Query Formulation from High-level Concepts for Relational Databases[*]

Guogen Zhang
IBM Santa Teresa Lab
San Jose, CA 95141, USA

Wesley W. Chu, Frank Meng, and Gladys Kong
Computer Science Department
University of California, Los Angeles, CA 90095, USA

Abstract

A new query formulation system based on a semantic graph model is presented. The graph provides a semantic model for the data in the database with user-defined relationships. The query formulator allows users to specify their requests and constraints in high-level concepts. The query candidates are formulated based on the user input by a graph search algorithm and ranked according to a probabilistic information measure. English-like query descriptions can also be provided for users to resolve ambiguity when multiple queries are formulated from a user input. For complex queries, we introduce an incremental approach, which assists users to achieve a complex query goal by formulating a series of simple queries. A prototype system with a multimodal interface using the high-level query formulation techniques has been implemented on top of a cooperative database system (CoBase) at UCLA.

1 Introduction

Many database applications require users to formulate ad-hoc queries instead of invocation of pre-compiled and stored queries. Thus there is a need to develop an intelligent query interface to allow users to specify queries by high-level concepts and constraints.

There are various techniques for formulating simple SELECT-PROJECT-JOIN (or SPJ) queries, such as universal relation model [1, 2] and Steiner tree approach [3]. The universal relation model [1, 2] based on the uniqueness assumption of relationships attempts to relieve users of the burden of specifying joins. However, it does not allow arbitrary user-defined concepts in a model, thus limiting its applicability [4]. Tree schemas derived from a cyclic schema by its maximal object theory may limit the queries that can be formulated.

Wald and Sorenson [3] formalized the query completion as a Steiner tree problem, and presented a search algorithm for partial 2-tree graphs. They used a deterministic directed cost for the edges, such as the cardinality of relationships, to measure the complexity of queries. Ioannidis and Lashkari [5] considered path expression completion in object-oriented queries with a partial order relationship between different paths for ranking. All these query formulation methods can only generate simple queries.

In [6], three high-level interfaces to database systems are discussed. These interfaces are concerned with browsing the database, retrieving neighborhood answers (similar-to) and providing a flexible interface where no query is rejected and explanations are given for null answers. The use of database contexts for disambiguating queries is developed in [7]. Dialogue is carried out with the user using a dialogue tree to determine what additional attributes the user is interested in.

It has been a difficult problem to formulate complex queries. In SQL, complex queries contain subqueries or the "`HAVING`" clause. Only when the query tools can support complex queries, will they be practical for end-users to solve real-world tasks.

There are some existing approaches to complex queries with limited success. Natural language interfaces to databases [8, 9] usually use pattern-based methods or parsing to translate a natural language input into a logical form and then to a query, which sometime is complex. The obstable is that it is usually difficult to understand an input for complex queries expressed in a natural language. Some other systems use visual languages with object-oriented techniques to compose complex queries by using high-level objects representing complex functions. [10] presents a methodology for using icons to query a database called Query By Icon (QBI). Each icon represents an object and provides a view of the database from the point of view of the object. A Binary Graph Model is used as the underlying semantic model. A progressive querying interface is presented in [11] in the visual programming paradigm with limited capabilities. We have developed a general graph search approach to formulating SPJ queries from incomplete user input. Our query formulator can formulate SPJ queries with aggregate functions and certain cyclic queries. Further,

[*]The work was performed while the first author was at Computer Science Department, UCLA. The research was supported by DARPA contract N66001-97-G8601.

we present an incremental approach that uses multiple SPJ queries to solve a complex query.

In our approach, a semantic graph, which models the objects in the database and user-defined relationships, can be semi-automatically generated from a database schema. Based on a graph search method, queries can be formulated by finding a set of paths in the semantic graph which encompasses the user input. Since it is possible to have multiple path sets for a given input, the system ranks the candidates based on the amount of information present in the nodes and links of the paths.

In an incremental query session, a subsequent query may use the results of the preceding queries (called *derived relations*) in addition to relations in the database. At each step, the user decides the next query step based on the intermediate results obtained. The system can also compose a complex query from the SPJ queries without executing them at each step if the user is not interested in knowing the intermediate result. Solving complex queries in such a manner mimics the human cognitive process and also provides a new way to formulate complex queries from a series of simple queries.

Our query formulation technique has several unique features. First, it does not have specific limitations on the graph structure, as required in [3], and it allows user-defined relationships incorporated into the graph. Therefore, it can be applied to most relational databases. Furthermore, some cyclic queries can be formulated, while previous systems were limited to tree queries only. Second, a probabilistic information measure is used for query ranking, which is adaptive to user feedback. Third, The search algorithm combines heuristics and exhaustive search with pruning, and is efficient and scalable. Finally, incremental query formulation provides a promising technique to extend the formulation capability beyond simple SPJ queries.

The rest of the paper is organized as follows. A discussion of the semantic graph is given in section 2. Section 3 describes query formulation from high-level concepts. Section 4 presents incremental query formulation for complex queries. Implementation and experience are presented in Section 5. Finally, a conclusion is given in section 6.

2 The Semantic Graph Model

In our semantic graph model for relational databases, nodes are used to represent relations and links are used to represent joins. Together they represent the following semantic constructs: (1) strong and weak entity types; (2) ISA relationship between entities; (3) HAS relationship between a strong and a weak entity; (4) simple link between entities without its own properties; (5) complex association between entities with its own properteis; and (6) user-defined relationships between entities. In addition, our semantic graph model contains more information to support user query interfaces and query formulation mechanisms.

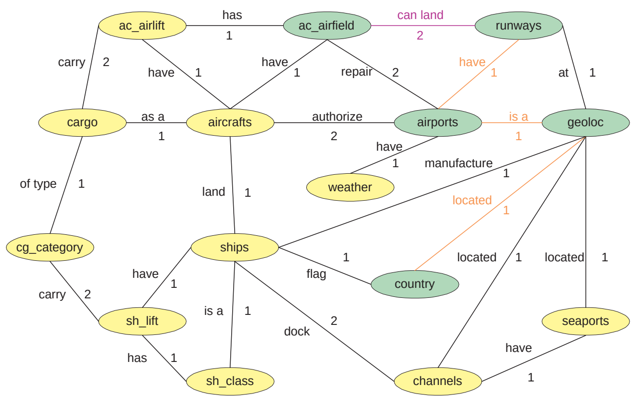


Figure 1: Semantic graph for the transportation database.

Formally, a *semantic graph* for a relational database is a weighted undirected graph $G = (V, E)$, where each node in the set of nodes V corresponds to a relation, and each link in the set of links E corresponds to a join between relations of the link's two end nodes. The joins can be natural or user-defined. Associated with each node and link is a conceptual term that can be used by the user to refer to the corresponding elements in the graph. Weights are assigned to the nodes and links in accordance with their relative importance, which are used in the query formulation. Lexical information is used to support English-like query descriptions.

As an example, part of the transportation database semantic graph is shown in Figure 1. It contains information about airports, aircrafts, seaports, channels, ships, etc. The link (`AIRPORTS`) `HAVE` (`RUNWAYS`) is an example of a natural join link, while the link (`AIRCRAFTS` with `AIRFIELD_CHARS`) `CAN LAND` (on `RUNWAYS`) is an example of a user-defined link. There are no self-join links in this graph. Note that the weight of each link is shown along the link, which gives a relative measure of the specificity of the link.

2.1 Semantics of Subgraphs and Queries

A link in a semantic graph represents a join. For example, in the transportation semantic graph, "`CAN LAND`" is a user-defined link between `AIRCRAFT_AIRFIELD_CHARS` and `RUNWAYS`. It corresponds to a complex join condition:

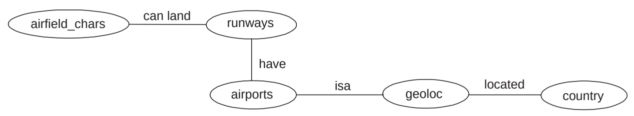


Figure 2: A query topic for "aircraft can land on airports at geographical locations of countries".

```
AIRCRAFT_AIRFIELD_CHARS.WT_MIN_AVG_LAND_DIST_FT
   <= RUNWAYS.RUNWAY_LENGTH_FT  AND
AIRCRAFT_AIRFIELD_CHARS.WT_MIN_RUNWAY_WIDTH_FT
   <= RUNWAYS.RUNWAY_WIDTH_FT
```

In general, a connected subgraph represents a relational algebraic expression consisting of a set of joins represented by the links. We call a connected subgraph a *query topic*, as shown in Figure 2.

The algebraic query expression corresponding to the query topic in Figure 2 contains four joins. When it is evaluated against the database, each of the resulting tuples will contain information about the airfield characteristics of an aircraft type, a runway, its airport, its geographical location information, and its country information, such that an aircraft of the aircraft type can land on the runway of the airport at the geographical location of the country.

Users are usually only interested in a subset of the tuples generated from the evaluation of a query topic by imposing selection conditions (i.e. constraints) on the topic. For example, the user can specify the aircraft type to be "C-5", and the country to be "Tunisia". We call these *query constraints* on the query topic.

A *query aspect* of a query topic is a list of attributes (and their expressions) that are contained in the nodes of the query topic. A query aspect corresponds to the select list of an SQL query or the projection part of an algebraic expression. The query topic, constraints, and aspect together can be converted into an algebraic query expression.

2.2 Semi-automatic Generation of Semantic Graph

In current database systems, after a semantic model is converted into relations, it is no longer stored in the data model (i.e. the schema). Thus we have to use reverse engineering[12] to reconstruct the semantic model from a database schema. However, the mapping from a relational model to a semantic model, such as the extended entity-relationship (EER) model [13], is not one-to-one. Therefore, in general human intervention is needed to construct the semantic model from the corresponding database schema.

An initial semantic graph model can be automatically generated based on all the natural join links between relations. User-defined links can then be added. Other information, such as conceptual terms, can be assigned to the nodes and links.

A natural join between two relations usually represents a relationship through a key and a foreign key[1]. If the same names are used for attributes referring to the same domain, natural join links between relations in a database schema can be automatically generated based on the corresponding key attribute names. However, the same attribute name may refer to different domains, and two different attributes may refer to the same domain. To accommodate this situation, when checking if a key of a relation appears in another relation as a foreign key, we use domain names instead of attribute names. The domain names have to be provided by the designers.

To generate all the links with natural joins as their conditions, we find all the natural joins between pairs of nodes for a given array of relation nodes. The time complexity of the algorithm is $O(kmn^2)$, where k is the maximum number of keys in a relation, m is the maximum number of attributes in a relation, and n is the number of relations, including the duplicates.

For example, consider the following three relations:

```
AIRPORTS( APORT_NM, ELEV_FT, GEOLOC_TYPE,
  GLC_CD, HARDSTANDS, HNS_SORTIES,
  MAX_SORTIES, MILITARY_CIVILIAN_FLAG,
  PARKING_SQ_FT, POL_BBLS,
  SECONDARY_NM, STATUS_FLAG ),
KEY( APORT_NM )
RUNWAYS( APORT_NM, GLC_CD,
  RUNWAY_LENGTH_FT, RUNWAY_NM,
  RUNWAY_WIDTH_FT, SURFACE_FLAG ),
KEY( APORT_NM, RUNWAY_NM )
GEOLOC( CIVIL_AVIATION_CD, CY_CD,
  GLC_CD, GLC_LNCN, GLC_LOG_RGN_CD,
  GLC_LTCN, GLC_NM, GSA_CITY_CD,
  GSA_COUNTY_CD, GSA_STATE_CD,
  INSTLN_TYP_CD, LATITUDE, LONGITUDE,
  PRIME_GLC_CD,PROVINCE_CD,RECORD_OWNER_UIC),
KEY ( GLC_CD )
```

According to the key and foreign key relationships, we can find the following three natural join links:
(1) AIRPORTS and RUNWAYS, linked by APORT_NM;
(2) AIRPORTS and GEOLOC, linked by GLC_CD; and
(3) RUNWAYS and GEOLOC, linked by GLC_CD.
Since GLC_CD is a foreign key in both AIRPORTS and RUNWAYS, a link between these two relations by GLC_CD is not generated.

[1]To avoid an excessive number of links being generated, the links are limited to natural joins between a key and a foreign key and a link is not generated for a join between two foreign keys.

2.3 Information Measure of Nodes and Links

In the query formulation, multiple queries may be formed for a given user input. The *information* [14] of nodes and links is used for selecting and ranking query candidates.

The *information measure* for an element a (a node or a link) in a semantic graph measures the *information content* of a; that is,

$$I(a) = -\log P(a)$$

where log denotes the base 2 logarithm, and $P(a)$ is the probability of using a in queries.

The definition is consistent with that used in information theory which represents the number of bits needed to encode a node or link. The measure reflects the information content of a node or link. A smaller value of $I(a)$ means a larger $P(a)$, thus a will more likely appear in queries.

For simplicity in computing the information of a subgraph, we assume that all the nodes and links are independent. For a subgraph with a set of elements (nodes and links) $A = \{a_i | i = 1, \ldots, n\}$, the independence assumption implies that the information measure is additive, that is,

$$P(A) = P(\{a_i | i = 1, \ldots, n\}) = \prod_{i=1}^{n} P(a_i).$$

Thus

$$I(A) = -\log P(A) = -\log(\prod_{i=1}^{n} P(a_i)) = \sum_{i=1}^{n} I(a_i) \quad (1)$$

Information Measure Update The information measure for nodes and links can be computed from their relative frequency. Let c_i be the number of times that a_i is used in queries, and c be the total number for all the elements used in a set of queries, then

$$P(a_i) = \frac{c_i}{c} \quad (2)$$

The information measure of element a_i can be updated by the definition.

Initial Information Measure Assignment If a large collection of queries are available at the beginning, initial counting can be performed. But if the query set is not available or is too small to be statistically significant, we can assign an equal initial information measure to all the nodes and assign information measures to links based on the link types and their specificity. An example of link types and their sample information measures is shown in Table 1. Based on the semantics of the links, in general, the relationships among the information measures are $0 < I_1 \leq I_2 \leq I_3 \leq I_4 \leq I_5 \leq I_6$.

Link Type	*Information*
specific-entity ISA generic-entity	$I_1 = 1$
strong-entity HAS (PROPERTY) weak-entity	$I_2 = 2$
entity ROLEOF association	$I_3 = 2$
entity LINK entity	$I_4 = 3$
entity USER-DEFINED-ROLEOF association	$I_5 = 3$
entity USER-DEFINED-LINK entity	$I_6 = 4$

Table 1: Link types and their corresponding information measures in a semantic graph. The numbers are the sample information measures.

Once the initial information measures are assigned, they are normalized according to the probability property (summed to 1), and then converted into initial counts.

3 Formulation of Simple Queries from High-level Concepts

To formulate simple queries our query formulator only requires users to specify concepts, attributes, and values about a query. Based on these input, the system constructs the query via the semantic graph.

The query topic is the major source of complexity in formulating a query. The user input contains unconnected nodes and links. To formulate a query, we need to add links and nodes to connect the input subgraph into a query topic.

When the graph is cyclic, multiple links can be connected for the same set of nodes which can cause query ambiguity. We resolve this problem by (1) ranking the candidate queries based on their information measure, generating English-like query descriptions for the candidate queries to allow the user to select the desired one.

3.1 An Example

The user input consists of three parts: the query aspect which corresponds to the SELECT clause of an SQL query, the constraints, and special link requirements expressed in conceptual terms. The user interface then converts the user input into relations, attributes, and links. Consider the formulation of the query "*Find airports in Tunisia that can land a*

C-5 cargo plane." The input to the formulator is as follows:
(1) query aspect: `AIRPORTS.APORT_NM`;
(2) constraints:
`AIRCRAFT_AIRFIELD_CHARS.AC_TYPE_NAME = 'C-5'` and
`COUNTRY_STATE.CY_NM = 'TUNISIA'`;
(3) links: "`CAN LAND`".

The query formulator searches the transportation semantic graph (Figure 1) and adds missing links and nodes to complete a subgraph. A list of query candidates can be formulated. The following is the first query candidate:

```
SELECT  R3.APORT_NM
FROM    AIRCRAFT_AIRFIELD_CHARS R0,
        AIRPORTS R3, COUNTRY_STATE R10,
        GEOLOC R11, RUNWAYS R16
WHERE   R0.AC_TYPE_NAME = 'C-5'
        AND R10.CY_NM = 'TUNISIA'
        AND R0.WT_MIN_AVG_LAND_DIST_FT
          <= R16.RUNWAY_LENGTH_FT
        AND R0.WT_MIN_RUNWAY_WIDTH_FT
          <= R16.RUNWAY_WIDTH_FT
        AND R11.GLC_CD = R3.GLC_CD
        AND R3.APORT_NM = R16.APORT_NM
        AND R10.CY_CD = R11.CY_CD
```

3.2 Query Formulation as a Graph Search Problem

Given a user input for query formulation, we can process it into an incomplete query topic T_I, an aspect A, and a constraint set S. T_I contains the links specified in the user input, the nodes involved in the links, and the nodes in A and S.

Since T_I is usually not a connected subgraph, we need to choose additional links and relevant nodes from the semantic graph to extend T_I to form a connected subgraph for the query topic. We call these links and nodes a *query completion candidate* for T_I.

Property of a query completion candidate *Given a semantic graph $G = (V, E)$, to formulate a query from an incomplete input query topic $T_I = (V_I, E_I)$, where $V_I \subseteq V$ and $E_I \subseteq E$, is to find a query completion candidate $T_C = (V_C, E_C)$ for T_I such that query topic $T = T_I \cup T_C = (V_I \cup V_C, E_I \cup E_C)$ is a connected subgraph of G, where $V_C \subseteq V$, $E_C \subseteq E$, $V_C \cap V_I = \emptyset$, and $E_C \cap E_I = \emptyset$.*

That is, V_C is a set of nodes and E_C is a set of links needed to complete a connected subgraph to formulate a query. If the semantic graph is cyclic, there can exist more than one query completion candidate for the same input. We use the following minimum missing information principle for ranking the candidates.

Minimum Missing Information (MMI) Principle *The query completion candidate T_C (the missing links and nodes) for an incomplete input topic T_I contains the minimum information; i.e.* $\min I(T_C)$.

Based on equation 1, $I(T_C)$ can be computed from the information of the nodes and links as follows.

$$I(T_C) = \sum_{v \in V_C} I(v) + \sum_{e \in E_C} I(e) \qquad (3)$$

Thus the MMI principle provides us the measure for ranking the query completion candidates.

The smaller the $I(T_C)$, the more likely the completion candidate will meet the user's query intention. Links and nodes of smaller information have a higher probability of being used in queries, thus are more likely to be in the intended query. Due to the independence assumption, the probability value used in the ranking is an approximation. Therefore, we find a set of completion candidates and let the user select one.

Based on the MMI principle, the end points in the completed subgraph must be from the user input, since otherwise, deleting any end node that is not in the input will reduce the information of the completion candidate without affecting the connectivity. Thus, our MMI principle is consistent with the formulation of the query completion as the minimum Steiner tree problem [3]. According to [3, 15], query completion as a graph search problem is NP-complete.

3.3 Algorithm for Searching Query Completion Candidates

A user input contains a query aspect, a constraint set, and a link set. The required pre-processing on the input is to extract all the nodes that appear in the aspect and constraints. The nodes together with links from the input form the incomplete input query topic T_I for searching the query completion candidate T_C.

The incomplete input topic is decomposed into a set of connected *components*. The process of finding a completion candidate is to repeatedly connect two components via a path and form into a larger component. This merging process terminates when only a single component remains. We use the following two methods to limit the search scope:
(1) L-step-bound paths: paths that connect two components with at most l links. This confines the search to a small area surrounding the input subgraph.
(2) K-minimum completion candidates: only a maximum of k candidates with minimum weights are kept in the search process. This further trims the search within the set of candidate paths.

L-step-bound Paths L-step-bound paths are used to limit the search within the neighborhood of the input subgraph. This is based on the observation that each query topic will only span a small region of the semantic graph. L-step-bound paths eliminate the long paths for connecting components. L-step-bound paths are found through a breadth-first search. Paths with l links that have not yet reached a destination node are dropped.

The worst-case time complexity for finding l-step-bound paths with n components is $O(n^2md^l)$, where m is the maximum number of nodes in a component and d is the maximum number of links connected to a node.

K-minimum Completion Candidates During the search for completion candidates, α-β pruning is applied to trim the search branches. If a partial completion has a larger weight than the maximum weight of the current k-minimum completion candidates, this partial completion is excluded from further search.

For the current component list, the algorithm repeats the following process until no more paths can be tested:

1. Find l-step-bound paths for the current component list
2. For each path, merge the two components connected by the path
3. Check if only one component remains in the list
4. If yes, a completion candidate is found, and go test the next path
5. Otherwise, check if the current partial completion candidate is acceptable for the k-minimum completions;
6. If yes, go forward with the resultant component list;
7. If not, test the next path. If no more paths left, go back use previous component list (backtracking).

A larger l will be used if a completion candidate cannot be found for the current l.

The search problem is NP-complete. The worst-case time complexity can grow exponentially with the size of the graph (assuming $P \neq NP$). In most cases, however, the search will only traverse limited part of the graph due to the l-step-bound path heuristics, no matter how large the semantic graph is. Moreover, typical semantic graphs are sparse. Therefore, our formulation search algorithm is scalable.

For the example query in Section 3.1, we have the initial components and 2-step-bound paths as in Figure 3. We found 2-minimum completion candidates

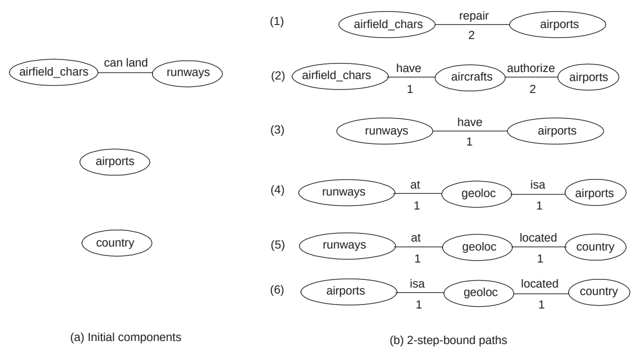

airfield_chars
can land
runways
airports
country
(a) Initial components
(1) airfield_chars repair 2 airports
(2) airfield_chars have 1 aircrafts authorize 2 airports
(3) runways have 1 airports
(4) runways at 1 geoloc isa 1 airports
(5) runways at 1 geoloc located 1 country
(6) airports isa 1 geoloc located 1 country
(b) 2-step-bound paths

Figure 3: Initial components and 2-step-bound paths for the "CAN LAND" query.

based on the 2-step-bound paths. The first contains paths (3) and (6) with an additional node GEOLOC, and the second contains paths (3) and (5) with the same additional node GEOLOC. After converting to SQL, the first completion condidate yields the query as shown in Section 3.1.

Correctness of the Query Completion Algorithm The k-minimum completion algorithm performs an exhaustive search with α-β pruning, which is equivalent to the exhaustive search. The query completion algorithm will find k-minimum completion candidates, as long as the algorithm for l-step-bound paths does not miss any paths that qualify in the k-minimum completion candidates.

Using l-minimum weight paths requires a depth-first search in the semantic graph which is not locally bounded. Therefore, l-step-bound paths are used which requires only a breadth-first search with a definite bound. For a reasonably large l, l-step-bound paths yield all possibly qualifying paths and ensure obtaining k-minimum completion candidates.

3.4 Generating English-like Query Descriptions for Disambiguation

Although sometimes some of the query candidates are equivalent, usually they yield different answers. Thus our system provides English-like query descriptions for the user to select one of the candidate queries that satisfies his query goal.

English-like query descriptions cannot be easily generated from SQL queries because of the lack of semantic information. However, the query representation based on the semantic graph can be used to provide such semantics. Entity nodes are translated into nouns, and links are translated into verbs. Nodes for complex associations including links can also be translated into verbs. Selection conditions on entity

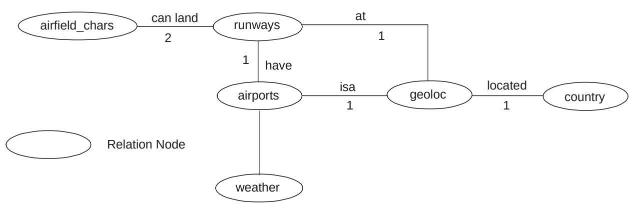


Figure 4: The semantic graph for the transportation domain.

nodes are translated into adjective phrases (modifiers on nouns), whereas conditions on association nodes are translated into adverbial phrases [16].

3.5 Capabilities of the Query Formulator

In addition to simple SPJ queries, queries containing aggregate functions can be formulated by generating a default `GROUP BY` clause which consists of all the non-aggregate function expressions in the SELECT list. Some of the cyclic queries can also be formulated by treating some special join conditions as selection conditions, although the query completion algorithm only searches for trees for query topics, which usually results in tree queries.

CoBase is a knowledge-based cooperative database system interface that run on top of relational databases. CoBase supports query relaxation to provide approximate query answers if an exact answer is not available [17]. The query formulator also supports cooperative operators, such as `SIMILAR-TO` and `NEAR-TO`, and relaxation control in the query language CoSQL [17, 18]. Using the above query formulation technique, we are able to formulate SPJ queries, with or without aggregate functions, as well as CoSQL queries.

4 Incremental Query Formulation for Complex Queries

Due to the difficulty in formulating complex queries directly, incremental formuation is introduced. The main issue in incremental querying is how to formulate a query at each step when derived relations are involved. To allow subsequent queries to use the results of preceding queries, the derived relations have to be incorporated into the semantic graph. In cases where a query yields an isolated node which is used in a later query, the system should be able to assist the user to find missing attributes for the query and link the result to other nodes. In the following, we first present the techniques to link derived relations into the graph and to find missing attributes for a query yielding an isolated node. Then, we give an example of incremental formulation in the transportation domain.

4.1 Support for Incremental Query Formulation

A *base relation* is a relation in the database schema and in the semantic graph at the beginning of a query session. A *derived relation* is a relation generated from a query during a query session. A relation used in a query whose attributes appear in a derived relation is called a *source relation* for the derived relation. A source relation for a derived relation may be a base relation or a derived relation. And there can be more than one source relation for a derived relation.

Incorporating A Derived Relation into the Semantic Graph For a derived relation to be used in subsequent queries, it has to be included into the semantic graph. As was done in the generation of the semantic graph, to link a derived relation to other relations, we need to distinguish key attributes from other attributes. Since an attribute in a derived relation is ultimately from some base relation except for complex expressions, it inherits the property in its base relation. We can differentiate a key from other attributes by the role it plays in the base relation, that is, if it is a key in the base relation.

The keys of a source relation included in a derived relation are the foreign keys for the derived relation. A derived relation R_d becomes a derived node in the semantic graph. For each source relation R_s, if its key attribute a_i is contained in R_s, a link with the key attribute as the equijoin attribute is generated to link R_d to its source relation R_s. A link between a derived relation and a source relation through the key attribute is called *deriving link*.

A derived relation may contain keys from several source relations, thus may have multiple deriving links to its source nodes. The deriving link has the semantics of `ISA`, so its information measure will be the same as the `ISA` links initially assigned. For the differentiation purpose, the links in the original semantic graph are called *base links*.

The deriving links found through key attributes suffice for the derived relation to be used in subsequent queries through the natural join. However, using these links to query other relations requires to navigate through the source relations from the derived relations. This may introduce unnecessary joins. To overcome this shortcoming, a derived relation can inherit links from its source relations, if the attributes involved in the links are contained in the derived relations. A link that is inherited from a source relations by a derived relation is called an *inherited link*. Inherited links will allow the derived relation to join with other relations directly. Weights on these links are also inherited from their corresponding links.

The algorithm of finding inherited links first extracts source relation nodes for the derived relation. For each source node, it then checks each link related to the source node to see if it can be inherited by checking if the attributes involved in the link are also contained in the derived relation. Note that the attributes involved in the inherited links are not necessarily keys. Links with non-key attributes, usually user-defined links, can also be inherited.

Suggesting Key Attributes for a Query Since users cannot foresee all the attributes/expressions and conditions they need for a future query, a query formulation attempt may fail when an isolated derived node is involved. This happens because the derived relation does not contain a key attribute, thus cannot be linked to its source nodes, and it has no necessary attributes to inherit other links. The query tool can detect an isolated node and suggest possible key attributes to include in the corresponding query, and then connect its node to one of its source nodes.

The suggestion is based on the information returned from a failed query formulation that includes the maximal components the formulator has reached. For an isolated node, if any of its source nodes appears in one of the components, the corresponding key is a candidate to be included in the query for the isolated node. Otherwise (no source nodes in components), the candidate can be obtained by searching for a closest source node to the nodes in the components.

For some derived relations, there are no key attributes from their source relations, and no key attributes can be added into the query due to semantic restriction, such as aggregate functions. For such cases, the user has to specify certain attributes to act as keys (called *acting keys*) if the derived relations have to be linked to other relations.

4.2 Capability of Incremental Query Formulation

In addition to the SPJ queries at each step, set operations can be easily supported, which do not require the query formulator. The `HAVING` clause of an SQL query can also be simulated by separate queries. Composite queries can be generated for most query sessions without execution of the simple queries. The expressiveness of incremental query answering is beyond relational completeness. Because we allow iterations in a session, transitive closures can be computed.

4.3 An Example for Incremental Query Formulation

We illustrate the incremental query formulation process by the following example with a series of three queries in a transportation domain, which shows queries whose answers depend on the answer set of the query from the previous step. The queries are presented in English as the following:

1. "Find airports in Tunisia."
2. "Which of these airports can land a C-5?"
3. "What is the weather at these airports?"

The user input for the first step is

```
// select list:
AIRPORTS.APORT_NM
// constraints:
COUNTRY.COUNTRY_NM = 'Tunisia'
// links: none
// relation name:
AIRPORTTUNISIA
```

The first query candidate generated by the formulator listed below is correct. The resultant derived relation is named `AIRPORTTUNISIA`, and is added to the semantic graph.

```
SELECT  R3.APORT_NM
FROM    AIRPORTS R3, GEOLOC R11, COUNTRY R10
WHERE   R10.COUNTRY_NM = 'Tunisia'
        AND R11.CY_CD = R10.CY_CD
        AND R3.GLC_CD = R11.GLC_CD
```

The user input for the second step is given as follows:

```
// select list:
AIRPORTTUNISIA.APORT_NM
// constraints:
AIRFIELD_CHARS.AC_TYPE_NAME = 'C-5'
// links:
CAN LAND
// relation name:
AIRPORTTUNISIACANLAND
```

The formulator will generate the following query and the answers to this query will be used when creating the derived node `AIRPORTTUNISIACANLAND`:

```
SELECT  R3.APORT_NM
FROM    RUNWAYS R16, AIRFIELD_CHARS R0,
        AIRPORTTUNISIA R20
WHERE   R0.AC_TYPE_NAME = 'C-5'
        AND R0.WT_MIN_AVG_LAND_DIST_FT <=
        R16.RUNWAY_LENGTH_FT
        AND R0.WT_MIN_RUNWAY_WIDTH_FT <=
        R16.RUNWAY_WIDTH_FT
        AND R16.APORT_NM = R20.APORT_NM
```

The third user input is as follows:

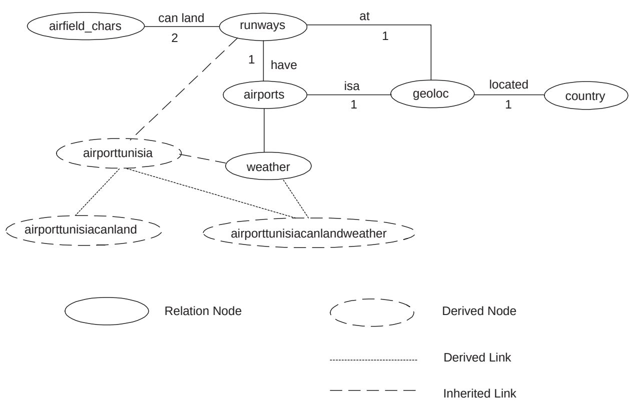


Figure 5: Extended semantic graph showing derived nodes, derived links and inherited links.

```
// select list:
WEATHER.CONDITION
// constraints: none
// links: none
// relation name:
AIRPORTTUNISIACANLANDWEATHER
```

The resulting formulated query is given below. The system will also add the derived node `AIRPORTTUNISIACANLANDWEATHER` to the semantic graph.

```
SELECT R15.CONDITION
FROM   WEATHER R15,
       AIRPORTTUNISIACANLAND R21
WHERE R15.APORT_NM = R21.APORT_NM
```

This example shows that the incremental query formulation provides a feasible and effective way to formulate complex queries. A semantic link was used in the second step of the incremental session and was translated into a set of conditions.

Figure 5 shows the semantic graph of figure 4 with the three derived nodes from the example. Notice that `AIRPORTS` has links to `RUNWAYS` and `WEATHER` with the link attribute `APORT_NM`, which is contained in the derived relation `AIRPORTTUNISIA`. Therefore, `AIRPORTTUNISIA` inherits these two links from the source node `AIRPORTS`. The inherited link from `AIRPORTTUNISIA` to `RUNWAYS` is used in the second step. The inherited links are shown in Figure 5 as dashed lines. To avoid clutter, only the inherited links for the derived node `AIRPORTTUNISIA` are shown.

5 Implementation and Experience

A database interface system using the above high-level query formulation techniques has been implemented at UCLA. The user chooses the set of semantic graph nodes, links and attributes as an incomplete query topic. The select list and the query constraints are also specified by the user. This information is used in the query formulation process, which uses the graph search techniques described in this paper to find a path through the semantic graph. The formulated query is given to CoBase for processing and the answers returned are displayed to the user.

Figures 6 to 9 show the user input for formulating the query, "Find an airport in Tunisia that can land a C-5.". Figures 6 and 9 allow the user to specify the concepts, attributes and constraints and Figure 7 allows the user to choose a high-level action (e.g. "CAN LAND" was chosen as the action). Figure 8 shows the formulated query and an English-like description of the query.

The query formulator was implemented in Java with three packages for the graph model, the query repre-

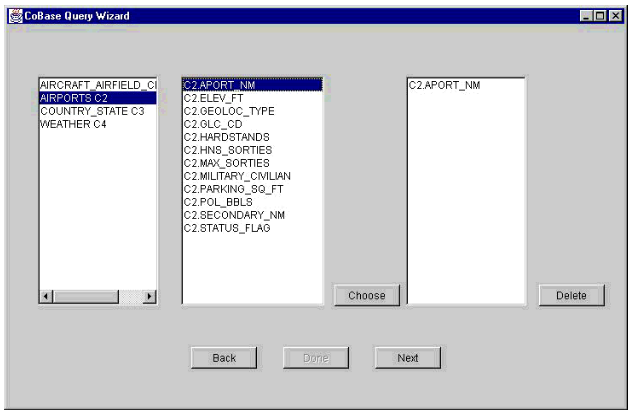


Figure 6: Concept and Attribute Specification Interface

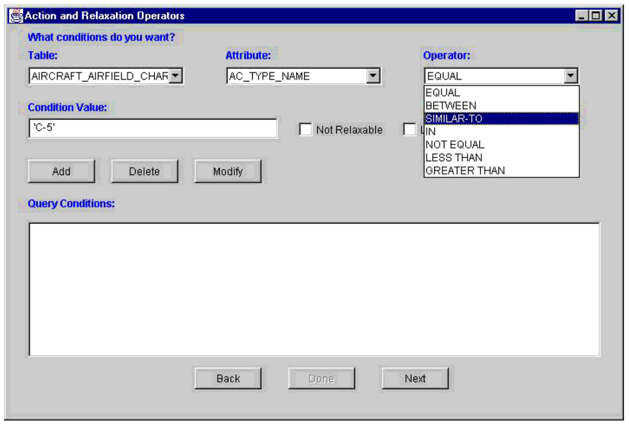


Figure 7: Query Constraint Specification

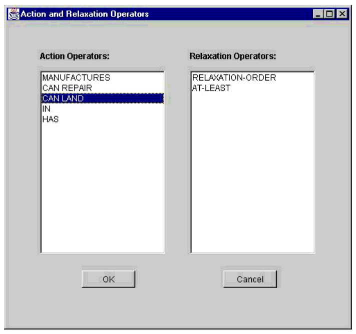
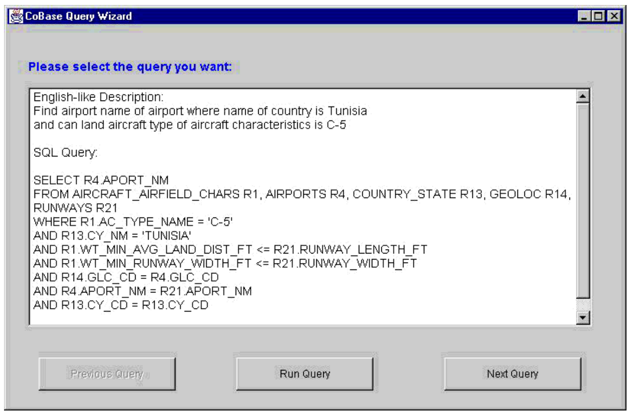


Figure 8: Action Specification

sentation, and the formulator, with a total of about 50 classes and 7,000 lines of code. The incremental support adds about 20 classes and 3,000 lines of code. The system was tested on both the Windows NT and Solaris platform using an Oracle database. The query formulator is JDBC compliant, therefore it can be ported to multiple platforms and databases.

The query formulator was tested on a transportation database provided by DARPA. There are about 300 relations and the largest relation has more than 50,000 tuples. The semantic graph was automatically generated from the database schema and a domain expert labelled some of the links with high-level concepts (e.g. 'CAN LAND'). Queries that involve up to 7 relations and contain multiple query constraints have been successfully formulated by the query formulator. CoSQL queries that involve coopertive operators like SIMILAR-TO, APPROXIMATE, RELAX-ORDER and NOT-RELAXABLE were also correctly formulated. To test its extensibility and portability, we also tested the query formulator on a different domain, a logistics database. The only effort required to port the query formulator to the logistics domain is the generation of the semantic graph and the labelling of the links by a domain expert. It is worth noting that information measures update provides good ranking result and adaptive behavior for the system. In many cases, the correct query will appear at the top among initially equally-ranked query candidates after a right user feedback.

Performance tests of accuracy and search time were conducted on the transportation database schema on a Sun Ultrasparc II machine. The semantic graph for part of the transportation database contains 50 nodes and 105 links. For 30 random query samples, all the query candidates ranked first are correct. The search time required to formulate the query ranges from 84 milliseconds to 871 milliseconds, with an average time of 323 milliseconds, which is well below the query ex-



```
English-like Description:
Find airport name of airport where name of country is Tunisia
and can land aircraft type of aircraft characteristics is C-5

SQL Query:

SELECT R4.APORT_NM
FROM AIRCRAFT_AIRFIELD_CHARS R1, AIRPORTS R4, COUNTRY_STATE R13, GEOLOC R14,
RUNWAYS R21
WHERE R1.AC_TYPE_NAME = 'C-5'
AND R13.CY_NM = 'TUNISIA'
AND R1.WT_MIN_AVG_LAND_DIST_FT <= R21.RUNWAY_LENGTH_FT
AND R1.WT_MIN_RUNWAY_WIDTH_FT <= R21.RUNWAY_WIDTH_FT
AND R14.GLC_CD = R4.GLC_CD
AND R4.APORT_NM = R21.APORT_NM
AND R13.CY_CD = R13.CY_CD
```

Figure 9: English-like Query Description and the Formulated Query

ecution time.

The query formulator provides a set of APIs that allow other applications to easily interface with it. We have integrated the query formulator with a point-and-click user interface, a map interface and a voice input interface [19]. We have implemented an interface with IBM's ViaVoice and which is capable of formulating, from voice input, a test set of simple SQL queries in the transportation domain. The system is currently operating on top of a cooperative database (CoBase) at UCLA to formulate SQL queries.

6 Conclusion

A new query formulation system based on a semantic graph model is presented. Query formulation as a graph search uses probabilistic information measure in the search process and to rank query candidates. When multiple queries are formulated from a user input, the user can resolve such query ambiguity based on ranking and English-like query descriptions. To limit the search scope in finding the subgraph from the semantic graph, a heuristic algorithm was employed to reduce the search complexities. The query formulator algorithm can formulate the SELECT-PROJECT-JOIN queries with aggregate functions as well as CoSQL queries.

To formulate complex queries, a new incremental approach is presented. A derived relation as the result of a query is linked to its source nodes by the key attributes from the corresponding source relations. A derived relation can also be linked to other nodes by inherited links from its source relations. If there are no linking attributes present, the system should provide information for the user to add the link. The user can formulate queries based on the results of preceding queries, or compose a complex query without executing query steps. Experiments show the technique is effective in formulating complex queries.

We have constructed a prototype system using the above technique with point-and-click interface. We have also developed a method to extract and disambiguate keywords (relations, attributes, constraints) from English-like sentences. Thus, keywords can be input to the high-level query formulator to generate the underlying SQL query. Our experience has shown that the technology is also very suitable for formulating SQL queries from English-like query input.

Acknowledgment

We would like to thank David Wu for his assistance in implementation of the system.

References

[1] Jeffrey D. Ullman. *Principles of database and knowledge-base systems, Vol.II.* Computer Science Press, 1988.

[2] Moshe Y. Vardi. The universal-relation data model for logical independence. *IEEE Software*, pages 80–85, March 1988.

[3] Joseph A. Wald and Paul G. Sorenson. Resolving the query inference problem using steiner trees. *ACM Transactions on Database Systems*, 9(3):348–368, September 1984.

[4] E. F. Codd. 'universal' relation fails to replace relational model. *IEEE Software*, June 1988.

[5] Yannis E. Ioannidis and Yezdi Lashkari. Incomplete path expressions and their disambiguation. In *Proc of SIGMOD'94*, pages 138–149, 1994.

[6] Amihai Motro. A trio of database user interfaces for handling vague retrieval requests. *IEEE Data Engineering Bulletin*, 12(2), 1989.

[7] A. D'Atri, P. Di Felice, and M. Moscarini. Dynamic query interpretation in relational databases. *Information Systems*, 14(3):195–204, 1989.

[8] I. Androutsopoulos, G. D. Richie, and P. Thanisch. Natural language interfaces to databases - an introduction. *Journal of Natural Language Engineering*, 1994.

[9] C. Raymond Perrault and Barbara J. Grosz. Natural-language interfaces. *Annual Review of Computer Science*, pages 47–82, 1986.

[10] Antonio Massari and Panos K. Chrysanthis. Visual query of completely encapsulated objects. In *Proc of the 5th Intl Workshop on Research Issues on Data Engineering*, pages 18–25, 1995.

[11] S.-K. Chang, M. F. Costabile, and S. Levialdi. Reality bites - progressive querying and result visualization in logical and vr spaces. In *Proceedings of IEEE Symposium on Visual Languages*, pages 100–109, St. Louis, MO, USA, October 1994.

[12] Roger H. L. Chiang, Terence M. Barron, and Veda C. Storey. Reverse engineering of relational databases: Extraction of an eer model from a relational database. *Data & Knowledge Engineering*, 12:107–142, 1994.

[13] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197–222, June 1986.

[14] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[15] F. K. Hwang and Dana S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[16] Peter P. Chen. English sentence structure and entity-relationship diagrams. *Information Sciences*, 29:127–149, 1983.

[17] Wesley W. Chu, M. A. Merzbacher, and L. Berkovich. The design and implementation of CoBase. In *Proc of ACM SIGMOD 93*, pages 517–522, Washington D. C., 1993.

[18] Wesley W. Chu, Hua Yang, Kuorong Chiang, Michael Minock, Gladys Chow, and Chris Larson. Cobase: A scalable and extensible cooperative information system. *Journal of Intelligent Information Systems*, 1996.

[19] Frank Meng and Wesley W. Chu. Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation. Technical Report CSD-TR 990003, University of California, Los Angeles, 1999.