Incomplete Path Expressions and their Disambiguation

Yannis E. Ioannidis* **Yezdi Lashkari†**
Computer Sciences Department, University of Wisconsin, Madison, WI 53706
{yannis,yezdi}@cs.wisc.edu

Abstract

When we, humans, talk to each other we have no trouble disambiguating what another person means, although our statements are almost never meticulously specified down to the very last detail. We 'fill in the gaps' using our common-sense knowledge about the world. We present a powerful mechanism that allows users of object-oriented database systems to specify certain types of ad-hoc queries in a manner closer to the way we pose questions to each other. Specifically, the system accepts as input queries with incomplete, and therefore ambiguous, path expressions. From them, it generates queries with fully-specified path expressions that are consistent with those given as input and capture what the user most likely meant by them. This is achieved by mapping the problem of path expression disambiguation to an optimal path computation (in the transitive closure sense) over a directed graph that represents the schema. Our method works by exploiting the semantics of the kinds of relationships in the schema and requires no special knowledge about the contents of the underlying database, i.e., it is domain independent. In a limited set of experiments with human subjects, the proposed mechanism was very successful in disambiguating incomplete path expressions.

1 INTRODUCTION

Given the question

```
'What are the courses of the Arts department?',
```

humans have no difficulty in interpreting it as meaning

```
'What are the courses taught by faculty
                    of the Arts department ?'.
```

In a typical university setting, however, there are a myriad of other options of courses that are associated with the Arts department, some of them mildly plausible, some frankly ludicrous. For example, the options

```
'courses taken by students in the Arts dept',
'courses taken by students taking courses
        taught by faculty of the Arts dept',
'courses taught by TAs taking courses
        taught by faculty of the Arts dept'
```

are all valid associations between the Arts department and a set of courses. Technically, the question `'What are the courses of the Arts department ?'` is underspecified, and hence has a number of possible interpretations. As humans, however, we know exactly what the question means. In everyday life, we 'fill in the gaps' in the specification of the question using our common-sense knowledge about teaching assistants, universities, and student life with absolutely no difficulty.

Supporting user interfaces with similar functionality in database management systems (DBMSs) has become increasingly important in recent years for several reasons. The need to expand the user base of DBMSs much beyond its current constituency (mostly database-literate people) requires that users can interact with a system with as little technical knowledge as possible, possibly without even knowing the schema of the database. Also, many current and future applications of DBMSs, e.g., scientific computing and decision support, require user interactions based on many ad-hoc queries, instead of the more conventional invocations of pre-compiled and stored application programs. Finally, the size of schemas continuously grows and the complexity of the desired queries follows along, so that remembering the database schema and writing queries become impossible and tedious, respectively. The combination of these three trends places a great demand for ad-hoc query interfaces that accept incompletely specified queries with minimal conformance to the schema, which are then completed by the system, benefiting both naive and expert users.

This paper introduces a mechanism with the above functionality for path expressions in object-oriented (OO) query languages. Incorporating the proposed technique inside an OODBMS will allow users to pose queries with incomplete (and therefore ambiguous) path expressions. In response, for each path expression, users will be presented with a (hopefully singleton) set of fully-specified path expressions and asked to approve some subset of it. Each presented path expression will be consistent with the original incomplete one and a likely completion based on the semantics of the relationships in it. A database query flow diagram

*Partially supported by the National Science Foundation under Grants IRI-9113736, IRI-9224741, and IRI-9157368 (PYI Award), and by grants from DEC, IBM, HP, AT&T, and Informix.

†Current address: The Media Laboratory, MIT, E15-305, Cambridge, MA 02139.

SIGMOD 94- 5/94 Minneapolis, Minnesota, USA
© 1994 ACM 0-89791-639-5/94/0005..$3.50

enhanced with the above mechanism is shown in Figure 1, where the path expression completion module introduces a loop between the user and the path expression evaluator.

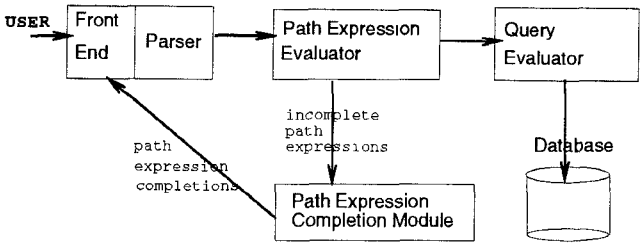


Figure 1: Flow of Queries/Path Expressions

The proposed approach is domain independent, i.e., it uses no knowledge on the real-world semantics of the data in any specific database. Instead, it exploits the semantics of the various kinds of relationships supported by the OO data model, and is therefore applicable on any schema with no need for additional information.

2 DATA MODEL AND QUERY PRIMITIVES

In this section, we describe some generic modeling and querying primitives that are important to our work. These appear in many semantic and OO systems, so the techniques developed are widely applicable.

2.1 The Data Model

Real-world entities are modeled by objects. Objects are grouped together in *classes*, which capture the objects' common properties. The *primitive* classes of *Integers*, *Reals*, *Character Strings*, and *Booleans* are system-provided and are denoted by I, R, C, and B, respectively. All other classes are user-defined. Binary *relationships* of various kinds describe the connections between objects in the schema classes. Schemas are represented as directed graphs: each class is a node in the graph (a circle for primitive classes and a rectangle for others), and each relationship is an edge between two nodes. Each relationship has a name, which if unspecified, is equal to the name of its target class. Although not necessary for the techniques developed, in this paper, we assume that for every relationship in a schema its inverse is present as well. Figure 2 shows a simple schema representing information about students, professors, departments, and universities. (For simplicity, inverse relationships are not shown.)

In presenting our methodology in this paper, we deal with five major kinds of relationships between classes. An *Isa* relationship connects a *subclass* to a *superclass*. Its inverse is a *May-Be* relationship. The semantics are that all objects in the subclass are also instances of the superclass (inclusion) and that the subclass inherits

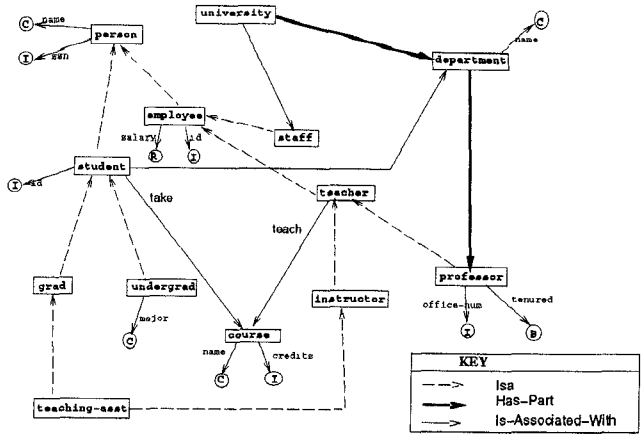


Figure 2: A Simple Schema

all the relationships of the superclass (specialization). The subclass may refine (redefine) these relationships and possibly define its own additional relationships. Multiple inheritance is allowed. Examples are student *Isa* person and person *May-Be* student.

Likewise, a *Has-Part* relationship connects a *superpart* class to a *subpart* class. Its inverse is an *Is-Part-Of* relationship. The semantics are that objects of the superpart class *structurally contain* objects of the subpart class. Examples are university *Has-Part* department and department *Is-Part-Of* university.

Finally, an *Is-Associated-With* relationship connects two classes whose objects are mutually related in some form that is irrelevant to their structure. Its inverse is an *Is-Associated-With* relationship as well. Examples are student *Is-Associated-With* course and course *Is-Associated-With* student.

2.2 Path Expressions in Queries

2.2.1 Path Expressions

A *path expression* corresponds to a path in the schema graph. It starts at a class, called the *path expression root* (which cannot be a primitive class), and continues traversing relationships. For each relationship traversed, the path expression contains a connector symbol corresponding to the kind of the relationship and the relationship name. *Isa*, *May-Be*, *Has-Part*, *Is-Part-Of*, and *Is-Associated-With* relationships use the connector symbols @>, <@, $>, <$, and ., respectively.

Path expressions are the primary mechanism in OO query languages for specifying object relationships, and have the following semantics. When evaluated, a path expression results in all objects reachable from each object in the path expression root. Some sample path expressions in the schema of Figure 2 together with their meaning follow:

student.take.teacher
(teachers of courses taken by students)
student$>person.ssn
(soc. sec. nums of persons who are students)
department.student$>person.name
(names of persons who are students of departments)

In essence, a path expression is a very simple query of the functional form discussed in the introduction whose answer is a set of objects related in some way with another set of objects (those in the path expression root). The focus of this paper is on queries of this special form, so henceforth, the term 'query' means simply a path expression. The techniques we develop, however, are easily applicable to general queries since path expressions are a central feature of these.

2.2.2 Incomplete Path Expressions

As mentioned in Section 1, one would like to avoid having to explicitly specify long path expressions. To this end, we introduce one additional connector symbol, ~ . While all other connectors are matched by a single edge in the schema graph when interpreting a path expression, ~ may be matched by an arbitrarily long path whose two ends are those specified on the two sides of ~ . A path expression having at least one instance of the ~ symbol is called *incomplete*, otherwise it is called *complete*. For ease of exposition and without loss of generality, in this paper, we concentrate on path expressions that have only one instance of ~ and no other connectors. The general case is treated in [17].

Consider the path expression $\xi = s \sim N$, where s is a class and N is the name of at least one relationship in the schema. A complete path expression ψ on a schema is *consistent* with ξ if its root is s and its last relationship name is N. The semantics of ξ are that it is equivalent to the optimal complete path expression(s) that is (are) consistent with ξ, where optimality is related to intuitiveness and cognitive plausibility and is formally defined in Section 3. If there are many such optimal expressions, the user must choose one. In principle, there is an infinite number of path expressions that are consistent with a given incomplete one, since cycles in schemas can be traversed multiple times. Humans, however, do not think circularly unless a circularity is explicitly stated [6, 8, 11]. For this reason, cyclic path expressions are ignored when looking for the optimal path expressions that are consistent with ξ.

For example, for the schema in Figure 2, the names of all teaching-assistants (class *ta* for short) may be captured by the incomplete path expression *ta ~ name*. This is equivalent to both
ta@>grad@>student@>person.name
ta@>instructor@>teacher@>employee@>person.name,
which are the optimal path expressions consistent with the given incomplete one, since both of them have the intended meaning. Note that path expressions

ta@>grad@>student.take.student@>person.name
(names of students taking courses with TAs)
ta@>grad@>student.take.name
(names of courses taken by TAs)
ta@>instructor@>teacher.teach.name
(names of courses taught by TAs)
ta@>grad@>student.department.name
(names of departments of TAs)

also share their two ends with the incomplete path expression above, but are obviously not as intuitive as those denoting the names of teaching assistants.

3 PROBLEM FORMULATION

Given an incomplete path expression ξ, let Ψ denote the set of all valid acyclic path expressions that are consistent with ξ. Our problem is to determine the subset $\Psi_{opt} \subseteq \Psi$ that contains the optimal (cognitively most plausible) path expressions related to ξ. We map this problem to an optimal path computation over a labeled directed graph. To simplify the description of the mapping and subsequent algorithms, without loss of generality, we assume that path expressions are always between classes (nodes) in a schema graph, or equivalently that all relationships have their default name. We begin with a brief, generic, outline of optimal path computation problems, and then describe the details of this mapping.

3.1 Optimal Path Computations

Much work has been done on the problem of computing several properties that are specified over the set of paths in a labeled directed graph [1, 4, 15, 23]. Examples of such properties include shortest path, most reliable path, and bill of materials. Their computation is termed a *path computation*. Most path computation studies use a path algebra formalism introduced by Carre [4]. Below, we present a version of the formalism that is applicable to our problem.

There is a label associated with each edge and each path in the graph. The label of a path is computed as a function, called CON (for *concatenate*), of the sequence of labels of the edges in the path. In addition, a path set P is also associated with a label set, which is computed as a function, called AGG (for *aggregate*), of the labels of the paths in P. Informally, the CON function computes the value of the desired property of a path, while the AGG function selects the path(s) having optimal values of that property. Formally, the above is achieved by defining AGG as a unary function[1] on path-label sets,

[1]For most common problems, AGG is based on a total order over the labels and therefore always returns a singleton set. Then, it can be defined as a binary function on labels, like CON.

and CON as a binary function on path labels[2] that has an identity denoted by Θ. These satisfy the following properties (L_1, L_2, L_3 are labels and $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are label sets):

1. Associativity of CON:
   $\text{CON}(L_1,\text{CON}(L_2, L_3))=\text{CON}(\text{CON}(L_1, L_2),L_3)$
2. 'Associativity' of AGG:
   $\text{AGG}(\mathcal{L}_1 \cup AGG(\mathcal{L}_2 \cup \mathcal{L}_3))=$
   $\text{AGG}(\text{AGG}(\mathcal{L}_1 \cup \mathcal{L}_2) \cup \mathcal{L}_3)$
3. Fixpoint of AGG on singletons:
   $\text{AGG}(\{L_1\}) = \{L_1\}$
4. Identity label for CON:
   $\text{CON}(\Theta, L_1) = \text{CON}(L_1, \Theta) = L_1$

For example, in the shortest path problem, CON is + over the nonnegative reals, AGG is 'min', and Θ is 0, while in the most reliable path problem, CON is * over the numbers between 0 and 1, AGG is 'max', and Θ is 1. It is straightforward to verify that these satisfy the above properties.

In principle, path computations must examine all paths in a graph, which is very expensive and also creates termination problems for cyclic graphs. To avoid examining all paths and to ignore cyclic paths, existing path computation algorithms require the following additional two properties [1, 4, 15, 23]:

5. Θ is annihilator of AGG:
   $\text{AGG}\ (\mathcal{L}_1 \cup \{\Theta\}) = \{\Theta\}$
6. 'Distributivity' of AGG over CON:
   $\text{AGG}(\{\text{CON}(L_1, L_3),\text{CON}(L_2, L_3)\})=$
   $\text{CON}(\text{AGG}(\{L_1, L_2\}),L_3)$

The problem of generating plausible completions of incomplete path expressions can be mapped to an optimal path computation over a labeled graph. This is detailed in the following subsections, which explain the graph on which the path computation is performed and the corresponding AGG and CON functions.

3.2 The Graph

The path computation graph is the schema graph. The *label* on each edge is a pair of the connector denoting the kind of the corresponding relationship and the *semantic length* of the relationship, which is a measure of how far apart (semantically) the classes at its two ends are. The concept of semantic length is closely related to the notion of *semantic distance* between two concepts in psychology and cognitive science. We define the semantic length of *Isa* and *May-Be* relationships to be 0 and that of the remaining relationships to be 1. For example, in Figure 2, the *Has-Part* relationship from department to professor (whose name is *professor*) has label [$>,1] In what follows, we use the terms 'connector' and 'relationship kind' indistinguishably.

[2]By extenting our notation, we allow the arguments of CON to be label sets as well. In that case, CON is applied on each pair of labels from the input set(s) separately.

3.3 The CON Function

Given a path, the CON function should return a label for it that captures the meaning of the relationship connecting the classes at its two ends and its semantic distance. CON computes the connector in this label based solely on the connectors of its input labels via a binary function CON_c on connectors, and computes the semantic length separately.

3.3.1 Composing Connectors

Based on the above description, the connectors of the labels that may appear on the edges of the graph are all in the set $\Sigma' = \{@>, <@, \$>, <\$, .\}$. We introduce a CON_c function under which Σ' is not closed, i.e., the result of CON_c on two members of Σ' may be a connector outside of Σ'. The additional connectors necessary are called *secondary* to distinguish them from the *primary* connectors of Σ' and represent kinds of additional (indirect) relationships that may hold between two classes, as discussed below.

A *Shares-SubParts-With* (resp *Shares-SuperParts-With*) relationship indicates that the two participating classes are related via *Has-Part* (resp. *Is-Part-Of*) relationships with a third class capturing the fact that their objects may contain (resp. be contained in) common objects. The inverses of both relationships are of the same kind. The *Shares-SubParts-With* relationship is denoted by $.\_{SB}$, while the *Shares-SuperParts-With* relationship by $.\_{SP}$. For example,

engine *Has-Part* screw
screw *Is-Part-Of* chassis
$\Rightarrow$ engine *Shares-SubParts-With* chassis,

motor *Is-Part-Of* assembly
assembly *Has-Part* shaft
$\Rightarrow$ motor *Shares-SuperParts-With* shaft.

An *Is-Indirectly-Associated-With* relationship indicates that the two participating classes are related via some arbitrary sequence of relationships in some way other than sharing. Such relationships capture essentially a looser form of an association. They are denoted '..'. For example, in Figure 2,

dept *Is-Associated-With* student
student *Is-Associated-With* course
$\Rightarrow$ dept *Is-Indirectly-Associated-With* course,

which essentially captures the fact that each department (dept for short) is indirectly associated with the courses taken by its students.

Finally, excluding *Isa* and *May-Be*, each of the remaining primary and secondary relationships that we have discussed has a *Possibly* version. Such a

relationship from class A to class B indicates that objects of A may or may not be related to objects of B. *Possibly* relationships are denoted by placing a $\star$ immediately after the connector symbol for the corresponding plain relationship, e.g., the *Possibly-Has-Part* relationship is denoted by $\$>\star$. For example,

course *Is-Associated-With* teacher
teacher *May-Be* professor
$\Rightarrow$
course *Possibly-Is-Associated-With* professor,

which captures the fact that a course may be taught by a professor, but not necessarily.

Let Σ'' be the set of connectors denoting the kinds of all the secondary relationships introduced above. If $\Sigma = \Sigma' \cup \Sigma''$, then we define the function CON_c so that Σ is closed under CON_c. The definition of CON_c is shown in Table 1, where the value of $\text{CON}_c(r, c)$ is the table entry at row r and column c. The construction of

Input	@>	<@	$>	<$	.	SB	SP	..
@>	@>	<@	$>	<$	.	SB	SP	..
<@	<@	<@	$>$\star$	<$$\star$	.$\star$	SB$\star$	SP$\star$	..$\star$
$>	$>	$>$\star$	$>	.SB	..	SB	..	..
<$	<$	<$$\star$	.SP	<$	..	..	SP	..
.	.	.$\star$	..	..	..	..	..	..
SB	.SB	SB$\star$	..	SB	..	..	..	..
SP	SP	SP$\star$	SP	..	..	..	..	..
..	..	..$\star$	..	..	..	..	..	..

Table 1: The Function CON_c

this CON_c function was based on the semantics of each relationship kind and intuitive evidence on the meaning of their combinations [8]. For example, if A *Has-Part* B and B *Has-Part* C, then A *Has-Part* C, or if D *Is-Associated-With* E and E *May-Be* F, then D *Possibly-Is-Associated-With* F. For the sake of conciseness, the above table does not include the rows and columns for the *Possibly* connectors. The missing entries essentially generate three more tables like the one above (one for the case where only the first argument of CON_c is a *Possibly* connector, one for the case where only the second one is, and one for the case where both of them are). All three tables are identical and may be derived from the one above by replacing each entry by its *Possibly* version. In other words, once any of the arguments of CON_c is a *Possibly* connector, the result will always be a *Possibly* connector.

3.3.2 Adding Semantic Lengths

As defined in Section 3.2, the semantic length of a path is a measure of the semantic distance of the concepts at its two ends. In our context, the basis for this distance is the database schema graph. The actual length of a path is clearly different from its semantic length. For example, a long chain of contiguous *Part-Of* connectors is equivalent to a single *Part-Of* connector; the two should have identical semantic lengths. The semantic length that we study is calculated based on a generalization of the above example plus some additional considerations reflecting cognitive plausibility. This is better understood by the following (conceptual) path restructuring. (Below, a maximal path with certain properties is one that cannot be extended any further and maintain these properties.)

1. Any maximal contiguous series of one of the @>, <@, $>, and <$ connectors is replaced by a single edge with the same connector These are the connectors on which CON_c is idempotent.

2. In the result path of step 1, the first (or last) edge of any maximal contiguous series of interchanged @> and <@ connectors is removed.

The semantic length of a path is defined as the actual length of the path generated after step 2 above [3] Note that this is consistent with the definition of semantic length for single edges given in Section 3.2. Further, it essentially captures that multiple @> (or <@, $>, <$) relationships do not affect the semantic length (step 1), and that @> and <@ relationships contribute to the semantic length only when they are interchanged (step 2). The . relationships contribute their actual length. For example, the semantic length of '*teacher.teach.student.department$>professor*' is 4, while that of '*stuff@>employee<@teacher<@instructor<@teaching-asst@>grad@>student*' is 2.

3.4 The AGG Function

Given a set of path labels, the AGG function should return the optimal ones, i.e., those that indicate a stronger relationship between classes. AGG compares labels primarily on connectors and secondarily on semantic lengths (if one of the connectors is not more preferable than the others). We discuss the two types of comparisons separately.

3.4.1 Primary Ordering of Labels based on Connectors

As mentioned above, each path is associated with a connector in Σ obtained by repeatedly applying CON_c to the ordered sequence of connectors in the path Based on cognitive science and psychology studies of semantic relations [5, 6, 9, 20, 27], we construct a partial order of importance/strength of these connectors. The partial order is named *better-than*, denoted by $\prec$, and shown in Figure 3 with arrows from the worse to the

[3] In reality, in order for the semantic length of a path to be computed as part of the CON function, the label of a path must have two additional parts that contain the connectors of the first and last edge of the path. These would not affect anything else, but are necessary for the computation of semantic length. Because of the simplicity of the semantic length concept and to not overload notation, we have ignored this subtlety and have presented labels as connector-semantic length pairs.

better connector. Two connectors with a nontrivial path between them in Figure 3 are called *comparable*; otherwise, they are *incomparable*. Note that every connector is incomparable to itself, inverse connectors are incomparable, and every connector is incomparable with its *Possibly* version.

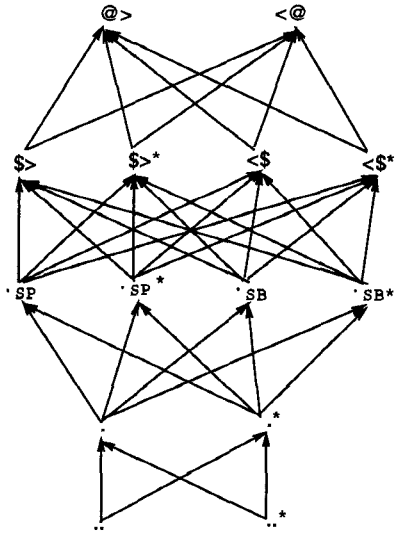


Figure 3: The Partial Order $\prec$ for Connectors

Clearly, there may be alternative orderings that are also plausible. We focused our attention to the one above because of its basis on cognitive and psychological evidence [20, 27], and because it gave the best results compared to other reasonable alternatives in a limited set of experiments that we conducted.

The above partial order is used in the computation of AGG as follows. For two labels $[c_1, f_1]$, $[c_2, f_2]$, if $c_1 \prec c_2$, then $\text{AGG}(\{[c_1, f_1], [c_2, f_2]\}) = [c_1, f_1]$.

3.4.2 Secondary Ordering of Labels based on Semantic Lengths

Cognitive Science studies support that "concepts with greater semantic distance between them are considered less plausible by humans than corresponding concepts with a lesser semantic distance" [9]. We have captured semantic distance by the *semantic length* of a path. Thus, for labels with connectors that are pairwise incomparable in $\prec$, AGG operates based on a comparison of their semantic length Specifically, for two labels $[c_1, f_1]$, $[c_2, f_2]$ such that $c_1 \not\prec c_2$ and $c_2 \not\prec c_1$, if $f_1 < f_2$, then $\text{AGG}(\{[c_1, f_1], [c_2, f_2]\}) = [c_1, f_1]$. In the above, $<$ is the natural total order on integers.

3.5 Properties of AGG and CON

One may easily verify that our CON and AGG satisfy properties 1-5 of Section 3.1:

1. CON is Associative. Thus, CON may be applied on the labels of the edges in a path in any order.
2. AGG is 'Associative'. Thus, AGG may be applied individually on each pair of labels in a label set.
3. AGG leaves singleton sets unchanged.
4. CON has [@>,0] as its identity.
5. AGG has [@>,0] as an annihilator. Thus, cyclic paths can be ignored as per the desired semantics.

Finally, in addition to the above, AGG and CON satisfy the following property

7. CON is monotonic with respect to AGG, i.e., for any two labels L_1, L_2,
   $\text{AGG}(\{L_1, \text{CON}(L_1, L_2)\}) = \{L_1\}$ or
   $\text{AGG}(\{L_1, \text{CON}(L_1, L_2)\}) = \{L_1, \text{CON}(L_1, L_2)\}$.
   Thus, extending a path can never improve its label with respect to AGG, so certain branch-and-bound type of optimizations can be applied when traversing a schema graph.

Property 7 is relevant only because we are interested in path computations between two given points; otherwise, it would be useless.

Unfortunately, property 6 of Section 3.1 ('distributivity' of AGG over CON) is not satisfied. This implies that certain techniques regarding common subpaths used by traditional transitive closure algorithms cannot be applied in our case. This is further discussed when our algorithm is presented in Section 4.

4 COMPLETION ALGORITHM

Based on the above path computation formulation, we have developed and implemented an algorithm to generate completions of incomplete path expressions. The algorithm performs a depth-first traversal of the schema graph and includes features that take advantage of the properties of AGG and CON discussed in Section 3.5, and also others that are beyond the path computation formulation and address specific characteristics of completing path expressions.

As a reference, we present below a depth-first search algorithm for a traditional path computation problem, where AGG and CON satisfy properties 1-6 of Section 3.1 and also the monotonicity property 7 of Section 3.5. Its input is the source and target nodes S and T and its output (as is common in the literature) is the optimal label(s) of paths from S to T. This may not be the only such algorithm, but is sufficient for our purposes. The fact that we can use depth-first search for path computations is due to properties 1-4. The effect of the remaining properties is discussed after the algorithm.

The algorithm uses local variables v, u to denote nodes and l_x, l_{xy} to respectively denote the labels of paths from S to x and the label of the edge (x, y), for arbitrary nodes x, y. It also uses the following global variables:

children[v] The set of children of v Each set is sorted in the order of best-to-worst label of the edge from v to help in branch-and-bound.

visited[v] A flag equal to **false** if v may be (re)visited and **true** otherwise. It is initialized to **false** for all nodes.

best[v] The set of optimal labels of paths that have been explored from S to v. It is initialized to $\emptyset$.

The algorithm is given below in the form of the 'traverse' routine, which should be called initially with arguments S and Θ (identity of CON):

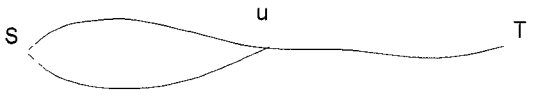
```
      proc traverse(v,l_v)
      begin
(1)   visited[v] := true;
(2)   if T ∈ children[v]
(3)   then best[T] :=
(4)          AGG({CON(l_v,l_vT)} ∪ best[T]) fi
(5)   for each u ∈ children[v]-{T} do
(6)      l_u := CON(l_v,l_vu),
(7)      if visited[u]=false ∧
(8)         AGG({l_u}∪ best[T]) ≠ best[T] ∧
(9)         AGG({l_u}∪ best[u]) ≠ best[u]
(10)     then best[u] := AGG({l_u}∪ best[u]),
(11)          traverse(u,l_u) fi
(12)  od
(13)  visited[v] := false
      end
```

Algorithm 1: Depth-First Search for Traditional Path Computation Problems

Lines (7), (8), and (9) test properties 5 (acyclicity), 7 (monotonicity), and 6 ('distributivity'), respectively. Note that if T is a child of v, it is explored out of order with respect to the other children of v so that a complete path may be discovered as early as possible, thus potentially blocking useless paths through the other children. For traditional problems, e.g., shortest path, each entry in best[] will always be singleton (or empty), because AGG is based on a total order. Other than that, Algorithm 1 is exactly what would be necessary. Below, we first discuss the ways in which we enhanced this algorithm for our purposes, and then conclude with the end result.

4.1 Non-'distributivity' of AGG over CON

Consider two paths between S and T that share a subpath from some node u to T. Assume that the first path has been explored by 'traverse' and u is visited through the second path. The 'distributivity' property of AGG over CON (property 6 in Section 3 1) permits the optimization that the subpath from u to T is reexamined



only if the label of the second path from S to T is better than or incomparable but distinct from the label of the first path This is achieved by line (9) of Algorithm 1. Unfortunately, the CON and AGG functions given in Sections 3.3 and 3.4 do not satisfy 'distributivity' for all inputs. Although the condition of line (9) may be satisfied, not exploring the path from u to T a second time is not always correct and plausible answers may be lost. In order to be as efficient as possible, our algorithm for path expression completion does not simply remove line (9), but enhances it so that it only blocks subpath exploration when this is safe. This is achieved via the use of *caution sets* for (the connector parts of) labels. The caution set of a label L_1 is the set of all labels L_2 for which $L_2 \prec L_1$ and there exists some label L_3 such that CON(L_1,L_3) and CON(L_2,L_3) are incomparable, i.e., that AGG({CON(L_1,L_3),CON(L_2,L_3)}) $\supset$ CON(AGG({L_1,L_2}),L_3). Caution sets are used in the following condition, which replaces the original one in line (9):

$$(\text{AGG}(\{l_u\} \cup \text{best}[u]) \neq \text{best}[u] \vee \text{caution}[l_u] \cap \text{best}[u] \neq \emptyset].$$

4.2 Need for Path Expressions

Clearly, given an incomplete path expression, returning simply the label(s) of the optimal consistent path expression(s) is not satisfactory; the path expression(s) themselves are needed. Thus, Algorithm 1 is modified in several places to handle paths: its input includes one more parameter that signifies the path from S to v explored currently, a list of the currently optimal paths from S to T is maintained, and the path to u via v is constructed. Most important though are changes in lines (8) and (9), the latter being orthogonal to the one above dealing with caution sets. Before, if l_u was already in best[T] or best[u], u was not visited again since the path label was the only interesting item. When interested in the actual path though, u must be reexplored in that case. Thus, line (8) and the original part of line (9) are respectively replaced by

$$l_u \in \text{AGG}(\{l_u\} \cup \text{best}[T]), \quad \text{and}$$
$$l_u \in \text{AGG}(\{l_u\} \cup \text{best}[u]).$$

4.3 Inheritance Semantics

It is interesting to note that, with one exception, the path computation formulation of Section 3 captures the traditional semantics of inheritance that other systems support. Given the expression $\xi = s \tilde{} N$, if there is an *Isa* path from class s to a class t with a relationship named N, that path will have the strongest

possible label and will be accepted as a completion of ξ. Similarly, multiple inheritance will result in multiple incomparable completions, which in our scheme will be resolved by the user. Thus, a system based on our formulation will behave like any other system that supports inheritance, except for one case of multiple inheritance, where all other systems accept a standard resolution mechanism, whereas in our case, the user must be involved in the loop. Specifically, consider

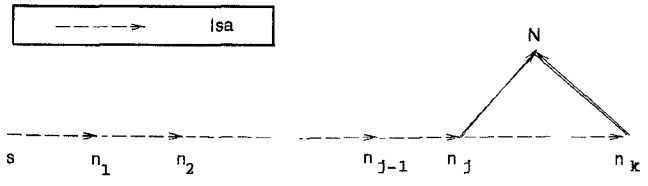


Figure 4: The Inheritance Semantics Criterion

Figure 4 and its two path expressions

$\psi_1 = s$ @>n_1 @>$n_2 \ldots$ @>n_j ϕ_1 N, and

$\psi_2 = s$ @>n_1 @>$n_2 \ldots$ @>$n_j \ldots$ @>n_k ϕ_2 N,

where s is an arbitrary path expression, N and $n_i, 1 \leq i \leq k$, are relationship (class) names, and ϕ_1 and ϕ_2 are arbitrary connectors except @>. Then, based on the traditional inheritance semantics employed by all systems, ψ_1 should be deemed preferable to ψ_2, and the root of s should inherit N from n_j and not its superclass n_k. When such a case arises, path expression ψ_1 is said to *preempt* path expression ψ_2, and the two of them are said to satisfy the conditions of the *Inheritance Semantics Criterion*. No path computation formulation can capture this criterion in terms of CON and AGG, because it is applicable only on full path expressions and not on arbitrary paths. To provide the same semantics as other systems, we enhance Algorithm 1 so that it can deal with this special case. Specifically, when complete paths are found (line (2)), the algorithm checks to see if there are paths that satisfy this criterion, in which case the appropriate preemptive action is taken.

4.4 Accepting Multiple Semantic Lengths

Our description of AGG in Section 3.4 compares semantic lengths strictly as numbers, where even a difference of 1 may result in rejecting paths. To allow more flexibility in treating semantic lengths, our algorithm uses a generalization of the original AGG function, denoted by AGG*. Its difference is that, given a set of labels with incomparable connectors, its output contains all labels whose semantic lengths are among the E lowest semantic lengths in the set, where E is a parameter that may be varied ($E \geq 1$).

4.5 The Enhanced Algorithm

Our algorithm for completions of path expressions is given below (Algorithm 2). It is Algorithm 1 modified as discussed in the previous subsections. It additionally uses the local variables p_x, p_{xy} to respectively denote paths from S to x and the edge from x to y, for arbitrary nodes x,y. It also uses the following global variables:

caution[l] It is the caution set of label l.

paths It is the set of optimal paths currently found from S to T. It is initialized to $\emptyset$.

Finally, it uses a procedure 'concat' that concatenates two paths, and a procedure 'update' that updates the 'paths' set when a new complete path is found that is not worse than earlier paths and also applies the Inheritance Semantics criterion. The algorithm is given again in the form of the 'traverse' routine, which should be called initially with arguments S, Θ, and S.

```
     proc traverse(v,l_v ,p_v)
     begin
(1)  visited[v] := true;
(2)  if T ∈ children[v]
(3)  then best[T] :=
(4)            AGG*({CON(l_v ,l_vT)} ∪ best[T]);
(5)       update(paths) fi
(6)  for each u ∈ children[v]-{T} do
(7)    l_u := CON(l_v ,l_vu); p_u := concat(p_v ,p_vu);
(8)    if visited[u]=false ∧
(9)      l_u ∈ AGG*({l_u } ∪ best[T]) ∧
(10)     ( l_u ∈ AGG*({l_u } ∪ best[u]) ∨
(11)       caution[l_u] ∩ best[u] ≠ ∅])
(12)   then best[u] := AGG*({l_u}∪ best[u]);
(13)        traverse(u,l_u ,p_u) fi
(14) od
(15) visited[v] := false
     end
```

Algorithm 2: Depth-First Search for Path Expression Completion

5 PRELIMINARY EXPERIMENTS

As mentioned in Section 1, the mechanism presented in this paper should be beneficial to both naive and knowledgeable users. In this section, we present the results of some preliminary experiments on a knowledgeable subject using a large, real-life, schema in the Moose data model and path expressions in the Fox query language [26]. These have been designed to serve the needs of a desktop Experiment Management System under development at the Univ. of Wisconsin - Madison [13]. Moose includes all the relationship kinds discussed in Section 2 plus additional ones, so the actual experiment dealt with more complex CON and AGG functions that captured the additional richness.

The goal of this experiment was to obtain some indication of the degree to which our mechanism can expedite querying by schema designers and database administrators, i.e., if it can be effectively used as a shorthand query formulation mechanism. We first present the metrics of effectiveness and overall methodology that were used in this experiment, and then the results.

5.1 Measures of Effectiveness

Let U be the set of complete path expressions meant by a user specifying an incomplete path expression, and S be the set of path expressions returned by the system. U may include more than one path expression in cases where the user consciously presents an incomplete path expression that is ambiguous. These two sets determine two important parameters of the effectiveness of information retrieval systems, *recall* and *precision* [24]. *Recall* is defined as the proportion of relevant answers that are retrieved, i.e., $|U \cap S|/|U|$. *Precision* is defined as the proportion of retrieved answers that are relevant, i.e., $|U \cap S|/|S|$. An ideal system has recall and precision values equal to 100 %. Clearly, one way of achieving a high recall rate is to retrieve all possible answers, but that will usually imply a very low precision rate. In general, there is an inherent trade-off between the two parameters that shifts as the number of answers returned by the system changes.

5.2 Experimental Methodology

We conducted the experiment on a Moose schema that captures the structure of the input parameters to CUPID, a large Fortran program used to simulate plant growth [22]. This schema was designed by a scientist in the Soil Sciences department of the Univ. of Wisconsin - Madison, who served as the human subject of this experiment as well. By the nature of experimental science, path expressions between almost any pair of classes on that schema are perfectly reasonable for queries. Given the large size of the schema (92 user-defined classes and 364 relationships), the formulation of ad-hoc queries is clearly a non-trivial task, even for the schema designer.

The experiment was conducted as follows. The schema designer was asked to propose ten ad-hoc incomplete path expressions on the schema diagram. For each of them, he specified the set U_0 of complete path expression(s) that he had in mind when formulating the incomplete one. Each incomplete path expression was then given as input to our algorithm, which produced the set S of path expression(s) that it derived as most plausible. In a very small number of cases, the schema designer thought that some path expression in $S - U_0$ was indeed natural and equally plausible with those in U_0, although he had overlooked it in the beginning. In such cases, we combined these path expressions with those in U_0 to obtain the final set U that was used for the calculation of recall and precision.

In addition to applying the standard algorithm on these incomplete path expressions, we also conducted (manually) a second experiment that used some domain-specific knowledge. Specifically, the schema designer specified that certain classes in the schema should never be a part of the completion of any incomplete path expression (they were auxiliary classes connected to a plethora of other classes but without much inherent semantic content). This allowed us to test the potential of domain-specific knowledge in further improving the effectiveness of the domain-independent approach presented in this paper.

5.3 Results

The results of the experiments are shown in Figures 5 and 6. The x-axis represents the value of the E parameter indicating the number of lowest semantic lengths maintained in the answer, while the y-axis represents the average of recall and precision, respectively, over ten incomplete path expressions. Overall, the system performed remarkably well in terms of being able to select the 'correct' path expression(s) among all those that were consistent with a given incomplete one. In particular, an average of over 500 acyclic path expressions are consistent with each incomplete path expression, whereas on the average, only 2-3 of them are returned by the algorithm when E=1.

The average recall of the system was at 90% and remained unaffected by changes in E. That is, increasing S did not affect $S \cap U$ (all additional path expressions, with higher semantic length, were not plausible) This indicates that semantic length does indeed capture strength of relationships verifying earlier observations [9]. On the average, 10% of all path expressions meant by the user were not returned by the algorithm. Most of them appeared to belong in special cases that are unlikely to be captured by a generic algorithm like the one presented here but would need some domain-specific knowledge beyond the limited forms with which we experimented.

As expected, the average precision of the system dropped dramatically (from 100% to 55%) as E increased, due to the resulting increases of S. Remarkably, for E=1, precision was at 100%, i.e., *all* path expressions returned by the algorithm with the least semantic length were indeed meant by the user. This justifies our use of the specific CON and AGG functions, including the choice of $\prec$ and the semantic length criterion.

Figure 6 also indicates that even a small amount of domain knowledge of the form mentioned above can dramatically improve the precision of the system. Many useless path expressions were blocked due to such knowledge, so that precision dropped to only 93% as E increased instead of 55% Note that the type of domain

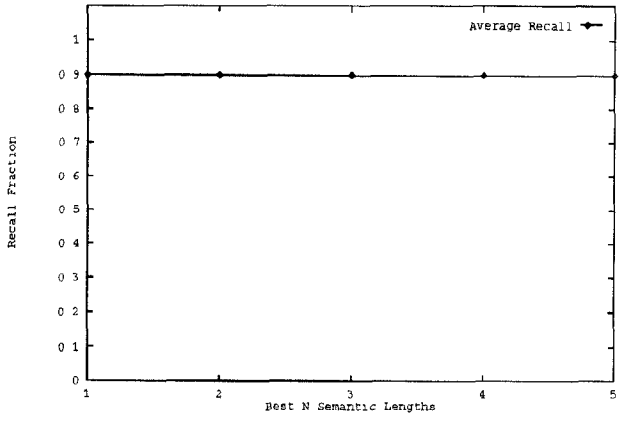


Figure 5: Average Recall Fraction

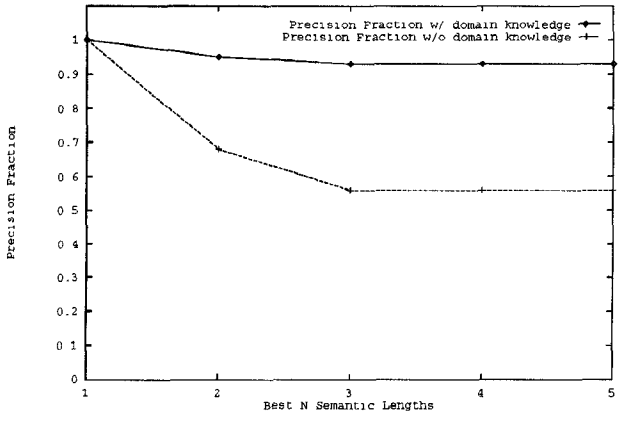


Figure 6: Average Precision Fraction

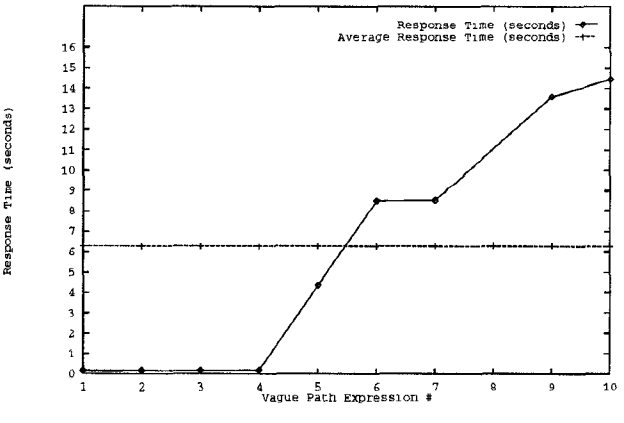


Figure 7: Response Time Per Query

knowledge that was used was only helpful in removing path expressions from the algorithm's output and not adding ones. This is why only precision is affected while recall remains unchanged.

We should mention that the average length of path expressions returned as an answer by the system was about 15. Clearly, specifying path expressions that traverse that many relationships is not a trivial task even for the schema designer. As the experiments show, our system does provide a convenient and effective shorthand mechanism to address a real problem.

5.4 Efficiency of Traversal

Given the size of the schema for CUPID, which is realistic for many database applications, the response time of the completion algorithm is an important issue. Independent of the effectiveness of the algorithm in finding the appropriate complete path expressions, a user should not wait too long for them. In the experiments with the CUPID schema designer, we looked into the number of recursive calls performed by the algorithm and their cost for each of the ten tested incomplete path expressions. Each recursive call corresponds to an exploration of a class node in the schema and takes on the average 0.17 msecs on a DecStation 5000/25 with 16M of memory.

Figure 7 plots the response time (y-axis) of the algorithm for each of the ten incomplete path expressions (x-axis), ordered in increasing processing complexity (for presentation reasons), when $E = 5$. Clearly, there is a large variance among the response times of processing different expressions, with some of them requiring almost no time, and on the average requiring 6.29 secs. Overall, however, even the most expensive expressions can be processed in almost real-time (14.45 secs) so that the user receives all the benefits of automatic completion without becoming impatient on the keyboard.

5.5 Summary of Results

The above experiment is by no means exhaustive. A comprehensive study should be performed with many schemas, users, and queries to precisely determine the effectiveness and efficiency of the proposed techniques. Nevertheless, the limited set of results obtained do indicate that our approach has potential. In addition, several interesting observations have been made, which should be helpful in designing a more complete algorithm, most notably that even restricted forms of domain-specific knowledge (which are easy to incorporate in a system) are very beneficial.

6 RELATED WORK

We have not been able to find any work that has a direct bearing on ours. There is some similarity with

research on DBMS user-interfaces and on common-sense reasoning, which we outline below.

6.1 Database User-Interfaces

Many user-interfaces have been proposed in the database literature with goals similar to ours, i.e., (i) allowing users with minimal knowledge about the schema to pose queries on the database and (ii) helping users in specifying very long queries Natural language interfaces hide the schema from users, while menu-driven and schema-based graphical interfaces guide users through the schema [2, 14, 16]. In both cases, users do not need to know the schema and are protected from making many errors when formulating queries. There are several drawbacks to these approaches, however. In general, natural language interfaces are far from being robust, except on very specific domains, and it is unclear if they will ever be preferred compared to more stylized languages and interfaces. Also, menu-driven and schema-based graphical interfaces do reveal the schema to users (which may not always be appreciated by them) and, as a side-effect, may have problems in dealing with very large schemas. Finally, most of the benefits of all such interfaces are with respect to the first goal above; for the most part, formulation of very long queries remains a tedious task

Our work serves both goals above. It can be incorporated in any of the aforementioned types of interfaces, and in some sense tries to bridge the gap between natural language and more formal, stylized, interfaces. The system must reveal a small part of the interface to users, i e , the classes and some relationship names, but not the specific structure of the relationships, which represents the bulk of the schema complexity This is nothing more than what the other approaches require as well. Further, path expressions only need to include their two end points (and optionally some intermediate points), thus relieving users from tedious typing, dragging, or clicking. In principle, it is applicable to arbitrarily large schemas and it is domain independent

Identifying the intended meaning of a user query is also an important issue in systems that provide cooperative query answers [7, 10, 12, 21]. In such systems, queries are syntactically correct and the goal is to use mostly domain-dependent knowledge to modify them so that the resulting answer is more informative, more succinct, clarifies user missuppositions or misconceptions, and takes into account the user's interests. We deal with a very different problem, where queries are syntactically incorrect (incomplete) and the goal is to correct them based on domain-independent schema information, and therefore the developed techniques are very different as well.

Finally, our problem is very similar to identifying a unique join query connecting a given set of attributes in relational systems that operate under the universal relation model [18, 19]. A basic foundation of such systems is the *relationship uniqueness assumption*, which states that there is a unique relationship among a given set of attributes that is implied by users when no explicit attribute connections are given. (Other connections must be explicitly specified.) These systems make some additional assumptions on the underlying database instance (e.g., the universal instance assumption) and then use various types of dependencies that are specified in the schema to identify a unique, most natural, relationship. In disambiguating incomplete path expressions, we essentially make the relationship uniqueness assumption as well, but none of the instance-based assumptions (which are not always realistic). We also use semantic information that is available in richer data models, which is at a higher level than common dependencies As a result, the developed approach is quite different from those proposed in universal relation systems.

6.2 Common-Sense Reasoning

A significant body of work in Artificial Intelligence is in the field of object hierarchies and common-sense reasoning. Various attempts have been published to formalize the mathematics of object hierarchies. Most of them concentrate on dealing with defeasible links, and hence non-monotonic inferences [25], while others have been concerned with approximate reasoning [3]. Due to the different emphasis (DBMS schemas do not contain defeasible links), the above work is orthogonal to ours.

Finally, there has been some work on constructing plausible inferences from pairs of adjacent relationships [8], which is similar to our notion of the CON function. This research uses a larger set of relationships than what Moose supports and is more oriented to various actions in English. It is only concerned with pairs of relationships, however, and does not attempt to make inferences on arbitrarily long series of them.

7 CONCLUSIONS

We have presented a domain-independent approach to generating plausible completions of incomplete path expressions by mapping the problem to a path computation over the schema graph. The associated algorithm is a modification of existing path computation algorithms to deal with some peculiarities of this problem We have also performed a very preliminary set of experiments on human subjects, which indicates that the proposed approach is both effective and efficient. The overall methodology can be generally applied to any semantically rich data model, by specifying appropriate CON and AGG functions on the kinds of relationships supported by the model. We should emphasize that although we have focused on a single ~ connector in path expressions, in general, our approach deals with

an arbitrary number of them [17]. Given the increasing demand for flexible user-interfaces by both naive and knowledgeable users, we believe that the presented work is a promising first step in the right direction.

Our current and future work includes several interesting issues that have arisen. First, we plan to conduct a comprehensive experiment with several human subjects, realistic queries, and large schemas, to solidify the evidence on the effectiveness of the presented approach. Second, we have already demonstrated that even simple forms of database-specific knowledge, when available, can improve the results. Studying several such forms of knowledge is also part of our plans. Moreover, the introduction of learning techniques based on user feedback is a promising mechanism to acquire arbitrary domain-specific and even user-specific knowledge that will be useful. Third, various psychological studies have indicated that when confronted with two homonymous concepts of widely differing sizes, humans tend to prefer the more specific or focused concept of the two, e.g., 'the courses I take' are preferred over 'the courses offered by my department'. We would like to investigate how such information can be incorporated into path-labels and used to discriminate between path expressions. Finally, although the CON and AGG functions discussed in this paper were chosen among ten and twenty corresponding alternatives, respectively, and gave very promising results, there is still room for further investigation of such functions that could be superior.

References

[1] R. Agrawal, S. Dar, and H. Jagadish, *Direct Transitive Closure Algorithms: Design and Performance Evaluation*, ACM TODS, Vol 15 (3), pp 427-458, 1990.

[2] C. Batini, T. Catarci, M. Costabile, and S. Levialdi, *Visual Query Systems, A Taxonomy*, Proc. IFIP Conf. on Visual Database Systems, 1991.

[3] D. Bobrow and A. Collins (eds), *Representation and Understanding*, Academic Press, New York, NY, 1975.

[4] B Carre, *Graphs and Networks*, Clarendon Press, Oxford, England, 1979.

[5] R. Chaffin, D. Herrmann, and M. Winston, *An Empirical Taxonomy of Part-Whole Relations : Effects of Part-Whole Relation Type on Relation Identification*, Language and Cognitive Processes, Vol 3 (10), pp 17-48, July 1988.

[6] R. Chaffin and A. Glass, *A Comparison of Hyponym and Synonym Decisions*, J. of Psycholinguistic Research, Vol 19 (4), pp 265-280, 1990.

[7] W. Chu, M. Merzbacher, and L. Berkovich, *The Design and Implementation of CoBase*, Proc ACM SIGMOD Conference, Washington, DC, pp 517-522, 1993.

[8] P. Cohen and C. Loiselle, *Beyond ISA: Structures for Plausible Inference in Semantic Networks*, Proc 7th National Conference on Artificial Intelligence, St. Paul, MN, pp 415-420, August 1988.

[9] A. Collins and M. Quillian, *Retrieval Time From Semantic Memory*, J. of Verbal Learning and Verbal Behaviour, Vol 8, pp 240-247, 1969

[10] F. Cuppens and R. Demolombe, *Cooperative Answering: A Methodology to Provide Intelligent Access to Databases*, Proc. 2nd Expert Database Systems Conf., Tysons Corner, VA, pp 333-353, April 1988.

[11] S. Fahlman, *NETL: A System for Representing and Using Real World Knowledge*, MIT Press, Cambridge, MA, 1979.

[12] T. Gaasterland, P. Godfrey, and J. Minker, *An Overview of Cooperative Answering*, J. of Intelligent Information Systems, Vol 1 (2), pp 123-157, Oct. 1992.

[13] Y. Ioannidis, M. Livny, E. Haber, R Miller. O. Tsatalos, and J. Wiener, *Desktop Experiment Management*, IEEE Data Engineering, March 1993.

[14] Y. Ioannidis (ed.), *Advanced User Interfaces for Database Systems*, ACM SIGMOD Record, Vol 20 (1), March 1992.

[15] Y. Ioannidis, R. Ramakrishnan, and L. Winger, *Transitive Closure Algorithms Based on Graph Traversal*, ACM TODS, Vol 18 (3), December 1993.

[16] T. Kunii (ed.), *Visual Database Systems*, North-Holland, Amsterdam, The Netherlands, 1989.

[17] Y. Lashkari, *Domain Independent Disambiguation of Vague Query Specifications*, MS Thesis, CS Dept., Univ. of Wisconsin - Madison, July 1993.

[18] D. Maier, D. Rozenshtein, and D. S. Warren, *Windows on the World*, Proc ACM SIGMOD Conference, San Jose, CA, pp 68-78, May 1983.

[19] D. Maier, J. Ullman, and M. Vardi, *On the Foundations of the Universal Relation Model*, ACM TODS, Vol 9 (2), pp 283-308, June 1984.

[20] G. Miller and C. Fellbaum, *Semantic Networks of English*, Cognition, Vol 41 (1-3), pp 197-229, 1991.

[21] A. Motro, *FLEX: A Tolerant and Cooperative User Interface to Databases*, IEEE Tran. on Knowledge and Data Engineering, Vol 2 (2), pp 231-245, June 1990.

[22] J. Norman, and G. Campbell, *Application of a Plant-Environment Model to Problems in Irrigation*, "Advances in Irrigation II", D. Hillel (ed), Academic Press, New York, NY, pp 155-188, 1983

[23] A. Rosenthal, et al, *Traversal Recursion: A Practical Approach to Supporting Recursive Applications*, Proc ACM-SIGMOD Conf., pp 166-176, 1986.

[24] G. Salton, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, Addison Wesley, 1989.

[25] D. Touretzky, *Implicit Ordering of Defaults in Inheritance Systems*, "Readings in Uncertain Reasoning", G. Shafer, and J. Pearl (eds), 1990.

[26] J. Wiener and Y. Ioannidis, *A Moose and a Fox Can Aid Scientists with Data Management Problems*, Proc 4th DBPL Workshop, New York, NY, August 1993.

[27] M. Winston, R. Chaffin, and D. Herrmann, *A Taxonomy of Part-Whole Relations*, Cognitive Science, Vol 11, pp 417-444, 1987.