be in $L(j)$, then i could be in the result. We cannot be sure, because we do not know anything about V.

We demonstrate the back tracing of the abstract operator by continuing our running example. The subquery $\mathcal{S}$ has the form $\omega_3(\mathcal{R}, L)$ with L defined as follows:

$$\kappa_{L(j)}(i) = \begin{cases} 1 \text{ , if } \exists l \in 1.\text{runs\_through} : j \overset{l}{\rightarrow} i \\ 0 \text{ , otherwise} \end{cases}$$

Stopping back tracing at subquery $\mathcal{R}$, we get the descriptive function

$$\delta_S^1(i) = \begin{cases} u, \text{ if } \exists j : \exists l \in 1.\text{runs\_through} : j \overset{l}{\rightarrow} i \\ 0, \text{ otherwise} \end{cases}$$

Insertion of δ_S^1 into rule (5) yields a revised test criterion for T:

$$\delta_T^4(i) = \begin{cases} u \text{ , if inhabitants}(i) < 50000000 \\ \qquad \wedge \exists j : \exists l \in 1.\text{runs\_through} : j \overset{l}{\rightarrow} i \\ 0 \text{ , otherwise} \end{cases}$$

Assume, *Singapore* and *Vatican* have no incoming link that matches the regular link name `1.runs_through`, because there is no river rising, which is stored in the object base, while *Australia* and *Portugal* do. Further assume the links incoming in *Australia* would start from *Murray* and *Ord*, and the links incoming in *Portugal* would start from *Mondego* and *Zezere*.

We test now the elements of $\widehat{U}_e \setminus \widehat{T}_e^0$ with type **Country** ($\{$*Australia*, *Portugal*, *Singapore*, *Vatican*, *China*$\}$) against δ_T^4. Of course, *China* still fails the test because of the part *inhabitants*$(i) < 50000000$; *Singapore* and *Vatican* fail, because they do not have the desired incoming link. Only *Australia* and *Portugal* have a test result of u. This leads to the new revision $\widehat{T}_e^4 = \{$*Rwanda*, *Canada*, *Mongolia*, *Switzerland*, *Australia*, *Portugal*$\}$, whereas T_y and δ_T remain unchanged.

The new revision of the vague result set is given by $X_y^3 = \{$*Rwanda*, *Canada*$\}$ and $X_e^3 = (\{$*Rwanda*, *Canada*, *Australia*, *Portugal*$\}, \delta_X^3(i) = 0)$.

Now, we can complete back tracing considering the selection criterion `length > 1000`, because $\mathcal{R} = \sigma_3(\mathcal{Q}, P)$ with $P =$ (`length > 1000`).

Building the descriptive function for R, we could again stop at subquery $\mathcal{Q}$. But we do not; instead we take the fact into account, that we can describe Q completely: it is $\kappa_Q(i) = \mathbf{hasType}(\text{River}, i)$ – the meaning of the predicate **hasType** should be evident. Because we do already consider type information, this predicate is implicitly covered by the descriptive function $\delta_Q(i) = 1$.

These considerations lead to the descriptive function

$$\delta_R(i) = (\text{length}(i) > 1000)$$

Finally, we achieve a remarkable result: we can describe Q completely. Hence, the test of an element i can not only show that i is in $\widehat{Q}_e$, but it can even show that i is in Q_y. And inserting this descriptive function into the surrounding expressions, this remains true for them, too.

The new descriptive function for S results to

$$\delta_S^2(i) = \begin{cases} 1, \text{ if } \exists j : \text{length}(j) > 1000 \\ \qquad\qquad \wedge \exists l \in 1.\text{runs\_through} : j \stackrel{l}{\rightarrow} i \\ 0, \text{ otherwise} \end{cases}$$

Insertion of δ_S^2 into rule (5) yields

$$\delta_T^5(i) = \begin{cases} 1 \text{ , if inhabitants}(i) < 50000000 \\ \qquad \wedge \exists j : \text{length}(j) > 1000 \\ \qquad\qquad \wedge \exists l \in 1.\text{runs\_through} : j \stackrel{l}{\rightarrow} i \\ 0 \text{ , otherwise} \end{cases}$$

Now we test the elements *Australia*, *Portugal*, *Singapore*, *Vatican* and *China* against the new descriptive function: As before *Singapore*, *Vatican* and *China* fail the test. Because *Murray* fulfills the condition `length > 1000`, *Australia* stands the test, what now means a value of 1; i.e. *Australia* $\in T_y^5$. And because *Mondego* and *Zezere* both do not fulfill the condition `length > 1000`, *Portugal* fails the test.

The new, and last result of $T \cap_3 U$ is $X_y^4 = \{$*Rwanda*, *Canada*, *Australia*$\}$, and $X_e^4 = (X_y^4, \delta_X^4(i) = 0)$ what is indeed an exact set.

Obviously an exact result set can not always be achieved. In our example, it can be achieved because the intersection operator is applied and one input set is exact. Nevertheless significant improvements, i.e. stricter results, can be obtained in many other situations as well.

6 Conclusion

In this paper we have presented an approach to deal with vague sets resulting from inaccessibility in distributed databases. We have shown, that in this context a hybrid representation is needed, that consists of (1) an enumerating part, which contains the elements we could access during query processing, and (2) a descriptive part, which describes the relevant elements we could not access. Further we have introduced propagation rules, which can be used to minimize the vagueness of a query result represented by the hybrid representation.

Further research is needed especially with respect to lists, e.g. resulting from sorting operations. Another topic of interest are aggregation operations. First promising results suggest that our approach will be adaptable to these areas.

References

[Cat93] R. Catell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, Calif., USA, 1993.

[CG94] A.G. Cohn and N.M. Gotts. Spatial regions with undetermined boundaries. In N. Pissinou and K. Makki, editors, *Proceedings of the Second ACM Workshop on Advances in Geographic Information Systems*, pages 52–59. NIST, ACM Press, December 1994.

[DPY93] D. Dubois, H. Prade, and R.Y. Yager, editors. *Readings in Fuzzy Sets for Intelligent Systems*, San Mateo, Calif., USA, 1993. Morgan Kaufmann.

[ECM93] European Computer Manufacturers Association, Geneva. *Portable Common Tool Environment - Abstract Specification (Standard ECMA-149)*, June 1993.

[Ges90] G.H. Gessert. Four valued logic for relational database systems. *ACM SIGMOD Record*, 19(1):29–35, March 1990.

[Ges91] G.H. Gessert. Handling missing data by using stored truth values. *ACM SIGMOD Record*, 20(3):30–42, September 1991.

[Haa95] O. Haase. Ntt – a set-oriented algebraic query language for PCTE. Technical Report 95/5, Universität – GHS – Siegen, D-57068 Siegen, 1995.

[Hen95] A. Henrich. P-OQL: an OQL-oriented query language for PCTE. In *Proc. of the Seventh Conf. on Software Engineering Environments*, pages 48–60, Netherlands, 1995. IEEE Computer Society Press.

[HH95] O. Haase and A. Henrich. Error propagation in distributed databases – mathematical foundations. Technical Report 95/6, Universität – GHS – Siegen, D-57068 Siegen, 1995.

[IL84] T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal of the Association for Computing Machinery*, 31(4):761–791, October 1984.

[Lip79] W. Lipski. On semantic issues connected with incomplete information databases. *ACM Transactions On Database Systems*, 4(3):262–296, 1979.

[Mor90] J.M. Morrissey. Imprecise information and uncertainty in information systems. *ACM Transactions On Information Systems*, 8(2):159–180, April 1990.

[Mot90] A. Motro. Accommodating imprecision in database systems: Issues and solutions. *ACM SIGMOD Record*, 19(4):69–74, December 1990.

[WJ93] L. Wakeman and J. Jowett. *PCTE – The Standard for Open Repositories*. Prentice Hall, Hemel Hempstead, Hertfordshire, UK, 1993.

[Won82] E. Wong. A statistical approach to incomplete information in database systems. *ACM Transactions on Database Systems*, 7(3):470–488, September 1982.