VAGUE: A User Interface to Relational Databases that Permits Vague Queries

AMIHAI MOTRO
University of Southern California, Los Angeles

A specific query establishes a rigid qualification and is concerned only with data that match it precisely. A vague query establishes a target qualification and is concerned also with data that are close to this target. Most conventional database systems cannot handle vague queries directly, forcing their users to retry specific queries repeatedly with minor modifications until they match data that are satisfactory. This article describes a system called VAGUE that can handle vague queries directly. The principal concept behind VAGUE is its extension to the relational data model with data metrics, which are definitions of distances between values of the same domain. A problem with implementing data distances is that different users may have different interpretations for the notion of distance. VAGUE incorporates several features that enable it to adapt itself to the individual views and priorities of its users.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*query processing*; H.3.3. [**Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models*

General Terms: Design, Human Factors, Languages

Additional Key Words and Phrases: Approximate match retrieval, database, data metric, neighborhood query, relational database, user interface, vague query

1. INTRODUCTION

Requests for data can be classified roughly into two kinds: *specific* queries and *vague* queries. A specific query establishes a rigid qualification and is concerned only with data that match it precisely. Some examples of specific queries are "How much does Jones earn?" or "When does flight 909 depart?" If the database does not contain salary information on Jones or departure time for flight 909, null answers should be returned; the user is not interested in the earnings of somebody else or in the departure time of a different flight. A vague query, on the other hand, establishes a target qualification and is concerned with data that are *close* to this target. As an example, consider "List the inexpensive French restaurants in Westwood." If there are none, a moderately priced Continental

This work was supported in part by National Science Foundation grant IRI-8609912 and by an Amoco Foundation Engineering Faculty Grant.
Author's address: Computer Science Department, University of Southern California, University Park, Los Angeles, CA 90089-0782.

restaurant in Santa Monica may have to do. Similarly, when a project calls for experienced C programmers with background in applied mathematics, we may want the personnel database to mention also that there is an engineer with some knowledge of Pascal.

Most conventional database systems cannot handle vague queries directly. Consequently, they must be emulated with specific queries. Usually, this means that the user is forced to retry a particular query repeatedly with alternative values until it matches data that are satisfactory. If the user is not aware of any close alternatives, then even this solution is infeasible.

In this article we describe a system called VAGUE that extends the relational data model [3] to provide it with vague retrieval capabilities. An initial scheme for handling vague queries in relational databases was proposed in [17].

1.1 Outline of Approach

To determine similarity between data values we introduce the notion of distance. Each database domain is provided with a definition of distance between its values called *data metric*. For example, in a database on restaurants there may be metrics to measure distances between cuisines, between locations, between price ranges, as well as a metric to measure distances between restaurants.

To express vague queries we introduce a vague selection comparator, called *similar-to*. A *similar-to* comparison is satisfied with data values that are within a predefined distance of the specified value. For example, the vague comparison "location *similar-to* Westwood" may be satisfied by Westwood, Santa Monica, and Beverly Hills.

Thus, the previous specific query "List the restaurants whose cuisine is French, whose price range is inexpensive, and whose location is Downtown" may be relaxed into a vague query such as: "List the restaurants whose cuisine is *similar-to* French, whose price range is *similar-to* inexpensive, and whose location is *similar-to* Downtown."

This model is quite straightforward, and its satisfactory operation relies almost entirely on the quality of the metrics that are provided for the individual domains. Here, VAGUE allows the database designer four choices: using one of several *built-in* metrics; providing a procedure that *computes* the distance between every two elements of the domain; providing a relation that *stores* the distance between every two elements of the domain; or using a *reference* relation (an existing database relation that is keyed on this domain). In the latter case, distances between elements of the domain would be defined as distances between their tuples in the reference relation, where tuple distance is defined as a combination of the individual distances between their corresponding components.

Each tuple in the answer to a query that includes several vague qualifications involves several deviations from the specific values mentioned in these qualifications. By combining these individual deviations into a single value, VAGUE can present the answer to the user in order of optimality.

Thus, there are two occasions when VAGUE combines several component distances into a single distance: in one of its metric types and in the presentation of vague answers. The particular formula VAGUE uses for combining distances is discussed in Section 1.3.3.

1.2 Design Considerations

The design of VAGUE reflects three fundamental requirements:

Conceptual Simplicity Within a Relational Framework. The purpose of VAGUE is to enhance a relational database system with vague retrieval capabilities. The relational data model is adopted primarily because of its widespread popularity, the simplicity of its structures, and the advantages of a formal query language such as the relational calculus. An important design guideline is to realize this goal with only minimal deviation from this popular model. The relational data model is extended with a single concept, *data metrics*, and the query language is extended with a single feature, a *similar-to* comparator. (Indeed, the relational data model is *generalized* since a nonmetricized database is a particular type of a metricized database.) To present queries, users need only to know about the new comparator.

Adaptability. To be useful, a system that implements vague queries must be able to adapt itself to the views and priorities of its individual users. VAGUE incorporates three adaptability features. (1) Often, distances between values of a given domain may be measured according to various metrics. For example, distances between values of domain CITY may be defined in miles "as the crow flies," or as shortest driving distances, or even as differences between the names of the cities. VAGUE permits *multiple metrics for the same domain.* When a query makes use of a *similar-to* comparator, the user is presented with the various possible semantics of this comparator in its present context and is asked to select. (2) With referential metrics, one of the metric types available in VAGUE, individual users are allowed to *influence the definition of the metric* according to their own views. For example, the distance between two cities may be defined as a combination of the distances between some of their available attributes, such as size of population, climate, and employment rate. If such a metric is selected, the user is allowed to judge the relative importance of the various attributes in the overall distance. (3) When a query involves several vague qualifications, users are allowed to *express their relative importance* in the overall query. For example, consider the previous vague query about restaurants whose cuisine is similar to French, whose price range is similar to inexpensive, and whose location is similar to Downtown. Each tuple in its answer involves three deviations from the specified values, which are then combined so that the answer may be presented in order of optimality (i.e., "best" tuple first). However, it may be that the user has a different willingness to compromise on the various qualifications; for example, the user may be willing to compromise more on the type of the restaurant than on its price range or location. VAGUE allows users to express their relative willingness to compromise and uses this input in the definition of the corresponding metric.

DBMS Externality. Ideally, data metrics and the similar-to comparator should be an integral part of the database system. However, it is also possible to enhance existing database systems by implementing these capabilities "on top" of these systems, an advantage when the database system cannot be modified. Since all distance information is represented as auxiliary relations and computations, it is relatively easy to maintain (e.g., add new metrics or modify existing metrics).

1.3 Related Research

1.3.1 *ARES.* A system that resembles VAGUE in its overall goals is ARES [8]. The designers of ARES extended a Query-by-Example (QBE) [29] interface with a similar-to operator. QBE queries are then translated into extended relational algebra operations called *ambiguous select, ambiguous project,* and *ambiguous join.* These, in turn, are translated into conventional relational algebra operations. ARES addresses the basic issue of "similarity matching" and it includes several useful graphics-based tools. However, it has several basic flaws, mainly: (1) Distances can only be defined via *tables.* This precludes defining distances for infinite domains, such as the distance between any two given strings (VAGUE has additional types of metrics such as *computational* and *referential*). (2) ARES lacks any of the adaptability features described above; for example, it does not allow multiple metrics for the same domain, and it cannot adapt itself to the views and priorities of individual users. Finally, (3) the extensions of the relational data model appear to be unnecessarily complex. Altogether, although ARES is a useful system that can benefit its users, these flaws substantially limit its usefulness.

1.3.2 *Fuzzy Databases.* The issue of vague retrieval has also been addressed in the context of *fuzzy systems.*

Buckles and Petri [2] assume that each database domain has an associated *similarity matrix* that assigns a value between 0 and 1 with each pair of domain elements. Relations are extended to allow values that are *sets* of domain elements. The output of standard relational algebra operators, such as join or project, is postprocessed to merge tuples (by performing unions of their respective components) if a prespecified similarity threshold is not violated.

Other researchers [19, 28] define fuzzy databases that adhere more closely to the theory of fuzzy sets and systems. The main feature of a fuzzy database is that it can store imprecise information of various kinds; for example, Tom's residence is either in Boston or in New York, Mary's hair color is .6 brown and .4 black, John is .9 smart, Betty's age is young. Definitions of fuzzy attribute values (such as "young") and fuzzy comparators (such as "much-greater-than") are provided by the system to allow users to formulate vague queries.

In comparison, our approach here assumes conventional databases that store only *precise* information, and our solutions are intended to be used in conjunction with conventional database technology: Existing databases may be metricized by providing the necessary distance information (this can even be done gradually over a period of time), and existing database systems can be extended (either internally or externally) to handle vague queries.

1.3.3 *Information Retrieval Systems.* The notion of *proximity* of information items, such as documents, has been investigated in conjunction with *information retrieval systems* [21, 25]. The most basic model describes each document with a set of relevant *terms.* This description is then converted to a *vector* over the space of all possible terms: If a term appears in the description of an item, then the corresponding position in its vector is set to 1; otherwise it is set to 0. A retrieval request specifies a set of applicable terms and is converted to a vector

over the same space. The request is then satisfied by all the documents whose vectors are "similar" to the vector of the request.

A standard refinement of this basic model is to use *weighted terms*. The vector position that corresponds to a particular term denotes the *degree of relevance* of this term to the particular document. These *weights* may be assigned manually, or they may be derived automatically from frequency counts of the terms in the document. The weight of a particular term in a particular vector expresses the *importance* of this term in comparison with the other terms that describe this document, as well as the *relevance* of this term to the document in comparison with other documents. A retrieval request describes the "ideal" document by means of a similar vector of weighted terms (these terms may also be viewed as a specification of the relative importance of the terms in the overall request).

Numerous measures have been suggested for the critical problem of determining the similarity of two such vectors, most notably *inner product*, *Dice*, *Jaccard*, *cosine*, and *overlap*.[1] The inner product is defined as the sum of the products of the corresponding components of the two vectors. When the components are binary, the inner product counts the matching terms. The other four measures may all be considered as normalized versions of the inner product [25, pp. 38–42]. The relative effectiveness of the various similarity measures has been studied both empirically and analytically. However, although these studies may show that various measures have individual characteristics, they are often inconclusive [10].

As mentioned earlier, there are two occasions when VAGUE must deal with vector distances: in referential metrics, where the distance between two tuples defines the distance between their corresponding key values, and in the answer space, where the distance between answer tuples and the "ideal" tuple is used for ranking the answer. Thus, while in information retrieval systems the components of vectors are *indications* of whether particular terms apply to the document (or, with weighted terms, of how strongly they apply), here they are arbitrary database *values*. Consequently, the previous measures, which were designed for comparing vectors of indicators, are not applicable to vectors of values. Instead, VAGUE applies individual component metrics to derive from the two vectors a vector of their component distances. Then, it weights the individual distances (this is similar to using weighted terms). Finally, it maps the vector into a single distance by computing its *Euclidean length*.

VAGUE's approach is to provide an environment for defining any desirable metric. Thus, various metrics can be defined for the database domains, and users may select the appropriate metric in a brief dialogue that follows the submission of a vague query. In accordance with this approach, it would appear desirable to allow users to select from different formulas for mapping the vector of distances into a single distance in the two aforementioned occasions (currently users can only provide the weights). However, although casual users can be expected to select intelligently among various domain metrics (their semantics may be described in plain language), the same cannot be assumed for the selection of formulas for mapping vectors of distances. The database designer should experiment with the various formulas and select the formula that performs most

[1] For example, the SMART information retrieval system [20] often uses the cosine measure.

satisfactorily. In the case of referential metrics, a particular formula may be associated with each referential metric. However, in the case of answer spaces, it is not feasible to associate a formula with each answer space (for a database with n attributes there are 2^n possible answer spaces). A possible compromise here is to allow the database designer to select one formula for the entire database.[2] Multiple mapping formulas are not yet available in the current version of VAGUE.

A language approach to best-match querying is advocated by [27]. The authors argue that the notion of closeness is extremely user dependent, and it is therefore infeasible to adopt predefined concepts of closeness. Instead, they suggest language features that allow users to specify their retrieval priorities. For example, when either of several values may be satisfactory, a special operator will enable the user to specify an order of preference (this should be preferred to the standard disjunction operator). Although most of the suggested language features appear useful, it should be noted that this solution does not address directly the issue of vague retrieval; for example, what if the user can just name a single value, and the system cannot match it? Although we concur with the opinion that the interpretation of proximity is individual, VAGUE demonstrates that the problem may be addressed in part by allowing multiple interpretations (some of them even parameterized) and determining the suitable interpretation via a brief dialogue with the user.

1.3.4 *Query Constructors, Browsers, and Cooperative Interfaces.* In our model, vague queries are distinct from specific queries only by their "soft" selection qualification. Thus, the same level of expertise is required to issue specific or vague queries. Another kind of vague request occurs when the user does not possess the knowledge required for formulating a proper query (this may be because the user is not familiar with the data model, the query language, the organization of the particular database, or because the user does not have a well-defined retrieval goal). This problem has been approached in two ways. *Interactive query constructors* help users crystalize their requests. A notable example is RABBIT [26], which applies a paradigm of repetitive reformulation of an initial goal. At each iteration in the construction process the user is presented with the answer to the current query. Having observed the answer, the user can then refine the query by critiquing it in one of several ways available. *Browsers*, such as TIMBER [22], SDMS [7], BAROQUE [15] or KIVIEW [18], provide users with a variety of features for exploratory searches. Often, the information is represented as a network, and the retrieval process is iterative. At each iteration the user is presented with information that corresponds to the current location on the network. The user can then issue a new command to advance the search in a particular direction. Elements of browsers are also present in the ME system [9]. The ME database is a network of files connected through links which represent weighted terms. A retrieval request is a set of terms, and a spreading activation process is used to match the files that are most relevant. As the user changes the terms of the query in one terminal window, the window that shows the matched files is updated dynamically.

[2] Note that the formula does not determine the tuples in the answer, only their ranking.

One obvious way to relax a selection qualification is to *delete* one of its conjuncts.[3] Although this method (which is equivalent to using a metric with an infinite radius) is cruder than relaxing that conjunct with a vague qualification, it can be used with standard (nonmetricized) databases. The effect of deleting a conjunct is to create a *more general* query (a query whose answer will contain the answer to the original query). This method was employed in various mechanisms for detecting *erroneous presuppositions* in user queries and for generating *cooperative responses* [4, 11, 16].

A user interface to databases that is capable of handling vague requests appears to be more "intelligent." This is because answering questions with information that is only close to what was requested, or somehow related to it, is a common feature of human interaction. Such interaction is known as *cooperative behavior*, and there has been much focus on how to improve man–machine interaction by emulating such behavior through various techniques. Various cooperative interfaces (including those mentioned above) are discussed in [1]. Not surprisingly, this added intelligence is made possible by including additional semantic information in the database, namely distances.

1.3.5 *Database Semantics.* Data metrics express important semantic information about the domains. This information permits certain manipulations of the data (such as safe substitutions of some values by similar values), and it can be used as evidence that two differently named attributes are similar. Thus, the concept of metricized domains is in the same general class as *abstract domains* [14] and *ordered domains* [6].

We have chosen to represent the semantics necessary to process vague queries (i.e., similarity information) through the mechanism of metrics and to retain the standard relational data model. An alternative approach would be to define a data model with structures rich enough to store these semantics; for example, some variant of a semantic network with a new type of link between its objects that would specify their proximity.

1.3.6 *Null Values.* Sometimes, the value that should be stored in a particular position in the database is not known, but it is clear that some appropriate value does exist. The prevailing approach in such situations is to fill these "vacancies" with *null* values [13, chap. 12].

If nulls are treated as regular values, then distances may involve them as well. We use the term *located null* for null values for which some distance information is available. The advantage of located nulls is that they participate in retrieval. Consider, for example, a restaurant whose exact type is not available but is known to serve food which is similar to French. If the null value that represents the type of the restaurant is located within the neighborhood of French, then every query about restaurants whose type is close to French will retrieve this particular restaurant. Thus, with data metrics, conventional database systems can be extended in two important ways: They can process *requests* that are less specific, and they can store *information* that is less specific.[4]

[3] One may also consider *adding* a disjunct, but this is much less straightforward.

[4] In this respect, null values are like fuzzy values.

1.4 Overview of This Article

The remainder of this article is organized as follows. Section 2 provides a formal definition of the model. The concepts of data metric and metricized databases are defined, various types of metrics are introduced, a detailed example is described, and issues of metric selection are discussed. Section 3 shows how data metrics are applied in retrieval. A formal definition of vague queries is given, QUEL, the query language of choice in this project, is extended to express vague queries, and the ranking of answers to vague queries is defined. Section 4 is devoted to the prototype implementation. The various components of the VAGUE system are described, and issues of update and performance are discussed. Section 5 is devoted to issues of incomplete database information in the presence of data metrics. Two kinds of information are discussed: missing data values and missing data distances. Section 6 concludes with a brief summary.

2. THE MODEL

In this section we define data metrics and metricized databases; we describe different methods for specifying metrics; we give a detailed example of a metricized database; and we provide guidelines for selecting appropriate metrics for the domains of a given database.

2.1 Metricized Databases

We assume the standard definition of relational databases. In particular, we distinguish between *attributes* and *domains*. An attribute is a named column in a relation. A domain is a set of values (possibly infinite). Each attribute is associated with one domain. The domain contains all the values that may appear in that attribute. In practice, a domain is either defined *abstractly* (e.g., a number in the range 1–100, an alphabetic string of at most 24 characters), or *implicitly* (i.e., it comprises the values currently stored in the database attributes that are associated with it), or there is a *reference* relation, whose key attribute defines all the values of the domain (e.g., relation COUNTRY whose key is COUNTRY_NAME serves as reference for attribute CITIZENSHIP in relation PERSON).

Let D be a database domain. A *data metric* for D is a function $M: D \times D \rightarrow R$ such that $\forall x, y \in D$:

(1) $M(x, y) \geq 0$
(2) $M(x, y) = 0$ iff $x = y$
(3) $M(x, y) = M(y, x)$
(4) $\forall z \in D: M(x, y) \leq M(x, z) + M(z, y)$

In addition to this standard definition of metric, we associate with each domain and metric two additional parameters: a *diameter* and a *radius*. The diameter is an upper bound on all the distances among the values of the domain; it will be used for *estimating* unknown distances (if an upper bound cannot be provided, then ∞ is used). The radius establishes standard neighborhoods; given a value, it determines the values that are *close* to it. The radius is also used to *scale* distances; dividing a distance by the radius yields a measure of proximity that is independent of the particular domain and metric.

A database is *metricized* if at least one data metric is associated with each of its domains. If a domain has more than one metric, then one of them is designated as the *primary* metric.

2.2 Types of Data Metrics

In general, data metrics can be thought of as *procedures* with input (x, y) and output $M(x, y)$. These procedures may involve both *computation* and *retrieval.* Three types of database metrics are of particular interest: *computational*, *tabular*, and *referential*, and they are described below.

A data metric is *computational* if it derives its distances by computation only (i.e., no retrievals are involved). An example of a computational metric on a numerical domain, such as SALARY, is the absolute value of the difference between two numbers. An example of a computational metric on a nonnumerical domain, such as PERSON_NAME, is a procedure that determines the degree of similarity between two strings of characters.

A data metric is *tabular* if it derives its distances by retrieval only (i.e., no computations are involved). The distance between every two values of the domain is stored in a table, and the metric simply searches this table. Naturally, relations provide convenient storage for distance tables. An example of a tabular metric is the geographic distance between locations.

Each database relation has a designated attribute which is its *key*. Technically, this key provides a means for unique identification of tuples of the relation. Semantically, it may be regarded as the *topic* of the relation. In other words, the nonkey attributes may be regarded as a *description* of the key attribute. Distances between values of the key attribute may then be interpreted as the differences between their descriptions; that is, some combination of the individual distances between the corresponding components of the descriptions. In this definition, some individual distances may be given more weight than others, and some distances may be ignored altogether. Therefore, each relation provides distance information for the domain of its key attribute. Such metrics are called *referential* metrics. As an example, consider a relation FILM that describes motion pictures with the following attributes: TITLE, DIRECTOR, CATEGORY, and RATING. Assume TITLE (of domain FILM_TITLE) is the key, and let `(Psycho, Hitchcock, Suspense, 3.5)` and `(Modern_Times, Chaplin, Comedy, 4.0)` be two tuples from this relation. The distance between the titles `Psycho` and `Modern_Times` may be defined as some combination of the individual distances between `Hitchcock` and `Chaplin`, `Suspense` and `Comedy`, and `3.5` and `4.0`.

When a domain cannot be provided with a suitable metric, the following DEFAULT metric should be used:

$$\text{DEFAULT}(x, y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$

When a domain is provided with this computational metric, together with the radius 0, the effect is that of isolation: Given a value of that domain, no other value is close to it. Obviously, by using the metric DEFAULT with radius 0 throughout, the database reverts to a conventional (nonmetricized) database.

RELATION	ATTRIBUTE	KEY	DOMAIN
FILM			
	TITLE	*	FILM_TITLE
	DIRECTOR		PERSON_NAME
	CATEGORY		FILM_CATEGORY
	RATING		FILM_RATING
THEATER			
	T_NAME	*	THEATER_NAME
	OWNER		THEATER_OWNER
	LOCATION		NEIGHBORHOOD
	CAPACITY		NO_OF_SEATS
ENGAGEMENT			
	FILM	*	FILM_TITLE
	THEATER	*	THEATER_NAME
	OPEN_DATE		DATE

Fig. 1. Scheme of database on films and theaters.

2.3 Example

As an example, Figure 1 describes a database on films and theaters. Each attribute is followed by its domain, and key attributes are indicated by a star. Figure 2 describes the metrics of this database. For each domain, one or more metrics are listed, with the primary metric first. Each metric is described by its name and type, its diameter and radius, and a short description of the semantics of the distance. A small instance of this database is shown in Figure 3.[5]

The Films and Theaters database consists of three relations with a total of eleven attributes. These attributes draw their values from nine different domains. The nine domains are measured by seven different data metrics. The data metrics are described below.

Values of FILM_RATING are assumed to be numbers between 0 and 4, and values of NO_OF_SEATS are assumed to be integers not greater than 2000. In each case distances are measured by the computational metric ABS, by which the distance between two values is the absolute value of their difference.

Values of PERSON_NAME, THEATER_OWNER, FILM_TITLE, and THEATER_NAME are assumed to be arbitrary strings of characters. In each case distances are provided by the computational metric STRING, which implements an algorithm that rates the similarity of the strings by values between 0 (identical) and 1 (entirely different).

Values of DATE are assumed to be strings of the form MM-DD-YY where MM is a month (01–12), DD is a day (01–31), and YY is a year (00–99). Distances between dates are measured by the computational metric DATE as the number of days between the dates.

Distances between values of FILM_CATEGORY and NEIGHBORHOOD are obtained from tables by these names. These tables are shown in Figure 4.

Finally, FILM_TITLE and THEATER_NAME also have referential metrics, obtaining their distance information from the relations FILM and THEATER,

[5] Film ratings are from Leonard Maltin's *TV Movies* (Signet, New York, 1982).

DOMAIN	METRIC	TYPE	DIAMETER	RADIUS	SEMANTICS
PERSON_NAME	STRING	com	1	0.2	Persons with similar names
FILM_TITLE	STRING	com	1	0.2	Films with similar titles
	FILM	ref	10	2	Films with similar attributes
FILM_CATEGORY	FILM_CATEGORY	tab	3	1	Film categories that are most similar
FILM_RATING	ABS	com	4	0.5	Film ratings within half notch
THEATER_NAME	STRING	com	1	0.2	Theaters with similar names
	THEATER	ref	10	2	Theaters with similar attributes
THEATER_OWNER	STRING	com	1	0.2	Owners with similar names
NEIGHBORHOOD	NEIGHBORHOOD	tab	50	5	Neighborhoods within 5 miles
NO_OF_SEATS	ABS	com	2000	200	Capacities within 200 seats
DATE	DATE	com	365	7	Open dates within 7 days
STUFF	DEFAULT	com	1	0	Exact matches only

Fig. 2. Metrics of database on films and theaters.

respectively. To explain how referential metrics are used, the procedure to determine the distance between two film titles is subsequently outlined.

First, the descriptions of the two films are retrieved, and the distances between their directors, categories, and ratings are obtained. Next, these individual distances are *adjusted* by multiplying them by 1.83, 1.47, and 2.20, respectively. Finally, the adjusted distances are combined by means of the root of the sum of their squares. Denoting individual distances by their attribute names, the distance between film titles is

$$\sqrt{(1.83 \cdot \text{DIRECTOR})^2 + (1.47 \cdot \text{CATEGORY})^2 + (2.20 \cdot \text{RATING})^2}.$$

The adjustment has three purposes: (1) to correct for the fact that different metrics are involved, (2) to reflect the relative importance of the attributes in the description, and (3) to guarantee that resulting distances would always be in a prescribed range. The first purpose is achieved by *scaling* each distance by its respective radius; in this example, the three distances are divided by 0.2, 1, and 0.5, respectively. The second purpose is achieved by *weighting* each distance by a constant supplied by the user to express the relative importance of this attribute; in this example, we assume that the multipliers are 1, 4, and 3, respectively. Since the original distances were in the ranges [0, 1], [0, 3], and [0, 4], respectively, the scaled and weighted distances are now in the ranges [0, 5], [0, 12], and [0, 24], respectively. If we were to combine distances in these ranges, we would

FILM			
TITLE	DIRECTOR	CATEGORY	RATING
Four_Feathers	Korda	Adventure	3.5
Modern_Times	Chaplin	Comedy	4.0
Psycho	Hitchcock	Suspense	3.5
Rear_Window	Hitchcock	Suspense	4.0
Robbery	Yates	Suspense	3.0
Star_Wars	Lucas	Adventure	3.5
Surf_Party	Dexter	Drama	0.0

THEATER			
T-NAME	OWNER	LOCATION	CAPACITY
Chinese	Mann	Hollywood	815
Egyptian	UA	Westwood	730
Music_Hall	Laemmle	Beverly_Hills	630
Odeon	Cineplex	Santa_Monica	414
Rialto	Independent	Downtown	567
Village	Mann	Westwood	452

ENGAGEMENT		
FILM	THEATER	OPEN_DATE
Modern_Times	Rialto	12-19-86
Star_Wars	Rialto	12-26-86
Star_Wars	Chinese	01-16-87
Rear_Window	Egyptian	12-12-86
Surf_Party	Village	11-28-86
Robbery	Odeon	01-23-87
Modern_Times	Odeon	01-30-87
Four_Feathers	Music_Hall	11-21-86

Fig. 3. Instance of database on films and theaters.

FILM_CATEGORY		
VALUE_1	VALUE_2	DISTANCE
Comedy	Drama	2
Comedy	Adventure	3
Comedy	Suspense	3
Drama	Adventure	2
Drama	Suspense	2
Adventure	Suspense	1

NEIGHBORHOOD		
VALUE_1	VALUE_2	DISTANCE
Beverly_Hills	Downtown	15
Beverly_Hills	Hollywood	8
Beverly_Hills	Santa_Monica	10
Beverly_Hills	Westwood	5
Downtown	Hollywood	10
Downtown	Santa_Monica	20
Downtown	Westwood	18
Hollywood	Santa_Monica	15
Hollywood	Westwood	10
Santa_Monica	Westwood	5

Fig. 4. Tabular metrics for database on films and theaters.

obtain distances in the range [0, 27.29]. To create a range of film distances that would be between 0 and 10, each individual distance is also multiplied by $(10^2/27.29^2)^{1/2} = 0.37$, yielding the multipliers shown in the formula.

As mentioned earlier, the radiuses specified in the metrics table determine whether values may be considered "close" or not. For example, two locations are considered close if they are within 5 miles; two capacities are considered close if they are within 200 seats; two films are considered close under the computational metric STRING if they are within 0.2 units of string distance, and under the referential metric FILM if they are within 2 units of film distance. In the example, the FILM distance between `Psycho` and `Rear_Window` is 1.1, so they are considered close; on the other hand, `Modern_Times` and `Surf_Party` are far apart: The FILM distance between them is 9.5.

2.4 Metric Design

The process of defining a new database is usually referred to as *database design.* A database designer models real world environments with elements of the data model, such as relations, keys, and constraints. In the extended model this process now also includes the determination of the appropriate metrics and their parameters (i.e., diameters and radiuses) for each database domain. Proper selection of these metrics and parameters is critical to the successful handling of vague queries. Some guidelines follow.

Often, database domains are numerical, and the absolute value distance is satisfactory. Sometimes, although a domain is nonnumerical, its values are strictly ordered (for example, a domain RANK with values such as `Excellent`, `Good`, `Fair`, and `Poor`). Such domains are easily metricized by mapping the domain onto a range of integers (while preserving the order), using the absolute value metric to derive distances, and then storing the distances in a table.

Metrics can also be derived from domain *partitions.* Assume that a domain can be partitioned into a collection of disjoint sets called *clusters*, each containing values that are judged to be similar. A tabular metric can then be defined as follows: All intracluster distances (distances between two values that are in the same cluster) are set to 0, and all intercluster distances (distances between two values in different clusters) are set to 1. This metric can be refined if a *hierarchical* partitioning of the domain is available (i.e., clusters are possibly partitioned into further subclusters). The metric is derived from the clusters at the bottom level (level 0). Again, the distance between two values that are in the same cluster is set to 0. The distance between two values that are not in the same cluster is set to the level of the cluster that contains both. For example, if at level 0 x and y are not in the same cluster, but x and y are in the same cluster at level n, then the distance between x and y is set to n. Thus, the hierarchical partitioning provides more refined intracluster distances. Clustering has been used extensively in information retrieval systems as an automatic classification method, usually to improve search performance: When similar documents are clustered together, then it may be sufficient to compare retrieval requests only with *cluster representatives* (for a review of clustering methods see [25, chap. 3] or [21, pp. 137–140]). However, although document clustering is usually derived from a similarity measure, the process here is roughly the inverse: We derive a metric

from clusters that have already been constructed. Indeed, the methods sketched above for deriving metrics from clusters are the inverses of known methods for deriving clusters from similarity measures. Possibly, other clustering methods may be adapted as well.

A domain that is defined by a reference relation may be provided with a referential metric: The distance between every two values of this domain is inferred from the difference between their descriptions in the reference relation. At times, it may be necessary to define a more complex metric, for example, a metric on character strings that considers similarities in *appearance* of characters or similarities in *pronunciation* (e.g., based on the Soundex method [12, pp. 391–392]). Such metrics should be defined through a computation. Finally, whenever a suitable metric cannot be defined, the default metric should be used.

Some data metrics definitions are permanent. For example, a computational metric, such as STRING, can be defined to handle every possible pair of values from that domain. However, tabular metrics may need to be updated after each addition to the domain. For example, if a new film category is added to FILM_CATEGORY, its individual distances from every other film category need to be incorporated into the metric. This suggests that tabular metrics are more suitable for static domains.

In selecting standard radiuses, a good practice is to adopt the distance that is the "smallest distance of significance." For example, a significant difference between theater capacities may be determined to be 200 seats, and a significant difference between two film ratings may be determined to be 0.5. With such radiuses, the standard neighborhood of a value will include all the values that are similar to it. If the metric was obtained by clustering, the intracluster distance may be selected as the radius. If the values of the domain are scattered arbitrarily, the radius may be defined as some fraction of the diameter of the domain. For example, if the maximal distance between locations is 50 miles, a standard radius of 5 miles will incorporate approximately 10 percent of all locations into each standard neighborhood.

3. VAGUE QUERIES

In this section we give a formal definition of vague queries; we show how the query language QUEL is extended to allow the specification of vague queries; and we describe a ranking of answers to vague queries.

3.1 A Formal Definition

The formal treatment of vague queries will be done in the context of tuple relational calculus. The definitions for tuple calculus are taken with minor changes from [24, pp. 156–158]. A tuple relational calculus query is an expression of the form $\{t \mid \psi(t)\}$, where t is a tuple variable and ψ is a formula in predicate logic with t as its only free variable. Except for t, every other tuple variable of ψ must be associated with exactly one relation. The ith component of a tuple variable u is denoted $u.i$. If u is a tuple variable associated with relation R, and A is an attribute of R, then $u.A$ denotes the component of u for the attribute A.

The atomic formulas of ψ may be of three kinds:

(1) $(u \in R)$, where R is a relation name and u is a tuple variable. These atomic formulas are used to associate variables with relations, as discussed above.
(2) $(u.i\ \theta\ v.j)$, where u and v are tuple variables, and θ is one of the following comparators: $=, \neq, <, \leq, >, \geq$.
(3) $(u.i.\ \theta\ a)$, where u and θ are as above, and a is a constant.

Assume now that x and y are values from the same domain D, and let M and r be, respectively, a metric on this domain and its radius. We define a new comparator, called *similar-to* (denoted $\sim$), as follows:

$$x \sim y \quad \text{if} \quad M(x, y) \leq r$$

Thus, two values from the same domain are *similar* if the distance between them is smaller than the radius. We extend the definition of atomic formulas to allow θ to be $\sim$. Note that the *similar-to* comparators in ψ may involve different metrics and radiuses.

A *vague query* is a tuple calculus query that incorporates *similar-to* comparisons (also called *vague qualifications*). Every specific query can now be relaxed into a vague query by substituting any of its *equal-to* comparators with *similar-to* comparators. Since the answer to a vague query always contains the answer to the specific query from which it was derived, the vague query is more *general* (in the sense of [16]) than the specific query.

Consider, for example, the following tuple calculus query to retrieve all the theaters in Westwood that show adventure films:

$$\{x \mid (\exists f)(\exists t)(\exists e)(f \in \text{FILM}) \wedge (t \in \text{THEATER}) \wedge (e \in \text{ENGAGEMENT})$$
$$\wedge\ (x = e.\text{THEATER}) \wedge (e.\text{FILM} = f.\text{TITLE}) \wedge (e.\text{THEATER} = t.\text{T\_NAME})$$
$$\wedge\ (t.\text{LOCATION} = \texttt{Westwood}) \wedge (f.\text{CATEGORY} = \texttt{Adventure})\}$$

Since the only theaters in Westwood are `Egyptian` and `Village`, and neither shows an adventure film, this query will return a null answer.

Assume now that we change this query, so that the location constraint becomes t.LOCATION $\sim$ `Westwood`. Both operands are from domain NEIGHBORHOOD, which has radius 5. Therefore, the new constraint is satisfied with locations that are within a distance of 5 from `Westwood`; namely, `Westwood`, `Beverly_Hills`, and `Santa_Monica`. Consequently, the new query will return `Music_Hall` in Beverly Hills, which shows an adventure film. If, instead, we change the category constraint to c.CATEGORY $\sim$ `Adventure`, then all theaters in Westwood that show either adventure or suspense films will be retrieved (in the example, `Egyptian`). And if both changes are made, then `Odeon` in Santa Monica, which shows a suspense film, will be retrieved in addition to the previous two theaters.

The *similar-to* comparator can also be used between two variables. Assume that the original query is changed so that the constraint that joins the ENGAGEMENT and FILM relations is relaxed to e.FILM $\sim$ f.TITLE. All Westwood theaters that show films that are *close* to adventure films will be retrieved. Similarly, if the constraint that binds the free variable x is relaxed to $x \sim e$.THEATER, then all

theaters that are *close* to Westwood theaters that show adventure films will be retrieved.

3.2 Expressing Vague Queries in QUEL

To demonstrate how vague queries are expressed in an actual query language, we choose QUEL [23]. QUEL uses a **retrieve** statement that corresponds to the following family of tuple calculus queries:

$$\{t^{(n)} \mid (\exists u_1) \cdots (\exists u_m)(u_1 \in R_1) \wedge \cdots \wedge (u_m \in R_m) \wedge (t.1 = u_{i_1} \cdot j_1) \wedge \cdots \wedge (t.n = u_{i_n} \cdot j_n) \wedge \phi\}$$

where ϕ is a tuple calculus formula composed of atomic formulas with *and* and *or* operators (note that negation can always be effected by changing atomic formulas to use complementary comparators). Although this family of queries is a strict subset of the queries of tuple calculus, it is a powerful subset. As an example, the previous query is expressed in QUEL as follows:

```
range of f is FILM
range of t is THEATER
range of e is ENGAGEMENT
retrieve (e.THEATER)
  where e.FILM = f.TITLE
  and e.THEATER = t.T_NAME
  and f.CATEGORY = Adventure
  and t.LOCATION = Westwood
```

To specify vague queries in QUEL only one minor syntactical modification is necessary: The symbol ?= is used for the *similar-to* comparator, and vague queries are specified simply by using ?= in the **where** part of the **retrieve** statement. Thus, a request to retrieve the theaters close to Westwood that show adventure-like films is expressed with the following vague query:

```
range of f is FILM
range of t is THEATER
range of e is ENGAGEMENT
retrieve (e.THEATER)
  where e.FILM = f.TITLE
  and e.THEATER = t.T_NAME
  and f.CATEGORY ?= Adventure
  and t.LOCATION ?= Westwood
```

Notice that vague queries tend to be short: When trying to express vague queries in a system that supports only specific queries, queries often tend to use many disjunctions.

3.3 Ranking Answers

Each answer to a vague query involves a "compromise," which is the deviation of the values used to derive this answer from the values specified in the query. Given two answers, the one that requires a smaller compromise may be considered *more optimal.* It is usually desirable to present the answers to the user in their order of optimality.

Let $\{t \mid \psi(t)\}$ be a vague query from the family defined above. Without loss of generality we shall assume that ϕ is in conjunctive normal form (i.e., ϕ is a chain of subformulas connected with *and* operators, where each subformula is a chain of atomic formulas connected with *or* operators). Let α_i $(i = 1, \ldots, k)$ be the subformulas of ϕ that include vague qualifications, and let $\beta_{i,j}$ $(j = 1, \ldots, n_i)$ be the vague qualifications in α_i. Let T denote the answer to this vague query.

Assume $u_1, \ldots, u_m$ are tuples from $R_1, \ldots, R_m$, respectively, that satisfy ϕ. Each *similar-to* comparison in ϕ now involves a particular distance. Let $d_{i,j}$ denote the distance involved in the comparison $\beta_{i,j}$. For each subformula α_i there are n_i such distances. Since α_i is a disjunction of comparisons, the *minimums* of these distances is the compromise necessary to satisfy the subformula α_i. Since ϕ is a conjunction of such subformulas, the *root of the sum of the squares* of the minimums is the compromise necessary to satisfy ϕ.

Thus, a single distance is obtained for each combination of tuples that satisfies ϕ. Since each answer in T may be derived from more than one combination of tuples that satisfies ϕ, the *minimum* of these single distances is the compromise associated with each answer. This final value determines the optimality of each answer.

The individual distances, $d_{i,j}$ are not the "raw" distances delivered by the metrics but *adjusted* distances. The adjustment is twofold: To correct for the fact that different metrics are involved, each distance is *scaled* by its respective radius. To allow users to express their individual views of optimality, each distance is multiplied by a *weight* supplied by the user.

For example, assume a person interested in seeing an adventure film in Westwood or in Hollywood whose rating is at least 3.0. Except for the rating, this person is willing to relax all other constraints. This request is expressed with the following vague query:

```
range of f is FILM
range of t is THEATER
range of e is ENGAGEMENT
retrieve (e.FILM, e.THEATER)
  where e.FILM = f.TITLE
  and e.THEATER = t.T_NAME
  and f.CATEGORY ?= Adventure
  and f.RATING ≥ 3.0
  and (t.LOCATION ?= Westwood
  or t.LOCATION ?= Hollywood)
```

Assume that this person is more willing to compromise on the location of the theater than on the category of the film and expresses these priorities with a pair of weights; for example, 1 for LOCATION and 3 for CATEGORY. Every raw distance between locations will be divided by 5 (its radius) and multiplied by 1 (its weight), and every raw distance between film categories will be divided by 1 (its radius) and multiplied by 3 (its weight). The films and theaters returned in response to this vague query will be ordered according to the root of the sum of the squares of the (adjusted) distance between the category of the film and adventure and the smallest of these distances: the (adjusted) distance between the location of the theater and Hollywood and the (adjusted) distance between

the location of the theater and Westwood. This ranking will reflect the priorities of this person.

In our example, the neighborhoods Santa Monica and Beverly Hills are similar to Westwood or Hollywood and the film category suspense is similar to adventure. Consequently, there are four engagements that satisfy this vague query: `(Star_Wars, Chinese)` is an adventure film showing at a Hollywood theater (total compromise 0); `(Four_Feathers, Music_Hall)` is an adventure film showing at a Beverly Hills theater (total compromise 1); `(Rear_Window, Egyptian)` is a suspense film showing at a Westwood theater (total compromise 3); and `(Robbery, Odeon)` is a suspense film showing at a Santa Monica theater (total compromise 3.16).

4. IMPLEMENTATION

The extensions to the relational data model that have been described in this article should become an integral part of the database system. However, it is also possible to provide similar functionalities by constructing a simple system "on top" of existing database systems. The advantage of this approach is that it can also be implemented in cases in which the database system in use cannot be modified. The main disadvantage is that query processing is less efficient. The prototype system VAGUE is of the latter kind. VAGUE was implemented on top of the database system INGRES [23] using the programming language C in the environment of the UNIX[6] operating system running on a Sun computer.

4.1 The Components of VAGUE

The VAGUE system includes components of three kinds:

—Metric information.
—A vague query interpreter.
—DBA tools.

Metric information is stored either in INGRES relations (alongside the actual database relations) or as executable programs. In particular, each INGRES database is augmented with the following items:

(1) A database relation, called SCHEME, that lists the database relations with their attributes and the domains of the attributes (as in Figure 1). Database attributes that do not have domains are assumed to be from a single domain STUFF.[7]

(2) A database relation, called METRIC, that describes the various metrics (as in Figure 2). Domains that do not have metrics (and the domain STUFF) are assumed to have the metric DEFAULT.

(3) For each tabular metric, a database relation that stores distances between values of the domain (as in Figure 4). Note that tabular metrics assume that the

[6] UNIX is a trademark of AT&T Bell Laboratories.

[7] This relation is necessary only because INGRES does not implement the concept of domains.

domain is defined implicitly (i.e., it consists of the values currently stored in the database attributes that are associated with it). For efficiency, these relations are indexed on their first two attributes.

(4) For each computational metric, a program that computes distinces between values of this domain. Note that computational metrics assume that the domain is defined abstractly (in effect, this abstract definition is present only in the form of input validity checks that are incorporated into the program). Several popular computational metrics such as ABS, STRING, and DATE are built into VAGUE and need not be defined separately.

The vague query interpreter is an interactive program: It solicits from the user queries in extended QUEL, processes them in the INGRES system, and delivers the results back. This component is described in more detail in Section 4.2.

The DBA tools assist the database administrator in the administration of the metrics, which is still under development. Eventually, we expect to have these three tools:

(1) A program to assist the DBA in updating the relations SCHEME and METRIC. It will prompt the DBA for the necessary information, check for validity of the inputs, and so on. In particular, if a domain is to have a new tabular metric, it will define the necessary INGRES relation that will store the distances.

(2) A program to assist the DBA in updating tabular metrics. Given the name of a tabular metric, it will determine the domain, prompt the DBA with pairs of values from this domain that currently do not have distances, solicit the missing distances, and store the information as triplets. The program will also check that distances are consistent with the requirements of metrics.

(3) A program to assist the DBA in defining computational metrics.

Although it is possible to verify that the definitions of tabular metrics satisfy the formal requirements of a metric, it is usually impractical to verify the definitions of computational metrics. The validity of referential metrics relies on the validity of the definitions of the component metrics and on the formula used to combine them (e.g., the formula used by VAGUE combines proper metrics into a proper metric). Consequently, it is possible to provide VAGUE with distance measures that are not true metrics.[8]

4.2 The Vague Query Interpreter

When invoked with a name of an INGRES database, the vague query interpreter (also called VAGUE) verifies that all the necessary relations and programs are indeed available. Thereafter, the interpreter goes into query processing mode. After a query is entered, VAGUE scans it for vague qualifications. For each *similar-to* found, it determines the domain of its operands (if the operands do not have a common domain, an error message is displayed). It then retrieves from the relation METRIC the possible interpretations for this *similar-to* comparator and displays them to the user. For example, consider the following query to

[8] On the other hand, at times it may be *desirable* to adopt distance measures that are not proper metrics (e.g., do not satisfy the triangle inequality).

retrieve the theaters near Westwood that are showing films like Psycho:

```
range of e is engagement
range of t is theater
retrieve (e.theater)
where e.film ?= "Psycho"
and e.theater = t.t_name
and t.location ?= "Westwood"
```

First, VAGUE asks the user about the first vague qualification.[9]

```
Analyzing vague qualification
e.film ?= "Psycho".
Possible interpretations:
1. Films with similar titles.
2. Films with similar attributes.
3. None of the above (use exact matches only).
Please select [1-3]: 2
```

The first two options represent the known metrics for the domain FILM_TITLE. The last option is actually the metric DEFAULT, which the user may select if one of the previous metrics are satisfactory. If the user selects 2, then the user's priorities are questioned.

```
To discover similarities among values of
engagement.film you must determine the
importance of each attribute [0-10]:
1. film.director: 1
2. film.category: 4
3. film.rating: 3
```

The values provided are used to combine the individual distances of the referential metric FILM into distances between film titles.

Similarly, for the second vague qualification, VAGUE displays

```
Analyzing vague qualification
t.location ?= "Westwood".
Possible interpretations:
1. Neighborhoods within 5 miles.
2. None of the above (use exact matches only).
Please select [1-2]: 1
```

[9] User answers are shown in italics.

At this point VAGUE has all the information necessary to process this vague query. However, before doing so, it offers the user two additional options. First, it asks the user whether the answer should be ranked.

```
Should answer be ranked?
1. Yes
2. No
Please select [1-2]: 1
```

If the user selects 1, and the query includes more than one vague qualification, then the user's priorities are examined.

```
To rank the answer you must determine the importance
of each qualification in the overall query [0-10]:
1. e.film ?= "Psycho": 1
2. t.location.film ?= "Westwood": 1
```

These weights would enable the system to rank the answer.[10]

When answering vague queries, it may be beneficial to include in the answers the values of the *similar-to* operands upon which each answer is based. If the user's notion of similarity does not match the system's, these augmented answers reduce the risk of confusion. In this example, the answer is based on films that were found to be similar to the specified title and locations that are similar to the specified location, but the user did not request a listing of the films or locations. Noticing this, VAGUE gives the user the following option:

```
Should answers include the selected values of
t.location and e.film ?
1. Yes
2. No
Please select [1-2]: 1
```

If 1 is selected, then all the variables that are involved in *similar-to* comparisons are added to the **retrieve** list. (Note that if a similarity comparison involves two variables, then both variables are added.) If 2 is selected, then the query is left unchanged.

Although these interactions with the user are usually quite brief, they can all be avoided by switching VAGUE to *terse* mode. In this mode VAGUE employs the primary metrics with equal weights for all referential metrics. It does not rank the answer, and it does not expand the **retrieve** list. Additional switches are available for editing queries, obtaining help, and so on.

At this point VAGUE selects an execution strategy and issues the necessary QUEL queries to perform the vague query. When done, it presents the answers

[10] Notice that this ranking would override any other ordering specified in the query itself.

to the user. Assuming the user requested both ranking and an expanded **retrieve** list, the result would be

theater	location	film
Egyptian	Westwood	Rear_Window
Odeon	Santa_Monica	Robbery

If the user did not request an expanded **retrieve** list, then only the leftmost column would be displayed. If the answer is empty, then VAGUE gives the user the following option:

```
No data matched.
You may
1. Retry, allowing weaker similarities.
2. Quit.
Please select [1-2]: 1
```

If the user selects 1, then the previous query is repeated with wider neighborhoods. VAGUE doubles each of the radiuses used in the processing of the query and tries again. This process is repeated until the query matches some data or the query is abandoned by the user.

4.3 Performance Issues

In the simplest form of query processing, the database system iterates over the tuples of a relation searching for tuples that satisfy a combination of specific qualifications. Each qualification is of the form $A\ \theta\ a$, where A is a particular attribute of the relation, a is a particular constant, and θ is a specific relationship (e.g., $=$, $\geq$). To check whether a specific qualification is satisfied requires only few computer instructions.

To handle vague queries, this search process must be extended to enable it to check vague qualifications of the form $A \sim a$. For each such check the system must compute the distance between a and the value of the attribute A in the current tuple. If the metric is computational (either built-in or user defined), a procedure has to be executed. If the metric is tabular, one additional tuple must be retrieved from the distance relation. If the metric is referential, two additional tuples must be retrieved, and the distances between their components must be computed (possibly requiring additional retrievals) and combined.

Thus, computational metrics do not require any retrieval, and their efficiency relies entirely on the number of computer instructions that have to be executed. To improve the efficiency of tabular metrics, the distance relations may be *indexed* on the first two attributes. Obviously, referential metrics require the most processing. Indeed, VAGUE does not allow "nesting" of referential metrics (i.e., the attributes of a reference relation should not be measured themselves by referential metrics). The performance of referential metrics would be improved considerably if the distances between every two key values of the reference relation (i.e., the distance between their descriptions) are precomputed and stored. Indeed, the result would be a tabular metric whose distances are derived

from a reference relation. The drawback is that users would not be able to provide their own weights for the tuple distance computation.

We have discussed processing of queries by iterating over relations. In practice, query processing is often performed through *indexes.* An index for an attribute A stores values of A with pointers to tuples of the relation that have these values in attribute A. Specific qualifications, such as $A = a$, are then processed very efficiently. Similarly, it is possible to construct *similarity indexes.* A similarity index for an attribute A will store values of A with pointers to tuples of the relation that have similar values in attribute A. Vague qualifications, such as $A \sim a$, can then be processed efficiently. *Similarity indexes* can be constructed for attributes that are measured by any type of metric, except when distances cannot be precomputed (i.e., when user-provided weights are to be taken into consideration). In principle, a vague query, whose vague specifications involve only metrics for which similarity indexes are available, should not require more processing time than the corresponding specific query.

Finally, although vague queries tend to be less efficient than specific queries, one should keep in mind that the only alternative to a vague query is to attempt a *set* of specific queries or a more *complex* query that involves many disjunctions (and this alternative is available only if the user is aware of close values).

4.4 Database Update

With referential and tabular metrics, database updates may require additional distance information. The task of detecting the need for additional distance information may be relegated to the referential integrity mechanism [5, chap. 12] in the underlying database system (assuming that such a mechanism is supported). If a domain is measured by a referential metric, then attributes of this domain should all be linked through referential integrity constraints to the key attribute of the reference relation. If a domain is measured by a tabular metric, then the attributes of this domain should all be linked to the first (or second) attribute of the distance relation.

Consider now a user update that affects a database value from domain D. Assume D is measured by a referential metric. If the update was in the key attribute of the reference relation itself, then, obviously, the necessary distance information is available in that relation. If the update was in another database attribute, then the system would have already verified that the new value is present in the key attribute of the reference relation (and therefore also the necessary distance information). Assume now that D is measured by a tabular metric. In this case the system would have already verified that the new value is present in the first (or second) attribute of the distance relation. That is, the distance of this new value from at least one other value is available. Until all other distances are provided, the distances between this value and other values will be estimated by the diameter.

Integrity constraints can also be used to check that each tabular metric is consistent with the requirements of a metric (e.g., the triangle inequality).

5. INCOMPLETE INFORMATION

In recent years there has been much interest in issues regarding databases with incomplete information (for a review of this topic see [13, chap. 12]). Incomplete

information in metricized databases involves two new issues, first, how the availability of distances affects the conventional approaches to incomplete information, and, second, how to deal with incompleteness of the distance information itself. The discussion in this section focuses on tabular metrics; however, much of it can be adapted to other types of metrics.

5.1 Partial Metrics

Data distances are information, and, like other kinds of database information, it may be incomplete (i.e., the metric is only a *partial* function). Distances that are missing may be *estimated* with the following method.

Let M be a partial metric on domain D, and let d be the diameter of D. Initially, all unknown distances are estimated with the range $[0, d]$. Assume that $M(u, v)$, where $u, v \in D$, is such an estimated distance. Let $w \in D$ be a third value and assume that both $M(u, w)$, and $M(v, w)$ are known distances. The triangle inequality constraint provides information on the distance between u and v. This distance must maintain

$$|M(u, w) - M(w, v)| \leq M(u, v) \leq M(u, w) + M(w, v)$$

And, in general, let

$$p = \mathbf{max}\{|M(u, w) - M(w, v)| \mid w \in D\}$$
$$q = \mathbf{min}\{M(u, w) + M(w, v) \mid w \in D\}$$

Then $p \leq M(u, v) \leq q$. Thus the triangle inequality constraints provide a better (i.e., narrower) range $[p, q]$, as an estimate for the unknown distance between u and v.

As an example, consider the previous database and assume that a new value `Malibu` is added to domain NEIGHBORHOOD and that the only known distances from `Malibu` are

```
Malibu Beverly_Hills 10
Malibu Santa_Monica 7
```

Distances from Malibu to the other three places are estimated as follows:

```
Malibu Downtown [13, 25]
Malibu Hollywood [8, 18]
Malibu Westwood [5, 12]
```

Consequently, a query that involves, for example, locations within 15 miles from `Malibu` will consider `Santa_Monica`, `Beverly_Hills`, and `Westwood`. Of course, these range distances will introduce range distances into any other metric that is based on this metric (e.g., the metric THEATER).

5.2 Located Nulls

We have discussed the situation in which the distance between two known values is unknown. The dual situation, when a value is unknown, but its distance from some other values is known, is also interesting.

Often, the value that should be stored in a particular position in a tuple is not known, but it is clear that some appropriate value does exist. The prevailing approach in such situations is to fill these "vacancies" with *null* values. The information encapsulated in such nulls is rather limited: A value does exist but is missing. With *marked* nulls [13, p. 379] it is also possible to capture situations in which the same missing value is known to occupy several "vacancies."

If marked nulls are treated as regular values of the corresponding domain, then distances may involve them as well. As marked null for which some distance information is available (whether exact or only range estimates) is termed a *located* null.

The advantage of located nulls is that they participate in retrieval. As an example, consider the previous database and a film called `Duck_Soup`, whose category is unknown. Although the person who provided the information about this film was unsure about its category, it could be described as "having elements of both a comedy and a musical." If a simple null value is stored under CATEGORY, this film would never be retrieved on the basis of its category. In a metricized database, however, the distances of this null value from other values may be recorded. For example, the distances between the category of `Duck_Soup` and both `Comedy` and `Musical` may be estimated to be in the range $[0, r]$, where r is the standard radius. In this way, the information available is captured, and thereafter this film will be retrieved whenever a query specifies its category as either "close to comedy" or "close to musical."

The information encapsulated in a located null may also be communicated back to the users. Whenever a located null has to be printed, it can be described in terms of values that are in its neighborhood. Thus, in response to a query on the CATEGORY of `Duck_Soup`, the system would print ~ (`Comedy`, `Musical`).

6. CONCLUSION

Many retrieval requests are intrinsically vague, and systems that allow users to express vague queries directly (rather than require them to iterate through numerous specific queries) are more cooperative and possibly more efficient. Although issues of vague retrieval have been addressed in related disciplines, particularly information retrieval and fuzzy systems, current relational database technology does not provide adequate tools for performing vague retrieval.

The purpose of VAGUE is to enhance relational database systems with vague retrieval capabilities. The principal design guideline behind VAGUE has been to realize this goal with only minimal deviation from this popular model. This was achieved by extending the model with a single concept (data metrics) and a standard query language with a single feature (a *similar-to* comparator). The metric DEFAULT guarantees that standard databases are indeed a special case of metricized databases, and it allows database designers to introduce metrics only where they appear to be useful. The experience of VAGUE suggests that with modest programming effort it is possible to extend current relational database systems to provide useful vague retrieval capabilities (possibly even through interfaces that are purely external).

VAGUE incorporates several features that contribute to its flexibility. For example, it allows multiple metrics for each domain, with the ability to select the appropriate metric for each query; it allows users to judge the relative importance of attributes of referential metrics; and it allows users to express their willingness to compromise in retrievals that involve several vague qualifications.

The design of VAGUE represents a compromise between the sometimes conflicting requirements for simplicity, flexibility, and efficiency. Some examples of design compromises are described below. Users of VAGUE cannot provide their own similarity thresholds for each vague qualification. It was observed that this will require that users become familiar with particular data metrics. Instead, VAGUE allows its users to double the radius and repeat the query. Similarly, except for the ability to enter weights for referential metrics, users of VAGUE are limited to interpretations of similarity (i.e., metrics and radiuses) that have been provided by others. Although it is possible to design an interface that will permit users to introduce their own interpretations of similarity, it was determined that the complexity of this task usually would exceed the expertise of many users, especially casual users. Instead, this task is reserved for database designers or administrators, and users are invited to select from menus of metrics that are currently supported. To prevent the querying process from becoming too tedious to the user, VAGUE tries to be economical in its dialogue with the user. At several places it may be possible to gain flexibility by additional interaction; for example, when a vague query does not match any data, it is possible to ask the user which *similar-to* comparator should be weakened (currently, VAGUE increases all radiuses simultaneously). Finally, since each tuple in an answer to a vague query must satisfy *all* the vague qualifications, it is possible that a tuple would not be retrieved, even if its total compromise is smaller than that of tuples that were retrieved. This approach was adopted primarily for reasons of efficiency. In addition, because the combination of individual distances into a single distance is sometimes risky, VAGUE prefers not to rely on it for *determining* its answers, only for *ranking* them.

The issue of appropriate similarity measures for retrieval has been researched and debated extensively. Our purpose in designing and implementing VAGUE is not to resolve this issue by adopting any one particular approach but to provide relational databases with a flexible mechanism with which different kinds of data metrics may be implemented and tested.

A legitimate concern is that vague queries will be satisfied by meaningless values, and we already emphasized the importance of selecting all the metrics and parameters carefully. Also, by extending the answers to include the values with which the vague qualifications were satisfied (a feature available in VAGUE), users can monitor the judgements made by the system. Finally, it can be assumed that users who consciously present vague queries (to systems or to humans) are well aware of the fact that subjective judgement is involved and would probably examine answers to vague queries more carefully than answers to specific queries.

ACKNOWLEDGMENT

The implementation of VAGUE was carried out by Fawzan Malas.

REFERENCES

1. BOLC, L., AND JARKE, M., Eds. *Cooperative Interfaces to Information Systems. Topics in Information Systems*, Springer-Verlag, Berlin, West Germany, 1986.
2. BUCKLES, B. P., AND PETRY, F. E. A fuzzy representation of data for relational databases. *Fuzzy Sets Syst. 7*, 3 (May 1982), 213–226.
3. CODD, E. F. A relational model for large shared data banks. *Commun. ACM 13*, 6 (June 1970), 377–387.
4. CORELLA, F., KAPLAN, S. J., WIEDERHOLD, G., AND YESIL, L. Cooperative responses to boolean queries. In *Proceedings of the IEEE Computer Society 1st International Conference on Data Engineering* (Los Angeles, Calif., Apr. 24–27). IEEE Computer Society, Washington, D.C., 1984, pp. 77–85.
5. DATE, C. J. *An Introduction to Database Systems.* Vol. 1 (4th Ed.). Addison-Wesley, Reading, Mass., 1986.
6. GINSBURG, S., AND HULL, R. Order dependency in the relational model. *Theor. Comput. Sci. 26*, 1, 2 (Sept. 1983), 149–195.
7. HEROT, C. Spatial management of data. *ACM Trans. Database Syst. 5*, 4 (Dec. 1980), 493–513.
8. ICHIKAWA, T., AND HIRAKAWA, M. ARES: a *relational database with the capability of performing flexible interpretation of queries. IEEE Trans. Softw. Eng. SE-12*, 5 (May 1986), 624–634.
9. JONES, W. P. On the applied use of human memory models: the memory extender personal filing system. *Int. J. Man-Mach. Stud. 25*, 2 (Aug. 1986), 191–228.
10. JONES, W. P., AND FURNAS, G. W. Pictures of relevance: a geometric analysis of similarity measures. *J. Am. Soc. Inf. Sci. 38*, 6 (Dec. 1987), 420–447.
11. KAPLAN, S. J. Cooperative responses from a portable natural language query system. *Artif. Intell. 19*, 2 (Oct. 1982), 165–187.
12. KNUTH, D. E. *The Art of Computer Programming.* Vol. 3, *Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
13. MAIER, D. *The Theory of Relational Databases.* Computer Science Press, Rockville, Md., 1983.
14. MCLEOD, D. J. High level definition of abstract domain in a relational data base system. *J. Comput. Lang. 2*, 3 (July 1977), 61–73.
15. MOTRO, A. BAROQUE: A browser for relational databases. *ACM Trans. Off. Inf. Syst. 4*, 2 (Apr. 1986), 164–181.
16. MOTRO, A. SEAVE: A mechanism for verifying user presuppositions in query systems. *ACM Trans. Off. Inf. Syst. 4*, 4 (Oct. 1986), 312–330.
17. MOTRO, A. Supporting goal queries in relational databases. In *Proceedings of the 1st International Conference on Expert Database Systems* (Charleston, S.C., Apr. 1–4). Institute of Information Management, Technology, and Policy, University of South Carolina, Columbia. S.C., 1986, pp. 85–96.
18. MOTRO, A., D'ATRI, A., AND TARANTINO, L. The design of KIVIEW: an object-oriented browser. In *Proceedings of the 2nd International Conference on Expert Database Systems* (Tysons Corner, Va., Apr. 25–27). George Mason University, Fairfax, Va., 1988, pp. 17–31.
19. PRADE, H., AND TESTEMALE, C. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Inf. Sci. 34*, 2 (Nov. 1984), 115–143.
20. SALTON, G., Ed. *The SMART Retrieval System—Experiments in Automatic Document Processing.* Prentice Hall, Englewood Cliffs, N.J., 1971.
21. SALTON, G., AND MCGILL, M. J. *Introduction to Modern Information Retrieval.* McGraw-Hill, N.Y. 1983.
22. STONEBRAKER, M., AND KALASH, J. Timber: a sophisticated database browser. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City, Mexico, Sept. 8–10). VLDB Endowment (Morgan-Kaufmann, Los Altos, Calif.), 1982, pp. 1–10.
23. SUN MICROSYSTEMS. *SunINGRES Manual Set.* Release 5.0 (Part Number 800-1644-01), Mountain View, Calif., 1987.
24. ULLMAN, J. D. *Principles of Database Systems.* Computer Science Press, Rockville, Md., 1982.
25. VAN RIJSBERGEN, C. J. *Information Retrieval* (2nd Ed.). Butterworths, London, 1979.
26. WILLIAMS, M. D. What makes RABBIT run? *Int. J. Man-Mach. Stud. 21*, 4 (Oct. 1984), 333–352.
27. YANG, C. S., AND SALTON, G. Best-match querying in general database systems—a language approach. In *Proceedings of COMPSAC 1978—the IEEE Computer Society 2nd International*

Computer Software and Applications Conference (Chicago, Ill., Nov. 13–16). IEEE Computer Society, Washington, D.C., 1978, pp. 458–463.

28. ZEMANKOVA, M., AND KANDEL, A. Implementing imprecision in information systems. *Inf. Sci. 37*, 1, 2, 3 (Dec. 1985), 107–141.
29. ZLOOF, M. Query-by-Example: a database language. *IBM Syst. J. 16*, 4 (Dec. 1977), 324–343.

Received February 1987; revised February 1988, accepted June 1988.