Haase, Oliver; Henrich, Andreas

A closed approach to vague collections in partly inaccessible distributed databases

Primary publication

Haase, Oliver; Henrich, Andreas (1999): A closed approach to vague collections in partly inaccessible distributed databases, in: Johann Eder, Ivan Rozman, und Tatjana Welzer (Ed.), Advances in databases and information systems : third East European conference, ADBIS '99, Maribor, Slovenia, September 13 - 16, 1999 ; proceedings, Berlin u.a.: Springer, pp. 261–274, doi: 10.1007/3-540-48252-0_20.

A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases

Oliver Haase[1] and Andreas Henrich[2]

[1] NEC Europe Ltd, C&C Research Laboratories Heidelberg,
Adenauerplatz 6, D-69115 Heidelberg, Germany
Oliver.Haase@ccrle.nec.de
[2] Otto-Friedrich-Universität Bamberg, Fakultät Sozial- und
Wirtschaftswissenschaften,
Praktische Informatik, D-96045 Bamberg, Germany
Andreas.Henrich@sowi.uni-bamberg.de

Abstract. Inaccessibility of part of the database is a research topic hardly addressed in the literature about distributed databases. However, there are various application areas, where the database has to remain operable even if part of the database is inaccessible. In particular, queries to the database should be processed in an appropriate way. This means that the final and all intermediate results of a query in a distributed database system must be regarded as potentially vague.
In this paper we propose hybrid representations for vague sets, vague multisets, and vague lists, which address the accessible elements in an enumerating part and the inaccessible elements in a descriptive part. These representations allow to redefine the usual query language operations for vague sets, vague multisets, and vague lists. The main advantage of our approach is that the descriptive part of the representation can be used to improve the enumerating part during query processing.

1 Addressed Problem

An interesting problem in the field of distributed database systems is query evaluation when a part of the database is temporarily inaccessible. If a database is distributed on top of an unreliable communication basis, e.g. the Internet, a network or machine failure can cause such a situation. Similar conditions may occur during the intended use of the system, if part of the database is stored on portable workstations, which can be disconnected from the network arbitrarily. Such situations are e.g. considered in the distribution model of PCTE [6], the ISO standard for an open repository, and related situations have to be handled in mobile environments as described in [11].

For application areas like those mentioned above it is desirable that the accessible part of the database still remains operable. In particular queries to the database should be processed in an appropriate way. Of course, there are applications that cannot do without a complete, one hundred percent correct result. These applications must be built on a reliable network, they must use

additional techniques like replication, and last but not least they must stop query processing when the database is not completely accessible and wait for a recovered situation instead. On the other hand, there are applications for which an approximation of the result is better than no result at all.

Unfortunately, in these circumstances it is not sufficient to process queries in the usual way, and to handle inaccessibility by taking only the accessible part of the database into account. Instead there are two sources causing the necessity of adapted query processing techniques:

The first reason is that in the case of partial inaccessibility the usual two-valued logic does not meet our requirements: If the evaluation of a selection criterion needs access to inaccessible data, it cannot yield one of the truth values *true* or *false*, but has to deliver an *unknown* result. The use of a three-valued logic in turn implies the use of *vague result collections* instead of *crisp result collections*. That is due to the elements that might belong to the result but of which it is not sure if they really do.

The second reason concerns the incompleteness of the computed result: In principle all accessible sure elements of the result as well as all accessible uncertain elements of the result can be computed. To this end the corresponding query must be transformed to a so-called *query condition*. The query condition is a predicate that can be applied to a potential candidate for the result; it will evaluate to *true*, if the candidate belongs to the result, it will evaluate to *false*, if the candidate does not belong to the result, and it will evaluate to *unknown*, if the candidate may or may not belong to the result. In an absolute declarative query language, e.g. the relational calculus, queries are already formulated as query conditions. In the case of complete accessibility of the database, the result consists of all elements fulfilling the query condition.

If the database is partly inaccessible, one possible query evaluation is to check each accessible element against the corresponding query condition. By doing so, the (vague) result consists of all accessible elements that can be or really are in the desired crisp result. However, this *predicative query evaluation* is very costly, and therefore normally query evaluation is not predicative but *constructive*; i.e. the evaluation commences with some particular start elements – e.g. the extension of a certain object type – and navigates successively to the desired result elements. These navigations can involve logical relationships (e.g. formulated by means of a theta join) as well as physical relationships (e.g. relationships in an E-R-data model). If a query is evaluated constructively, partial inaccessibility may break some navigation paths from certain start elements to the corresponding result elements. These breaks can be caused by the inaccessibility of intermediate elements. Thus, a result element can be **unreachable** with respect to the execution plan whilst still being **accessible**.

Hence, constructive query evaluation in partly inaccessible databases leads to *incomplete* vague result collections. This incompleteness is a big drawback, both for the end-user and the query processing itself. For the end-user it means the system answering as follows: there are some elements fulfilling the query; additionally there are some elements *maybe* fulfilling your query; and finally each

element of the database is also a potential candidate for your query, since the computed result is incomplete. Evidently, this answer is far from being satisfying.

Incompleteness is also a problem during the query processing: Assume we have the vague results of two sub-queries, each of which being incomplete due to partial inaccessibility. If these vague sets are to be combined by a binary operation, each of both must regard the other one as consisting of potential candidates for itself. Hence, all elements occurring in at least one vague input collection must be taken into account for the vague result collection. This uncertainty leads to a growing vagueness of the final result during the query processing. Much of that vagueness is not forced by the partial inaccessibility, but by an awkward query processing. The predicative evaluation of the same query would lead to immensely less vagueness in the result. What we need is a technique that combines the low costs of a constructive evaluation with the accuracy of a predicative query evaluation.

To sum up, the situation is as follows: Partial inaccessibility of the database leads to a certain vagueness of the result. The intuitive representation of this kind of vagueness is to distinguish between the sure and the uncertain part of the result. But due to the normal constructive evaluation of a query, vague collections are in general incomplete. Thus, a representation as well as adapted operations are needed that overcome the described problems.

2 Related Work

In the area of *relational databases* much work has been done to deal with unknown or incomplete data. The basic concept is *null values* [5,18,12,13,8].

In general null values are used to cope with the vagueness of the stored information — usually induced by unknown or undefined attribute values — which may result in the vagueness of a query result. However, a fundamental difference between null values and inaccessible parts of a database is the following: a null value stands for an unknown *single attribute*; the values of the remaining attributes and the existence of the whole object (tuple) are known. If parts of a database are inaccessible, even the *existence* of objects is unknown.

Another approach to describe vague values is the use of fuzzy sets. A prerequisite for this approach is probabilistic information, such as the distribution of the concerned attributes. Some relevant papers in this respect are [3,7,15].

The articles [1,14,19,20,16] are concerned with vague sets resulting from vague attribute values – sometimes called *rough sets*. To this end, the result is approximated by a lower bound (containing all objects surely contained in the desired crisp result) and an upper bound (containing all objects potentially contained in the desired result). The most important difference between these interpretations of vague sets and our approach is that all papers mentioned above presuppose the accessibility of all relevant data. Furthermore, the mentioned papers deal with sets only, whereas multisets and lists are not considered.

The work presented in [4,17,2] deals with partial inaccessibility as follows: in this case the query processor yields a so-called *partial result* that consists

of a partly evaluated query. This query contains the used accessible parts of the database materialized as constants. It can be evaluated at a later point of time, when the accessibility situation will have changed. This work is based upon the assumption that in an unreliable environment the partial inaccessibility takes only short time and that it concerns changing nodes. Only in these circumstances it is useful to materialise the accessible data; otherwise the pure redo of the query evaluation has the same effect. The facilities to extract relevant data from the partial results are rather low. To this end the user himself has to formulate so-called *parachute queries* that operate on the partial results. Another assumption of this approach is the relational data model with no horizontal segmentation of the relations. Hence, in the case of partial inaccessibility a relation is completely accessible or it is completely inaccessible; a restriction we do not impose.

In [9] we have presented a first approach to deal with vague *sets* in the sketched scenario and in [10] a comprehensive discussion of the mathematical foundations of our representation of vague sets is given together with the correctness proofs and a short outlook on multisets. In contrast the present paper adds convenient representations for vague *multisets* and vague *lists*. Hence, we propose a *closed* approach to deal with partial inaccessibility that covers *all three kinds of collections*. Furthermore we consider aggregate functions.

3 Example Environment

Assume an object-oriented or E-R-based distributed database management system supporting links to represent relationships between objects. Further assume a schema for modelling diagrams for object-oriented analysis (OOA) methods, described as follows: An object of type *OOA_Diagram* is connected with the classes defined in the diagram via links of type *contains* representing the whole-part-relationship between the diagram and its classes. Furthermore, two attributes (*name* and *comment*) are defined for the object type *OOA_Diagram*. Five originating link types are defined for the object type *Class*. Links of the types *has_attribute* and *has_method* represent the relationships to the attributes and methods of the class. Links of the types *has_subclass* and *has_superclass* define the position of the class in the inheritance graph. Furthermore, links of type *in* lead to the diagram containing the class definition. Finally, a method is connected with its parameters via links of type *has_parameter*.

Now, let us assume that the objects stored in the database are distributed on four segments as depicted in figure 1 and that *segment 3* is inaccessible at the moment. For the following examples it is necessary to define the physical location of the depicted links. To this end let us assume that they are stored together with their origin objects, i.e. a link originating from an object located on *segment 2* resides on *segment 2*, too.

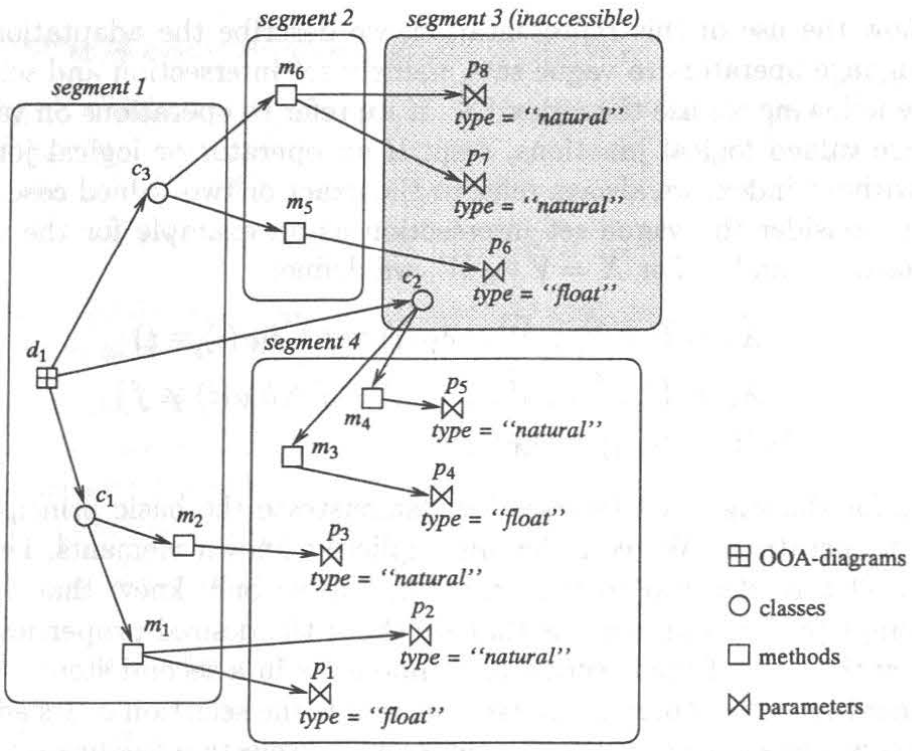

Fig. 1. An example database

4 Vague Sets

In the following we will shortly describe the representation of vague sets. For a more thorough description including illustrative examples please refer to [9,10].

A vague set V representing the result of a query processed on a partly inaccessible database can be envisaged as a set of sets. Each element is a set which could be the correct answer to the computed query, if the entire database was accessible. Informally speaking, a vague set V is defined by: (1) A set $\widehat{V}_\lambda$ that consists of the *accessible* and *reachable* elements which surely belong to the correct answer, (2) a set $\widehat{V}_\upsilon \supseteq \widehat{V}_\lambda$ that consists of the accessible and reachable elements for which we cannot be sure that they do *not* belong to the correct answer and (3) a three-valued logical predicate δ_V, the so-called descriptive component, that states for each element i of the foundation set, if it *surely* belongs to the correct answer ($\delta_V(i) \equiv t$), if it *maybe* belongs to the correct answer ($\delta_V(i) \equiv u$), or if it surely *does not* belong to the correct answer ($\delta_V(i) \equiv f$).

Formally, a vague set over the foundation set T can be defined as follows:

Definition 1 (vague sets). *The triple* $V = (\widehat{V}_\lambda, \widehat{V}_\upsilon, \delta_V)$ *with* $\widehat{V}_\lambda \subseteq T$, $\widehat{V}_\upsilon \subseteq T$, $\delta_V : T \to \{t, f, u\}$ *is a* vague set *over the foundation set* T ($V \in \widetilde{\mathbf{Set}}(T)$), *iff*

$$\widehat{V}_\lambda \subseteq \widehat{V}_\upsilon \quad \wedge \quad \forall i \in \widehat{V}_\lambda : \delta_V(i) \equiv t \quad \wedge \quad \forall i \in \widehat{V}_\upsilon \setminus \widehat{V}_\lambda : \delta_V(i) \equiv u$$

Furthermore, the sets V_λ *and* V_υ *are defined as* $V_\lambda = \{i \in T : \delta_V(i) \equiv t\}$ *and* $V_\upsilon = \{i \in T : \delta_V(i) \not\equiv f\}$. *A set* $S \subseteq T$ *is said to be contained in* V, *iff it is* between V_λ *and* V_υ: $S \in V \Leftrightarrow V_\lambda \subseteq S \subseteq V_\upsilon$. □

To show the use of this representation, we describe the adaptation of two query language operators to vague sets, namely set intersection and selection.

In the following we use the index "3", if we refer to operations on vague sets or to three-valued logical junctions, resp. If an operator or logical junction is written without index, we always refer to the exact or two-valued case, resp.

Let us consider the vague set intersection as an example for the usual set operations $\cup$, $\cap$, and $\setminus$. For $X = V \cap_3 W$, we define:

$$\begin{aligned} \widehat{X}_\lambda &= \{i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon \mid \delta_V(i) \equiv t \wedge \delta_W(i) \equiv t\} \\ \widehat{X}_\upsilon &= \{i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon \mid \delta_V(i) \not\equiv f \wedge \delta_W(i) \not\equiv f\} \\ \delta_X(i) &\equiv \delta_V(i) \vee_3 \delta_W(i) \end{aligned}$$

The rules for the vague set intersection demonstrate the basic principle for all binary set operations. We consider *all* explicitly known elements, i.e. all $i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon$. This is identical to the proceeding if we only knew that V and W were incomplete. The difference is that we check the desired properties for each element by the help of the descriptive components in a second step.

Another important query language operator is the selection σ. Its adaptation to vague sets maps a vague set and a three-valued selection predicate to a vague result set: $\sigma_3 : \widetilde{\mathbf{Set}}(T) \times (T \to \{t, f, u\}) \to \widetilde{\mathbf{Set}}(T)$. The semantics is as follows:

$$\begin{aligned} \sigma_3(V, P_3) = X, \text{ with } \widehat{X}_\lambda &= \{i \in \widehat{V}_\lambda \mid P_3(i) \equiv t\} \\ \widehat{X}_\upsilon &= \{i \in \widehat{V}_\upsilon \mid P_3(i) \not\equiv f\} \\ \delta_X(i) &= \delta_V(i) \wedge_3 P_3(i) \end{aligned}$$

As an example for a relation on vague sets over T we state the vague set inclusion:

$$(V \subseteq_3 W) \equiv \begin{cases} t \text{ , if } \forall i \in T : (\delta_V(i) \Rightarrow_3 \delta_W(i)) \equiv t \\ f \text{ , if } \exists i \in T : (\delta_V(i) \Rightarrow_3 \delta_W(i)) \equiv f \\ u \text{ , otherwise} \end{cases}$$

5 Vague Multisets

Analogously to vague sets, a vague *multi*set is a set of *multi*sets where each element is a multiset which could be the correct answer to the computed query. Applying the base idea underlying vague *sets* – using a hybrid representation which comprises an explicit, enumerated part and a descriptive part – to *multisets*, we get

1. A set $\Delta_\mathcal{V} \subseteq T$, that comprises those potential elements of the multiset $\mathcal{V}$ we actually have under access,
2. a mapping $\lambda_\mathcal{V} : T \to \boldsymbol{N}_0$ that maps each element i of the foundation set to its minimum number of occurrences in $\mathcal{V}$, and
3. a mapping $\upsilon_\mathcal{V} : T \to \boldsymbol{N}_0^\infty$ that maps each element i of the foundation set to its maximum number of occurrences in $\mathcal{V}$; the notation $\boldsymbol{N}_0^\infty$ means the set $\boldsymbol{N}_0 \cup \{\infty\}$, i.e. the extension of $\boldsymbol{N}_0$ by a maximum element to a complete lattice.

Formally, a vague multiset can be defined as follows:

Definition 2 (vague multisets). *A vague multiset over the foundation set T ($\mathcal{V} \in \widetilde{\mathbf{Bag}}(T)$) is a triple $\mathcal{V} = (\Delta_\mathcal{V}, \lambda_\mathcal{V}, \upsilon_\mathcal{V})$, with the components $\Delta_\mathcal{V} \subseteq T$, $\lambda_\mathcal{V} : T \to \boldsymbol{N}_0$, $\upsilon_\mathcal{V} : T \to \boldsymbol{N}_0^\infty$, iff the following conditions hold:*

$$\forall i \in T : \lambda_\mathcal{V}(i) \leq \upsilon_\mathcal{V}(i) \quad \wedge \quad \forall i \in \Delta_\mathcal{V} : \upsilon_\mathcal{V}(i) > 0$$

The relation $\in$ between a multiset $\mathcal{A} \in \mathbf{Bag}(T)$ and a vague multiset $\mathcal{V} \in \widetilde{\mathbf{Bag}}(T)$ is defined as follows: $\mathcal{A} \in \mathcal{V} \Leftrightarrow \forall i \in T : \lambda_\mathcal{V}(i) \leq \mathcal{A}(i) \leq \upsilon_\mathcal{V}(i)$ □

The above definition ensures that we do not carry elements in $\Delta_\mathcal{V}$ which are surely not contained in the vague multiset $\mathcal{V}$.

Let us consider a simple example that shows the use of our representation:

Example 1. Assume the query: *Select the types of the parameters of all methods of all classes of the diagram called "Billing"!*, where the result should contain duplicates.

A natural way to evaluate this query would be to start from the diagram d_1 and to navigate via the appropriate links. In this case the component $\Delta_\mathcal{V}$ of the resulting vague multiset $\mathcal{V}$ would be $\Delta_\mathcal{V} = \{natural, float\}$, because we would only reach p_1, p_2, and p_3.

Let O denote the set containing all objects in the database. Further assume that the function $val(o, a)$ yields the value of the attribute a for the object o and that the three-valued logical predicate $pred_3 : O \to \{t, f, u\}$ requires (1) that a candidate must be of type *Parameter*, (2) that it must have an incoming link of type *has_parameter* with origin m, (3) that m must have an incoming link of type *has_method* with origin c, (4) that c must have an incoming link of type *contains* with origin d, and (5) that the value of the attribute *name* of d must be "Billing". Obviously each sub-predicate of $pred_3$ must be evaluated in a three-valued manner. Then we can define the functions $\lambda_\mathcal{V}$ and $\upsilon_\mathcal{V}$ as follows:

$$\lambda_\mathcal{V}(i) = \begin{cases} 1 \text{ , if } i = float \vee (\exists_3 p \in O : i = val(p, type) \wedge pred_3(p)) \equiv t \\ 2 \text{ , if } i = natural \\ 0 \text{ , otherwise} \end{cases}$$

$$\upsilon_\mathcal{V}(i) = \begin{cases} 0 \text{ , if } (\exists_3 p \in O : i = val(p, type) \wedge pred_3(p)) \equiv f \\ \infty \text{ , otherwise} \end{cases}$$

This definition takes into account that (1) two occurrences of *natural* and one occurrence of *float* are reached, and that (2) we know that there might be missing elements.

Now we can show the adaptation of two query language operators to vague multisets. First we consider the vague multiset intersection $\cap_{m,3}$ as an example for a typical base multiset operation [1]. To compute $\mathcal{X} = \mathcal{V} \cap_{m,3} \mathcal{W}$, we define:

$$\Delta_\mathcal{X} = \{i \in \Delta_\mathcal{V} \cup_m \Delta_\mathcal{W} \mid \upsilon_\mathcal{X}(i) > 0\}$$
$$\lambda_\mathcal{X}(i) = \min(\lambda_\mathcal{V}(i), \lambda_\mathcal{W}(i))$$
$$\upsilon_\mathcal{X}(i) = \min(\upsilon_\mathcal{V}(i), \upsilon_\mathcal{W}(i))$$

[1] Note that we use the index m to distinguish set and multiset operations.

The other operator we consider is the selection on multisets σ_m. Its adaptation to vague multisets maps a vague multiset and a three-valued selection predicate to a vague result multiset: $\sigma_{m,3} : \widetilde{\mathbf{Bag}}(T) \times (T \to \{t, f, u\}) \to \widetilde{\mathbf{Bag}}(T)$ The semantics is as follows:

$$\sigma_{m,3}(\mathcal{V}, P_3) = \mathcal{X}, \text{ with } \Delta_\mathcal{X} = \{i \in \Delta_\mathcal{V} \mid P_3(i) \not\equiv f\}$$

$$\lambda_\mathcal{X}(i) = \begin{cases} \lambda_\mathcal{V}(i) \text{ , if } P_3(i) \equiv t \\ 0 \quad \text{, otherwise} \end{cases}$$

$$\upsilon_\mathcal{X}(i) = \begin{cases} \upsilon_\mathcal{V}(i) \text{ , if } P_3(i) \not\equiv f \\ 0 \quad \text{, otherwise} \end{cases}$$

As an example for a relation on vague multisets over T we state the vague multiset inclusion:

$$(\mathcal{V} \subseteq_{m,3} \mathcal{W}) \equiv \begin{cases} t \text{ , if } \forall i \in T : \upsilon_\mathcal{V}(i) \leq \lambda_\mathcal{W}(i) \\ f \text{ , if } \exists i \in T : \lambda_\mathcal{V}(i) > \upsilon_\mathcal{W}(i) \\ u \text{ , otherwise} \end{cases}$$

6 Lists

Before we actually deal with *vague* lists, we explain our understanding of an *exact* list in the context of query languages.

6.1 Exact Lists

In the context of query languages, a list over the foundation set T is usually envisaged as a mapping $N \to T$. In our opinion this is not exactly what we need, because in most cases a list arises from sorting a (multi)set. The sorting criteria actually used to order a multiset (1) build a partition of the multiset where each subset (= rank) consists of elements with the same value for the sorting criterion, and (2) impose an irreflexive order upon these ranks.

To become more precise, we give an example: Assume a set of employees which is to be sorted with respect to their ages. There may be different employees with the same age. If we want to represent the result list of the sorting operation as a mapping $N \to$ **employee**, we have to impose an additional order on the employees of the same age. By doing so, we lose a certain degree of freedom, that we conceptually could have. Of course, the representation of a query result usually assumes the additional order, too, but its early introduction during query processing is not necessary.

Another approach would be to represent a list by a multiset together with an irreflexive order over the different elements. But unfortunately — e.g. due to list concatenation — one and the same element of the foundation set can happen to occur at *disconnected* places in a list. To handle these cases, we identify each separated cluster of the same element i by a unique number n, and we represent an exact list by a multiset where the individual pairs (i, n) constitute the domain, together with an irreflexive order over these pairs.

Definition 3 (exact lists). *An exact list* $\mathtt{A}$ *over the foundation set* T *(*$\mathtt{A} \in \mathbf{List}(T)$*) is a pair* $(o_{\mathtt{A}}, <_{\mathtt{A}})$*, with* $o_{\mathtt{A}} : T \times \mathbb{N} \to \mathbb{N}_0$*,* $<_{\mathtt{A}} \subseteq (T \times \mathbb{N}) \times (T \times \mathbb{N})$*. The components are defined as follows:*

1. $o_{\mathtt{A}}$ *is a multiset over* $(T \times \mathbb{N})$*, i.e.* $o_{\mathtt{A}} \in \mathbf{Bag}(T \times \mathbb{N})$*.*
2. $<_{\mathtt{A}}$ *is an irreflexive order over the relevant part* $\rho(\mathtt{A}) = \{(i,n) \in (T \times \mathbb{N}) \mid o_{\mathtt{A}}(i,n) > 0\}$ *of* $\mathtt{A}$*, i.e. it is (a) irreflexive:* $\forall x \in \rho(\mathtt{A}) : x \not<_{\mathtt{A}} x$*, (b) transitive:* $\forall x,y,z \in \rho(\mathtt{A}) : x <_{\mathtt{A}} y \wedge y <_{\mathtt{A}} z \Rightarrow x <_{\mathtt{A}} z$*, and (c) trichotome:* $\forall x,y \in \rho(\mathtt{A})$ *: exactly one of the relations* $x <_{\mathtt{A}} y$*,* $y <_{\mathtt{A}} x$*, or* $x =_{\mathtt{A}} y$ *is true (the equality '*$=_{\mathtt{A}}$*' is interpreted as the equivalence of two elements relative to* $<_{\mathtt{A}}$*).* □

Example 2. Assume the query: *Calculate a list of all methods of all classes in the diagram named "Billing" which have at least one natural parameter, sorted with respect to the number of their natural parameters!*, evaluated on the database in figure 1, but *segment 3* **being accessible.**
The resulting list $\mathtt{A}$ could be given by

$$o_{\mathtt{A}}(i) = \begin{cases} 1 \text{ , if } i \in \{(m_1,1),(m_2,1),(m_6,1)\} \\ 0 \text{ , otherwise} \end{cases}$$
$$<_{\mathtt{A}} \; : \; (m_1,1) =_{\mathtt{A}} (m_2,1) <_{\mathtt{A}} (m_6,1)$$

There are two points to be mentioned with respect to this example: (1) Because the list arises from a set instead of a multiset, both the mapping $o_{\mathtt{A}}$ and the numbering of the individual "clusters" of the same element (that have cardinality 1) are actually unnecessary. (2) We numbered the elements of $\rho(\mathtt{A})$ only for reasons of simplicity by 1. The numbering can be arbitrary.

From a mathematical point of view we first build the partition of $\rho(\mathtt{A})$ and then we impose an order on this partition. Nevertheless — from a technical point of view — both steps can be computed simultaneously. To this end, we evaluate the predicate $x <_{\mathtt{A}} y$ as well as $y <_{\mathtt{A}} x$. This implicitly yields the partition of $\rho(\mathtt{A})$, because those pairs x, y for which neither $x <_{\mathtt{A}} y$ nor $y <_{\mathtt{A}} x$ holds belong to the same rank.

6.2 Vague Lists

If we take inaccessibility into account, not only the elements of a list may become uncertain, but also the ordering may become vague, e.g. because the complete evaluation of the order criterion would require access to inaccessible parts of the database. When we evaluate the ordering $<_3$ three-valued, the results given in table 1 are possible [2]. The missing combinations are not possible due to the trichotomy of an irreflexive order.

Definition 4 (vague order). *The three-valued logical predicate* $<_3 : U \times U \to \{t,f,u\}$ *is a vague order over* U*, iff*

[2] The question mark means that we have no information about the relation between x and y.

$x <_3 y$	f	f	f	u	u	t
$y <_3 x$	f	u	t	f	u	f
relation between x and y	$=$	$\geq$	$>$	$\leq$	?	$<$

Table 1. Relation between x and y for $<_3$

1. $\forall x \in U : (x <_3 x) \equiv f$ *(irreflexivity)*
2. $\forall x, y \in U : (x <_3 y) \equiv t \Rightarrow (y <_3 x) \equiv f$ *(trichotomy)*
3. $\forall x, y, z \in U : (y <_3 x) \equiv f \wedge (z <_3 y) \equiv f$
   $\Rightarrow (z <_3 x) \equiv f \wedge (x <_3 z) \equiv (x <_3 y) \vee_3 (y <_3 z)$ *(transitivity)*

An exact order $<$ over U is said to be contained in $<_3$ ($<\in<_3$), iff $\forall x, y \in U : x < y \Rightarrow (x <_3 y) \not\equiv f$. □

If we interpret condition 3 (transitivity) by means of table 1, it can be read as: $\forall x, y, z \in T : x \leq y \wedge y \leq z \Rightarrow x \leq z \wedge (x < z \Leftrightarrow x < y \vee y < z)$

Now that we have defined a vague order, we can also define a vague list $\mathtt{V} \in \widetilde{\mathbf{List}}(T)$ as a quadruple consisting of three components that constitute a vague multiset over $T \times \boldsymbol{N}$ (as mentioned before, the bundling of an element of T with a number serves for the numbering of the individual clusters of elements of T), and one component that builds a vague order over the vague multiset.

Definition 5 (vague lists). *A vague list over the foundation set T ($V \in \widetilde{\mathbf{List}}(T)$) is a quadruple $\mathtt{V} = (\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}, \prec_\mathtt{V})$, with the components $\Delta_\mathtt{V} \subseteq T \times \boldsymbol{N}$, $\lambda_\mathtt{V} : T \times \boldsymbol{N} \to \boldsymbol{N}_0$, $\upsilon_\mathtt{V} : T \times \boldsymbol{N} \to \boldsymbol{N}_0^\infty$, $\prec_\mathtt{V}$: $(T \times \boldsymbol{N}) \times (T \times \boldsymbol{N}) \to \{t, f, u\}$, iff the following conditions hold:*

1. $(\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}) \in \widetilde{\mathbf{Bag}}(T \times \boldsymbol{N})$.
2. *$\prec_\mathtt{V}$ is a vague order over the relevant part* $\rho(\mathtt{V}) = \{(i, n) \in (T \times \boldsymbol{N}) \mid \upsilon_\mathtt{V}((i, n)) > 0\}$ *of* $\mathtt{V}$.

The relation $\in$ between a list $\mathtt{A} \in \mathbf{List}(T)$ and a vague list $\mathtt{V} \in \widetilde{\mathbf{List}}(T)$ is defined as: $\mathtt{A} \in \mathtt{V} \Leftrightarrow (o_\mathtt{A} \in (\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}) \wedge <_\mathtt{A} \in \prec_\mathtt{V})$. □

Example 3. On the database presented in figure 1 we evaluate the following query: "*Calculate a list of all methods of all classes in the diagram named "Billing" which have at least one float parameter, sorted with respect to the number of their float parameters!*"

To state the vague result list of this query we use (1) an auxiliary predicate $pred_3 : O \to \{t, f, u\}$ which requires that a candidate must be a method of a class of the diagram named "Billing" having at least one float parameter, and (2) the function $func_3 : O \to \mathcal{P}(\boldsymbol{N}_0)$ which determines the vague number of float parameters of a method. Then the vague result list can be given as:

$$\Delta_{\mathtt{V}} = \{(m_1,1),(m_5,1),(m_6,1)\}$$

$$\lambda_{\mathtt{V}}((i,n)) = \begin{cases} 1 \text{ , if } (i,n) = (m_1,1) \vee pred_3(i) \equiv t \wedge n = 1 \\ 0 \text{ , otherwise} \end{cases}$$

$$\upsilon_{\mathtt{V}}((i,n)) = \begin{cases} 0 \text{ , if } pred_3(i) \equiv f \vee n = 0 \\ 1 \text{ , otherwise} \end{cases}$$

$$\prec_{\mathtt{V}}((i,n),(j,m)) = \begin{cases} f \text{ , if } n = m = 1 \wedge (i,j) = (m_1,m_5) \\ u \text{ , if } n = m = 1 \wedge (i,j) \in \{(m_1,m_6), \\ \qquad (m_5,m_6),(m_5,m_1),(m_6,m_1),(m_6,m_5)\} \\ func_3(i) <_3 func_3(j), \text{otherwise} \end{cases}$$

6.3 Vague List Operations

Now we explain the adaptation of list concatenation and selection to vague lists.

Whenever we combine two lists V and W, we always presuppose (due to technical reasons) that the relevant elements $i \in \rho(\mathtt{V})$ and $j \in \rho(\mathtt{W})$ are numbered differently, i.e. an element (i,n) is never contained both in $\rho(\mathtt{V})$ and in $\rho(\mathtt{W})$. This premise can easily be achieved by renumbering the corresponding elements.

As an example for a base operation on vague lists we show the adaptation of the list concatenation $\circ$ to vague lists. In order to compute $\mathtt{X} = \mathtt{V} \circ_3 \mathtt{W}$, we define:

$$\Delta_{\mathtt{X}} = \Delta_{\mathtt{V}} \cup \Delta_{\mathtt{W}}$$

$$\lambda_{\mathtt{X}}(k) = \begin{cases} \lambda_{\mathtt{V}}(k) \text{ , if } k \in \rho(\mathtt{V}) \\ \lambda_{\mathtt{W}}(k) \text{ , if } k \in \rho(\mathtt{W}) \\ 0 \text{ , otherwise} \end{cases}$$

$$\upsilon_{\mathtt{X}}(k) = \begin{cases} \upsilon_{\mathtt{V}}(k) \text{ , if } k \in \rho(\mathtt{V}) \\ \upsilon_{\mathtt{W}}(k) \text{ , if } k \in \rho(\mathtt{W}) \\ 0 \text{ , otherwise} \end{cases}$$

$$\prec_{\mathtt{X}}(k,l) \equiv \begin{cases} \prec_{\mathtt{V}}(k,l) \text{ , if } k \in \rho(\mathtt{V}) \wedge l \in \rho(\mathtt{V}) \\ \prec_{\mathtt{W}}(k,l) \text{ , if } k \in \rho(\mathtt{W}) \wedge l \in \rho(\mathtt{W}) \\ t \text{ , if } k \in \rho(\mathtt{V}) \wedge l \in \rho(\mathtt{W}) \\ f \text{ , otherwise} \end{cases}$$

The adaptation of the selection on lists σ_l to vague lists is a mapping, that expects a vague list and a three-valued logical predicate as input, and that delivers a vague list as result: $\sigma_{l,3} : \widetilde{\mathbf{List}}(T) \times (T \to \{t,f,u\}) \to \widetilde{\mathbf{List}}(T)$

The semantics is as follows:

$$\sigma_{l,3}(\mathtt{V},P_3) = \mathtt{X}, \text{ with } \Delta_{\mathtt{X}} = \{(i,n) \in \Delta_{\mathtt{V}} \mid P_3(i) \not\equiv f\}$$

$$\lambda_{\mathtt{X}}(k) = \begin{cases} \lambda_{\mathtt{V}}(k) \text{ , if } P_3(i) \equiv t \\ 0 \text{ , otherwise} \end{cases}$$

$$\upsilon_{\mathtt{X}}(k) = \begin{cases} \upsilon_{\mathtt{V}}(k) \text{ , if } P_3(i) \not\equiv f \\ 0 \text{ , otherwise} \end{cases}$$

$$\prec_{\mathtt{X}}(k,l) \equiv \prec_{\mathtt{V}}(k,l)$$

7 Aggregate Functions

Another important type of operations is aggregate functions. Due to space limitations we restrict ourselves to aggregate functions on vague sets here, but the presented techniques can be adapted to vague multisets and vague lists as well.

An aggregate function is a mapping $\alpha : \mathcal{P}(T_1) \rightarrow T_2$, where T_2 is *not* a collection-valued type. This comprises e.g. the maximum, the minimum, the sum, the cardinality, or the arithmetic average of a set.

Taking inaccessibility into account – and thus assuming a *vague* input set instead of a crisp one – we get a vague aggregate function $\alpha_3 : \widetilde{\mathbf{Set}}(T_1) \rightarrow \mathcal{P}(T_2)$. In this case we have to compute a whole set of possible aggregate values each of which could be the correct aggregate value. Formally spoken, $\alpha_3(V)$ is defined as $\alpha_3(V) = \{\alpha(A) \mid A \in V\}$.

If a vague set V is not incomplete, $\alpha_3(V)$ can be determined by the computation of $\alpha(A)$ for each $A \in V$. Because the cardinality of V grows exponentially with the number of uncertain elements in V (namely $|V| = |\mathcal{P}(\widehat{V}_\upsilon \setminus \widehat{V}_\lambda)|$) also the cost for the computation of $\alpha_3(V)$ grows exponentially.

To avoid this effort whenever possible, we identify two classes of aggregate functions for which the vague adaptation can be computed more efficiently. We differentiate set-monotone and element-monotone aggregate functions.

7.1 Set-Monotone Aggregate Functions

There are set-monotone *increasing* and set-monotone *decreasing* aggregate functions. They behave completely analogous.

A set-monotone increasing (decreasing) aggregate function $\alpha : \mathcal{P}(T_1) \rightarrow T_2$ is an aggregate function for which its value does not become lower (higher) when adding elements to the input set: $\forall A, B \in \mathbf{Set}(T_1) : A \subseteq B \Rightarrow \alpha(A) \leq \alpha(B)$.

Examples for set-monotone increasing (decreasing) aggregate functions are the sum of some positive numbers or the maximum (minimum) of a set.

Due to the above definition, the condition $\forall A \in V : \alpha(V_\lambda) \leq \alpha(A) \leq \alpha(V_\upsilon)$ holds for a vague set V and a set-monotone increasing aggregate function α.

By means of this condition, an estimation for the vague aggregate function α_3 on a *complete* vague set V can be computed very efficiently:

$$\alpha_3(V) \subseteq [\alpha(\widehat{V}_\lambda); \alpha(\widehat{V}_\upsilon)]$$

In general this interval also contains values i, for which there is no $A \in V$ with $\alpha(A) = i$. But in most cases this decrease in accuracy will be over-compensated by the increased performance. An example for an application where the decrease in accuracy does not matter at all, might arise if we evaluate a predicate checking whether the maximum of a set is higher than a given value or not.

If the vague set V is *not complete*, we can nevertheless use $\alpha(\widehat{V}_\lambda)$ as a lower (upper) bound for the value of $\alpha_3(V)$ of a set-monotone increasing (decreasing) aggregate function α_3. Unfortunately, the upper (lower) bound for $\alpha_3(V)$ has to be set to the lowest upper bound (greatest lower bound) of the set T_2 in this case.

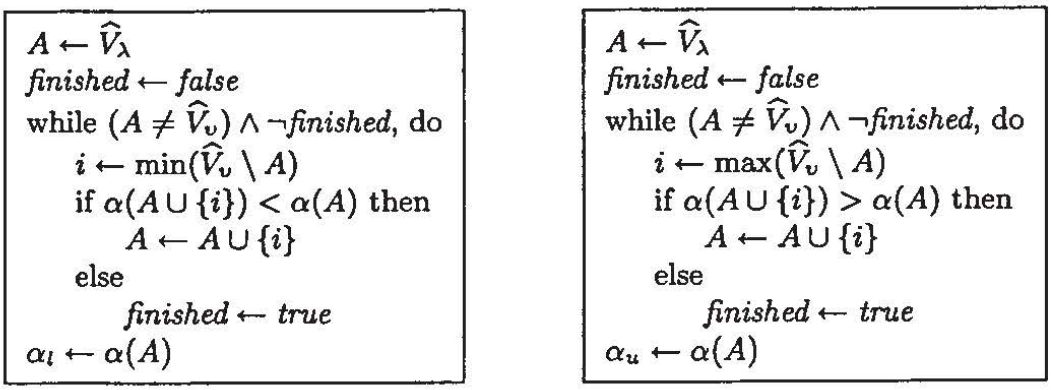

Fig. 2. Algorithms to calculate bounds for the vague version of an aggregate function

7.2 Element-Monotone Aggregate Functions

The second class of aggregate functions for which we propose an efficient algorithm for the computation of (useful) lower and upper bounds is element-monotone aggregate functions. Similarly to set-monotone aggregate functions, there are element-monotone increasing and element-monotone decreasing aggregate functions. Again both subclasses behave analogous.

An element-monotone increasing aggregate function is an aggregate function $\alpha : \mathcal{P}(T_1) \rightarrow T_2$ for which the insertion of a smaller element into the input set yields no higher aggregate value than the insertion of a greater element. Formally: $\forall A \in \mathcal{P}(T_1); i, j \in T_1 \setminus A : i \leq j \Rightarrow \alpha(A \cup \{i\}) \leq \alpha(A \cup \{j\})$

A popular example for an element-monotone increasing aggregate function is the arithmetic average of a set of numbers.

Now we can describe the algorithms for the computation of the vague adaption of an element-monotone increasing aggregate function.

For a complete vague set V a lower bound α_l and an upper bound α_u for the vague adaptation of an element-monotone increasing aggregate function $\alpha_3(V)$ can be computed by the algorithms given in figure 2.

Both algorithms have an asymptotic execution time of $O(n^2)$ in the worst case where n is the number of uncertain elements in V, i.e. $n = |\widehat{V}_\upsilon \setminus \widehat{V}_\lambda|$. Fortunately the asymptotic execution time of the algorithms can be reduced to $O(n \log n)$ by previously sorting $\widehat{V}_\upsilon$ (at a cost of $O(n \log n)$).

Unfortunately, the described algorithms are only applicable if the vague set V is indeed complete. Otherwise special techniques exploiting specific characteristics of the concrete aggregate function α and the information contained in the descriptive component of the vague input set have to be applied.

References

1. S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *Proc. ACM SIGMOD 1987 Int. Conf. on Management of Data*, pages 34–48, San Franzisco, Cal., USA, 1987.

2. L. Amsaleg, M.J. Franklin, A. Thomasic, and T. Urhan. Scrambling Query Plans to Cope With Unexpected Delays. In *Proc. 4th Int. Conf. on Parallel and Distributed Information Systems (PDIS96)*, Miami Beach, Florida, Dec 1996.
3. K. Basu, R. Deb, and P. K. Pattanaik. Soft sets: An ordinal formulation of vagueness with some applications to the theory of choice. *Fuzzy Sets and Systems*, 45:45–58, 1992.
4. P. Bonnet and A. Thomasic. Partial Answers to Unavailable Data Sources, 1997. INRIA Rocquencourt, Rapport de Recherche 3127.
5. E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.
6. European Computer Manufacturers Association, Geneva. *Portable Common Tool Environment - Abstract Specification (Standard ECMA-149)*, 1993.
7. W. L. Gau and D. J. Buehrer. Vague sets. *IEEE Transactions on Systems, Man and Cybernetics*, 23(2):610–614, 1993.
8. G.H. Gessert. Handling missing data by using stored truth values. *ACM SIGMOD Record*, 20(3):30–42, 1991.
9. O. Haase and A. Henrich. Error propagation in distributed databases. In *Proc. 4th Int. Conf. on Information and Knowledge Management (CIKM'95)*, pages 387–394, 1995.
10. O. Haase and A. Henrich. A hybrid representation of vague collections for distributed object management systems. accepted for publication in IEEE Transactions on Knowledge and Data Engineering, 1999.
11. A. Heuer and A. Lubinski. Database access in mobile environments. In *Proc. 7th Int. Conf. on Database and Expert Systems Applications, volume 1134 of LNCS*, pages 544–553, Zürich, 1996. Springer.
12. T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal Association for Computing Machinery*, 31(4):761–791, 1984.
13. W. Lipski. On relational algebra with marked nulls. In *Proc. 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 201–203, Waterloo, Canada, 1984.
14. J.M. Morrissey. Imprecise information and uncertainty in information systems. *ACM Transactions On Information Systems*, 8(2):159–180, 1990.
15. A. Motro. Accommodating imprecision in database systems: Issues and solutions. *ACM SIGMOD Record*, 19(4):69–74, 1990.
16. Z. Pawlak. Rough Sets. *International Journal of Computer and Information Sciences*, 11(5):341–356, 1982.
17. A. Thomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, and L. Raschid. The Distributed Information Search Component (Disco) and the World Wide Web. In *Proc. SIGMOD 1997 Conference*, pages 546–548, Tucson, Arizona, May 1997.
18. Y. Vassiliou. Null values in data base managment: A denotational semantics approach. In *Proc. ACM SIGMOD 1979 Int. Conf. on Management of Data*, pages 162–169, Boston, Mass., USA, 1979.
19. E. Wong. A statistical approach to incomplete information in database systems. *ACM Transactions on Database Systems*, 7(3):470–488, 1982.
20. L. Y. Yuan and D.-A. Chiang. A sound and complete query evaluation algorithm for relational databases with disjunctive information. In *Proc. 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 66–74, Philadelphia, Pa., USA, 1989.