Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation

Frank Meng and Wesley W. Chu
Computer Science Department
University of California, Los Angeles, CA 90095, USA
{fmeng, wwc}@cs.ucla.edu

CSD-TR 990003

Abstract

This paper describes a natural language interface to database systems which is based on the query formation capabilities of a High-level Query Formulator. The formulator relies on the Semantic Graph of the database, which is a model of the data stored in the database. The natural language interface accepts a user input in natural language and extracts the necessary information needed by the formulator. This extraction process is performed using keywords obtained from the Semantic Graph and the database. Because keywords may have several meanings within a given domain, keyword meaning disambiguation is done using a statistical approach which involves comparing vectors of n-grams. N-grams are n contiguous words within a given text of natural language and they are capable of capturing lexical context. Traditionally, natural language interfaces have been heavy with grammars and other knowledge, but have been wide-ranging in functionality. The interface presented in this paper is more portable and flexible in comparison, but sacrifices some functionality. We feel such high-level interfaces to information systems will be needed as information systems become more readily available to more users.

1 Introduction

This paper will describe a natural language interface to database systems as an extension of an existing High-level Query Formulator for SQL queries [12]. The High-level Query Formulator uses the Semantic Graph of a relational database, which is a graph representation of the semantic model of the underlying database to formulate valid SQL queries from incomplete information from the user. This incomplete information takes the form of tables, attributes, database values, and other associated information which reflects the user's intended query. Because the casual user will have little knowledge of the query language and the schema of the underlying database, there is certain information which will be hard for the user to supply to the system. We envision the interface of the High-level Query Formulator to be a point-and-click graphical interface which displays the Semantic Graph of the database and allows the user to choose the tables, attributes and other information which will participate in the final query. This paper will describe a natural language interface which will allow users to supply query information to the formulator using

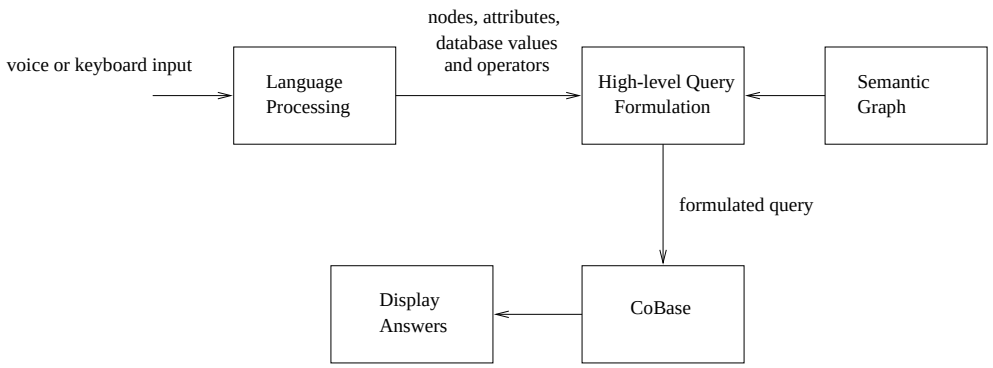

Figure 1: System Diagram

natural language inputs. The goal of our natural language processing system, similar to other natural language interfaces to databases, will be to extract the necessary information from the user's natural language input and feed this information to the formulator.

Processing natural language to extract query information will be done using keywords and statistical keyword disambiguation using a vector similarity measure. We define keywords to be words or phrases that hold particular meaning within the domain. In our system, keywords will come from the Semantic Graph and the database. If the system can determine every keyword and their corresponding meaning from the user input, then the meaning of the user input can be determined by the meaning of the keywords. Determining the meaning of a given keyword is a non-trivial problem, since it is possible for a keyword to have multiple meanings within a domain due to the ambiguous nature of natural language. We propose to use vectors of n-grams to help disambiguate keyword meaning. N-grams are n contiguous words in a natural language text, and vectors of n-grams are able to summarize the lexical context in which a keyword is used. By measuring the similarity between the lexical context of the user input with the lexical contexts of each possible meaning of a keyword, the system can determine how likely a keyword corresponds to a particular meaning.

Figure 1 shows a diagram of the system. The natural language processing component takes text from keyboard input or speech input and extracts the necessary information for formulating a database query. The High-level Query Formulator takes this information and generates a database query using the Semantic Graph. The query is sent to CoBase [5, 6], a cooperative database system, or any relational database and answers are returned and displayed to the user.

Linking a semantic model of the database with natural language is discussed in [4]. Much work has been done in statistical methods for language understanding [3], which focuses on using probabilities computed from a large corpus. Many natural language interfaces to database systems are based on syntactic grammars or semantic grammars and some examples are described in [11, 7, 9, 2]. Pattern matching systems [10] have been proposed as an alternative to grammar-

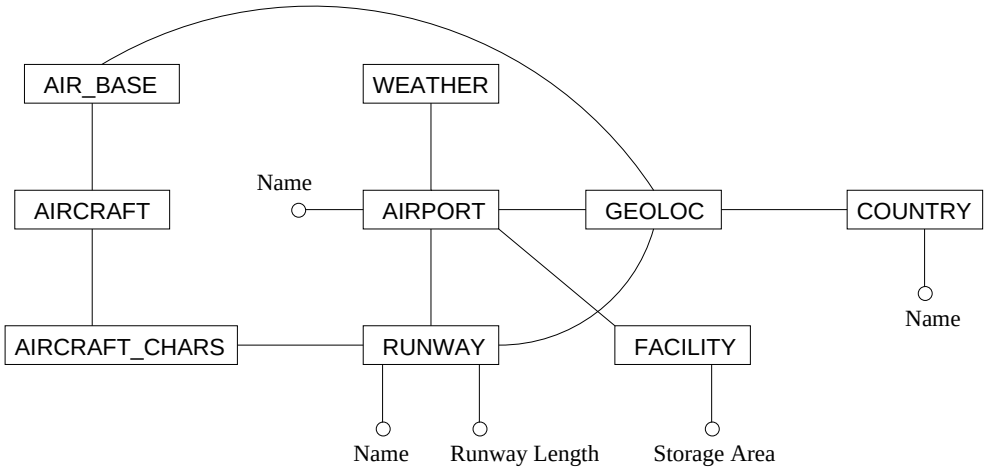

Figure 2: Example Semantic Graph for a Transportation Database

based systems. These systems often use templates to match certain words and phrases.

Our method for processing natural language does not rely on any grammars and uses little outside knowledge and thus carries less overhead. This comes at the cost of having to gather statistics which are specific to the domain. We feel our system fits well into applications which cannot carry much overhead in terms of outside sources of knowledge (grammars, semantic networks, etc.), and have domains which have limited ontologies. The world wide web is an example of such an application, where many high-level interfaces can be used over many domains with a limited ontology.

2 The Semantic Graph and High-level Query Formulation

Our research group has developed a method for automatically formulating a query (in SQL) based on incomplete information from the user [12]. The formulation is done utilizing a Semantic Graph of the database which describes the objects and relationships between objects modeled in the database. The Semantic Graph consists of three elements: nodes, attributes, and links. The nodes are the objects in the world, the attributes are the features of the nodes, and the links describe how the nodes are related to each other.

Figure 2 shows a portion of the Semantic Graph of a transportation database. The nodes of a Semantic Graph are represented by rectangles and attributes are represented by small circles. The links are represented by directionless lines between two nodes or between a node and an attribute. The Semantic Graph is used as the basis for query formulation, where the user input consists of incomplete query information, and the job of the formulator is to generate a valid database query based on what was given to the system by the user. The High-level Query Formulator takes incomplete information from the user in the form of nodes, attributes and/or links

from the Semantic Graph and, using graph search techniques, generates a valid query. We define the Formulator Input as a set of three things: an *incomplete query topic*, a *select list*, and a set of *query constraints*. An *incomplete query topic* is a set of subgraphs from the Semantic Graph which have been chosen by the user to be components of the final query. The incomplete query topic represents the nodes, attributes and links that the user wants as part of the query. The High-level Query Formulator completes an incomplete query topic by finding a set of links and nodes which can connect all the components. The final connected subgraph represents a specific topic of a query and thus is called the *query topic* (see figure 3). Since the user may not be familiar with the database schema, information may be missing, such as how the chosen nodes and links can be connected or joined. The High-level Query Formulator of our system is capable of supplying the join conditions for the user based on the Semantic Graph elements that were chosen for the query. The *select list* is the information which the user wants the system to retrieve as the answer to the query (the SELECT list of an SQL query). The *query constraints* are the conditions of the query which impose restrictions on what answers should be retrieved from the database (the WHERE conditions of an SQL query). The three components of the Formulator Input are used by the formulator to produce a valid database query. The subsequent sections of this paper will discuss how the Formulator Input can be extracted from a user input in natural language.

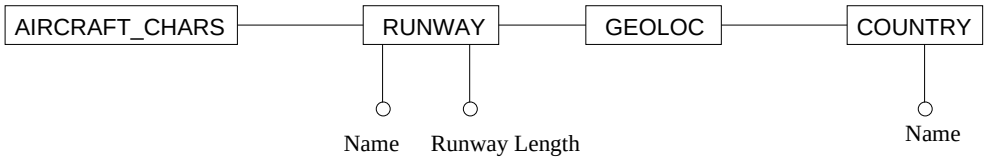

Figure 3: Query Topic for Finding Suitable Runways for Aircrafts

3 Natural Language Processing

In order to process natural language, we use a subset of words or phrases which hold particular meaning within the database domain. These words or phrases are what we define as *keywords*. Given a user input as a sequence of words $I = (w_1, w_2, ..., w_n)$, we can approximate the meaning of I by the meaning of the keywords present in the sentence. We perform natural language processing in two steps. In the first step, given the user input I, the system will determine all the keywords contained in I. In the second step, the system will use a similarity comparison between n-gram vectors to determine the most likely meaning of each keyword contained in I. In our system, the keywords indicate what nodes, attributes, database values, operators, and other information are mentioned by the user. Once the set of keywords that occur in I have been determined, the set of possible meanings for each keyword is enumerated based on how the keywords are used in

the Semantic Graph and whether the keywords are used as database values. For instance, the keyword "condition" may refer to either the *condition* of a runway or the weather *condition* of an airport in our transportation database example.

In order to determine the most likely meaning of a given keyword within the lexical context of the user input I, statistical methods in the form of n-gram vector comparisons for keyword meaning disambiguation are applied. Since it is possible that a given keyword may refer to several possible meanings, the surrounding words of context are used to help disambiguate the meaning of the keyword. We use the cosine of the angle between two n-gram vectors as a similarity measure [8] because it has been shown to be both simple and effective.

The following sections will describe in more detail how keywords are acquired and used, and how n-gram vectors are generated and compared.

3.1 Keywords

For query formulation, the words or phrases that take on particular meaning are those words which will participate in the formulation process of the query. For our system, the keywords will be those words and/or phrases which denote the nodes and attributes of the Semantic Graph, database values, or operators. These four "types" will represent the four keyword categories in our system. The difficulty of processing natural language input is that user intentions are expressed in a potentially ambiguous and indirect manner. In a point-and-click interface, the user's intentions are clear because the user explicitly chooses which elements will participate in the final query. With natural language, the system must extract the necessary information from the user input, which will denote what nodes, attributes, database values, or operators will participate in the final query.

Nodes and attributes of the Semantic Graph are uniquely identified by their internal label which is taken from the database schema. We choose not to use these internal labels as keywords because they are usually non-descriptive and not written in natural language form. The Semantic Graph allows the domain expert to attach multiple natural language labels to any given node or attribute which can be more descriptive and more naturally used. These natural language labels will be the keywords in our system which denote their corresponding node or attribute. For instance, in our example transportation database, the attribute with an internal label, "runway_length_ft" will have the natural language labels "runway length" and "length of runway". The system uses simple pointers to keep track of which natural language labels correspond to which node or attribute.

Database values are used in the constraints of the query (WHERE conditions) and must be mentioned by the user's natural language input if it is to be used in the final query. In a relational database, each attribute of each table has a set of database values. These values not

only determine the possible values each attribute can take on, but also impose constraints on what values are valid for each attribute. Each database value for each attribute will be used by the system as a keyword. Since database values should be known to the user who is querying the system, we do not feel there is a need to attach natural language labels to database values. Simple pointers are used to keep track of which database values can correspond to which attributes.

Comparator operators are used in the query constraints for comparing database values with constant values. Each operator will have associated with it a set of natural language labels (e.g. "greater than" for the operator ">"). Operators are more difficult to handle than nodes, attributes, or database values because there are many natural language labels which denote a given operator. For instance, the ">" operator, denoting "greater than" can have many labels depending on the context in which it is used. If lengths are being compared, then ">" may be denoted as "longer than", and if height is being compared, then it may be "higher than" or "taller than". One way to resolve this problem is to use a richer set of semantic data types for each numeric attribute (attributes with unordered values can only use the equals or not-equals operators). Some common semantic types may be *length*, *width*, or *height*. Associated with each semantic type would be a set of natural language labels for the standard operators including ">", "<", "=", "<=", ">=", "!=", etc. In our example transportation database, the attribute *runway_length_ft* may have the semantic data type *length*.

The other, more simpler, solution is to force the user into using the standard names for each operator (e,g, "greater than", "less than" "equal to"). How the system is designed in the end will be a tradeoff between low overhead (more portability) versus flexibility (easier for the user to use).

Note that a given natural language label can correspond to more than one category of keywords. For instance, the keyword "runway" may denote the node *runway* or the attribute *runway* of the node *airport*. For efficiency, the system will only keep one copy of a unique natural language label and keep multiple pointers to the nodes, attributes, database values, or operators that it may correspond to. Each node, attribute, database value, and operator that corresponds to a given keyword will be part of the keyword's formal meaning, which will be discussed later in this section.

As an example of keywords in our transportation database domain, the user may ask the database, "What is the weather condition of Bizerte airport?" From looking at figure 2 and knowing the values contained in the database, this question has the keywords *weather*, *condition*, *Bizerte*, and *airport*. *Weather* and *airport* are node names, *condition* is an attribute name of the node *weather*, and *Bizerte* is a database value. In this example, the *incomplete query topic* of the Formulator Input will consist of the nodes *weather* and *airport* and the attribute *condition*. The *select list* will be determined by what attributes are mentioned by the user input, but are not

used in the query constraints (as described in section 3.3). The *query constraints* are determined using the database value *Bizerte*, which would result in the query condition *aport_nm = 'Bizerte'*. A Formulator Input will be generated from these three parts and will be sent to the High-level Query Formulator, where a formal database query is produced.

Determining the meaning of keywords is not always as trivial as in the example above, because a given keyword may have one of four possible meanings: a node, an attribute, a database value or an operator. More formally, we define the function *category(k)* $\in$ {node, attribute, database value, operator}, which returns the category of the keyword k. The formal definition of a meaning m of k is a pair $(category(k), e)$, where e is a particular node or attribute in the Semantic Graph. Since k could have several possible meanings, the *meaning set* of k is defined as follows.

$$meaning\_set(k) = \{(\text{node}, n) | n \in N \& k \in labels(n)\} \cup$$

$$\{(\text{attribute}, a) | a \in A \& k \in labels(a)\} \cup$$

$$\{(\text{database value}, a) | a \in A \& k \in values(a)\} \cup$$

$$\{(\text{operator}, a) | a \in A \& k \in operators(a)\}$$

In this equation, N is the set of nodes in the Semantic Graph, A is the set of attributes in the Semantic Graph, $labels(e)$ returns the set of natural language labels of a Semantic Graph element e (node or attribute), $values(a)$ returns the set of database values of an attribute a, and $operators(a)$ returns the set of operators which can correspond to attribute a. The function $operators(a)$ is needed because not every operator is valid for a given attribute a. For instance, if a corresponds to an attribute having unordered values, then the operator ">" will not be valid.

The above equation just says that the set of possible meanings for a keyword k (k's *meaning set*) is the union of all the nodes which can have k as a natural language label, all the attributes which can have k as a natural language label, all the database values which take on the value k, and all the operators which can be described as k. If a keyword category is *node*, then the meaning is just the node having the keyword as one of its natural language labels. Similarly, a meaning with category *attribute* is just the attribute with the keyword as one of its natural language labels. For the *database value* category, the keyword is the database value, and the meaning is just the attribute it corresponds to in the query's condition. For instance, the keyword "Bizerte" is a database value which corresponds to the attribute *country_nm* if the final query has the condition *country_nm = 'Bizerte'*. For the category *operator*, the keyword represents the operator, and the meaning is just the attribute it corresponds to in the query condition. For instance, the keyword "less than" may correspond to the attribute *runway_length_ft* if the final query has the condition *runway_length_ft < 5000*.

Using the *meaning_set()* function, the system can determine a set of keyword-meaning pairs KM of the user input I which corresponds to all the keywords mentioned in I and all their possible meanings. The set KM will have the form $\{(k_1, m_{1,1}), (k_1, m_{1,2}), (k_1, m_{1,j}), ..., (k_n, m_{n,1}), (k_n, m_{n,m})\}$. k_i refers to a keyword in the user input I and $m_{i,j}$ refers to the jth meaning of keyword k_i.

As an example, consider the keyword "name" in the domain of the Semantic Graph shown in figure 2. The keyword "name" can correspond to either the name of an airport, the name of a runway or the name of a country. Thus, the *meaning set* of the keyword "name" would be *meaning_set("name")* = {(attribute, *airports.aport_nm*), (attribute, *runways.runway_nm*), (attribute, *country.country_nm*)}. The keyword "name" happens not to correspond to any nodes, database values, or operators in this domain, so this would be the full meaning set of "name".

Processing keywords determines the meaning set for each keyword in the user's question. However, to fully extract the information necessary to construct a Formulator Input, the system needs a way of determining the "best" meaning among all possible meanings in the meaning set. This will be done using a vectory similarity measure which will be described in the next section.

3.2 Using N-Gram Vectors for Capturing Lexical Context

N-grams can be used to provide a lexical context in which we can measure how likely a given keyword has a particular meaning in a given user input. What a keyword means depends upon the lexical context in which it is used, which is just the words surrounding it. An n-gram is a set of n contiguous words in a given section of natural language text. n can range from 1 to the length of the text. For the sentence, "The black cat slept", the n-grams for $n = 2$ are (the black), (black cat), and (cat slept). For $n = 3$, they are (the black cat) and (black cat slept).

We will use n-grams as the basis for our statistical keyword meaning disambiguation process. Given a set of meanings of a particular keyword, we would like to use the n-grams in the user input to find the meaning that best matches the keyword and the context that it is in. Given the user input as an ordered list of natural language words $I = (w_1, w_2, ..., w_m)$, a set of n-grams can be generated. In practice, we will use n-grams where $n = 3$ (trigrams) because our early empirical results have shown that they capture enough lexical context without degrading system performance. The vector of trigrams of the user input I is $\vec{V}_I = [(w_1, w_2, w_3), (w_2, w_3, w_4), ..., (w_{m-2}, w_{m-1}, w_m)]$.

Every possible meaning m will have a vector $\vec{V}_m$ in n-gram space associated with it. Given the set of keyword-meaning pairs KM extracted from the user input I, for each $(k, m) \in KM$, the system can determine a ranking measure for the meaning m of k by measuring the similiarity between the n-gram vector for I, $\vec{V}_I$, and the n-gram vector for m, $\vec{V}_m$. The meaning m which has the vector $\vec{V}_m$ with the largest cosine measure with $\vec{V}_I$ is ranked the highest. The cosine of the angle between two vectors is a standard similarity measure in the information retrieval field [8] and has been shown to be effective yet simple.

Continuing with our example from the previous section, to determine the most likely meaning of the keyword "name" based on past experiences, the system would first retrieve the n-gram vectors of each possible meaning of "name". In this case, "name" has three possible meanings and there will be three vectors associated with each meaning. For the meaning referring to attribute *airports.airport_nm*, there is an associated vector $\vec{V}_{airport\_nm}$. Similarly there are vectors $\vec{V}_{runway\_nm}$ and $\vec{V}_{country\_nm}$ for the attributes *runways.runway_nm* and *country.country_nm* respectively. To determine the most likely meaning for the keyword "name" within the context of the user input, "what is the *name* of the airports with runway length of 5000 feet?", the vector of n-grams for the user input I, $\vec{V}_I$, is generated. The cosine of the angle between $\vec{V}_I$ and each of the vectors $\vec{V}_{airport\_nm}$, $\vec{V}_{runway\_nm}$, $\vec{V}_{country\_nm}$ is calculated. The meaning m with the vector $\vec{V}_m$ that has the highest cosine value with $\vec{V}_I$ will be chosen by the system to be the meaning of the keyword k within the context of I.

There has been other work in computational linguistics and statistical language understanding which focuses on word sense disambiguation within a large corpus [3, 1]. The keyword meaning disambiguation in a database domain differs somewhat from general word sense disambiguation problems. In our case, each user input is short and is usually not connected to any discourse context. This results in very little global context to consider and very little surrounding context words to go by. Thus, in order to disambiguate keywords at a reasonable accuracy rate, the system needs to take groups of related words into consideration. Since we do not perform syntactic parsing, the word phrases and syntactic structure come from grouping contiguous words within the user input, which is the function of n-grams.

One standard method for word sense disambiguation is to use word collocations, which measures the likelihood of two words appearing in the same window of words. This method may not perform too well with the types of user inputs we are trying to handle because the keyword meanings are harder to distinguish. For instance, the question Q_1 = "are there flights from Los Angeles to New York?", has the keywords *Los Angeles* and *New York*. In Q_1, *Los Angeles* is the departure city and *New York* is the arrival city. We also may have a question Q_2 = "are there flights from New York to Los Angeles". In Q_2, *New York* is the departure city and *Los Angeles* is the arrival city. If a large window of surrounding words was used to determine the meaning of the keywords, it would be difficult to determine the difference between Q_1 and Q_2, since these windows may look very similar. If smaller windows are used, the lexical context may be too specific. However, if n-gram vectors are used, we can see that the trigrams *(from Los Angeles)* and *(to New York)* are unique to Q_1 and the trigrams *(from New York)* and *(to Los Angeles)* are unique to Q_2. N-grams seem to address some of the issues we are concerned with and we are in the process of performing empirical experiments to determine the effectiveness of using n-grams as opposed to other techniques such as word or phrasal collocation probabilities.

3.3 Determining the Select List

The select list of a Formulator Input is dependent upon what the user wants returned as an answer. Our system handles two types of inputs: ones with explicit select lists and ones with implicit select lists. Inputs with explicit select lists are ones in which the user explicitly specifies which attributes are to be returned by the system. Examples of such inputs include "What is the weather condition of Bizerte airport?" and "Get me the runway length and runway width of all airports in Tunisia." Inputs which have implicit select lists include "Find me an airport in Tunisia with sunny weather" and "Are there any airports that can land a C-5?"

For inputs with explicit select lists, the system uses a heuristic for determining what attributes belong in the select list. The system keeps a list of all attributes explicitly mentioned by name in the input. Those attributes that do not participate in the query constraints are automatically put into the select list. The assumption is that attributes used in the query constraints already have values, and those not in the query constraints need their values retrieved.

Inputs with implicit select lists are more difficult to handle because it is not clear what information is expected by the user. Each node in the Semantic Graph has a key attribute (similar to key attributes of a relation), which uniquely identifies each particular instance of the node. The schema of the database holds information concerning what attributes are keys for each node. The key attributes for each node mentioned in the user's input will be added to the select list. This will guarantee that a unique identifier is returned to the user for the particular information the user is interested in. For instance, in the example input, "Are there any airports that can land a C-5?", the keyword "airports" denotes the node *airports* in the Semantic Graph. Since this input does not have an explicit select list, the key attribute for the node *airports* is chosen as an attribute for the select list.

3.4 An Example of Query Formulation using the Natural Language Interface

Recall that the input to the High-level Query Formulator is a Formulator Input consisting of three parts: the incomplete query topic, the select list, and the query constraints. The *incomplete query topic* is determined by all the node and attributes mentioned in the user input. The database values and operators are used to construct the *query constraints* and the *select list* is determined using the methods described in the previous section.

Suppose the user input is, I = "Find me airports in Tunisia with runway length greater than 5000 feet." This natural language input would correspond to the SQL query shown in figure 4. First, all the keywords need to be spotted from the user input I. As described previously, this involves matching individual words or subsequences of words in I with the set of natural language labels for each node and attribute of the Semantic Graph, the database values of each table in the database, and the names of each operator. For our specific user input I, the keywords are

```
select airports.aport_nm
from airports, runways
where runways.runway_length_ft > 5000
and airports.country_nm = 'Tunisia'
and runways.aport_nm = airports.aport_nm
```

Figure 4: Example Formulated Query

"airports", "Tunisia", "runway length", "greater than" and "5000". Next, the system must determine the meaning set for each keyword using the *meaning_set()* function. The resulting meaning sets are shown in figure 5.

meaning_set("airports") = {(node, *airports*)}

meaning_set("Tunisia") = {(database value, *country.country_nm*)}

meaning_set("runway length") = {(attribute, *runways.runway_length_ft*)}

meaning_set("greater than") = {(operator, *runways.runway_length_ft*), (operator, *runways.runway_width_ft*), ...}

meaning_set("5000") = {(database value, *runways.runway_length_ft*), (database value, *facility.storage_area*), ...}.

Figure 5: Example Meaning Sets

Notice that the keyword "greater than" can correspond to any numeric attribute in the database and the keyword "5000" can correspond to any attribute which can have 5000 as a value.

Once the meaning sets for each keyword is determined, then the set of keyword-meaning pairs KM can be generated. Using the set KM, for each distinct keyword, the system can assign a ranking measure which is the cosine of the angle between the n-gram vector for the user input I, $\vec{V}_I$, and the n-gram vector for each of the keyword's meanings m, $\vec{V}_m$. After the vector similarity measures have been assigned to each meaning m of each keyword, the m which has the highest ranking among all the possible meanings for each keyword will be the *final meaning* of the keyword.

The system can then generate the Formulator Input based on the information extracted from the user input, given the final meaning of each keyword. Any node or attribute will participate in the *incomplete query topic*, any database value and operator, along with attributes, will participate in the *query constraints*, and the select list will be determined by the heuristics mentioned in section 3.3. For the query in figure 4, the query constraints correspond to the WHERE condi-

tions *runways.runway_length_ft* > *5000* and *airports.country_nm* = *'Tunisia'*, assuming the vector similarity measure ranked the meaning (database value, *runways.runway_length_ft*) the highest for the keyword "5000" and the meaning (operator, *runways.runway_length_ft*) the highest for the keyword "greater than". The keyword "Tunisia" does not need to be ranked since it is unambiguous. The select list of the query is the key attribute for the node *airports*. The query also has a join condition between the *runways* table and the *airports* table, which was supplied by the High-level Query Formulator.

3.5 Determining N-Gram Vectors

In traditional statistical language understanding, statistics are gathered using a training corpus of language which is similar to the language the system is designed to handle. Our system also needs to be trained in order to gather enough statistics for each n-gram vector associated with each meaning in the database domain. We feel it may be somewhat difficult to obtain a corpus of natural language questions for each database domain. This will increase the overhead for porting the system to different domains, something we would like to avoid. One simple method for obtaining data is to gather statistics as the system is being used. Each time the system is used, the user input n-gram vector $\vec{V}_I$ is added to each meaning vector $\vec{V}_m$ if a keyword $k \in I$ has the meaning m. This requires that the user correct any mistakes the system may have made in disambiguating keyword meanings. We feel that a form-based clarification interface would be a clear and simple way for users to interact with the system. The system would present the interpreted query in a form, and the user would review the form and make any necessary changes to the form. The user would then "submit" the form when it has been deemed to be correct. An example of a clarification form is shown in figure 6. Here, the system displays the user's original input and displays the relevant information in the style of a form. Using point-and-click input, the user is free to change the elements enclosed in a box. For instance, if the keyword "Tunisia" does not correspond to a country name, then the user can click the "Country name" box which will result in a menu of the other possible attributes for the value "Tunisia".

The submitted form would have each keyword and its correct corresponding meaning. The system would then generate the set of keywords from the user input I and produce an n-gram vector $\vec{V}_I$. Given a keyword-meaning pair from the form, the system would add $\vec{V}_I$ to the meaning's vector, $\vec{V}_m$. This would update the statistics for the next time the system is used. Obtaining n-gram statistics through user interaction has the advantage of having the system accumulate information as it is being used, but has the disadvantage that the accuracy of the system may be poor at the beginning stages of operating within a new domain. The accuracy of the system, however, is expected to improve as more statistics are obtained. This area of clarification dialogue and gathering statistics during system use is currently being researched.

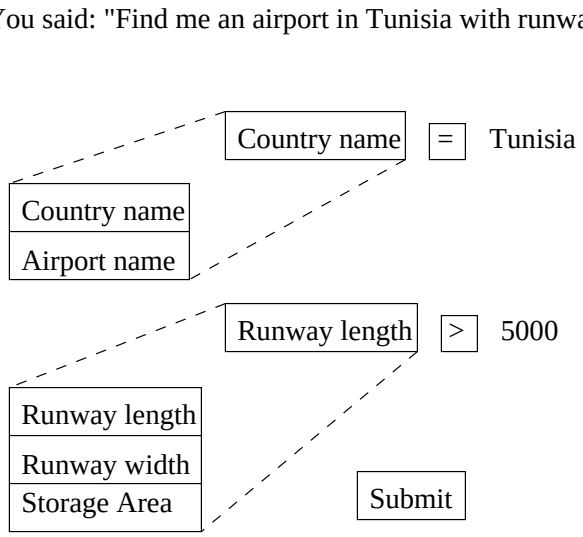

Figure 6: Example Form-based Clarification User Interface

4 Conclusion

We have developed a natural language query formulation methodology based on a semantic model which contains semantic information from the database. We use keywords to approximate the meaning of the user's input. Keywords are words or phrases within the database domain which have particular meaning. The natural language labels of nodes and natural language labels of attributes of the Semantic Graph, the values contained in the database and the operators are all keywords in our system. From the user's natural language input, the system can determine what keywords are present and how they correspond to the components of the Semantic Graph, database values, or operators. A statistical method based on n-gram vectors is used to disambiguate keyword meanings. The final meaning of the keywords from the user's input are used by the High-level Query Formulator to compose a formal database query. The system uses no grammars or much outside knowledge and most of the knowledge utilized by the system can be obtained from the database itself. This system is most suitable for applications where the user input is short and simple, and it is necessary to support domains with a limited ontology. This system is portable in that there is little overhead when moving from one domain to another and it is rather flexible, because there is no hard restriction on what the user input must be. We feel that with the advent of information systems being readily accessible to the general populace through information sources such as the world wide web, portable and flexible high-level interfaces will be needed.

To allow the system to gather statistical data as it is being used, more research needs to be done in the area of interacting with the user for clarification, where the user is asked to correct any mistakes made by the system in the process of keyword meaning disambiguation.

References

[1] James F. Allen. *Natural Language Understanding.* Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interface to databases - an introduction. *Journal of Natural Language Engineering*, 1:29–81, 1995.

[3] Eugene Charniak. *Statistical Language Learning.* The MIT Press, Cambridge, Massachusetts, 1993.

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

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

[6] 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*, 6(11), 1996.

[7] Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978.

[8] Gerard Salton. *Automatic Text Processing.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[9] David L. Waltz. An english language question answering system for a large relation database. *Communications of the ACM*, 21(7):526–539, 1978.

[10] Y. Wilks. An intelligent analyzer and understander of english. *Communications of the ACM*, 18(5):264–274, 1975.

[11] W. A. Woods. Semantics and quantification in natural language question answering. In M. Yovitz, editor, *Advances in Computers*, volume 17. Academic Press, 1978.

[12] Guogen Zhang. *Interactive Query Formulation Techniques for Databases.* PhD thesis, University of California, Los Angeles, 1998.