Supporting Queries with Imprecise Constraints

Ullas Nambiar
Dept. of Computer Science
University of California, Davis

Subbarao Kambhampati
Dept. of Computer Science
Arizona State University

Abstract

In this paper, we motivate the need for and challenges involved in supporting imprecise queries over Web databases. Then we briefly explain our solution, AIMQ - a domain independent approach for answering imprecise queries that automatically learns query relaxation order by using approximate functional dependencies. We also describe our approach for learning similarity between values of categorical attributes. Finally, we present experimental results demonstrating the robustness, efficiency and effectiveness of AIMQ.

Introduction

The rapid expansion of the World Wide Web has made a variety of autonomous databases like bibliographies, scientific databases, vendor databases etc. accessible to a large number of lay external users. The increased visibility of these *Web databases*[1] has brought about a drastic change in their average user profile from tech-savvy, highly trained professionals to lay users demanding "instant gratification". However, database query processing models have always assumed that the *user knows what she wants* and is able to formulate a query that accurately expresses her needs. Hence, to obtain a satisfactory answer from a Web database, the user must formulate a query that accurately captures her information need; often a difficult endeavor. Although users may not know how to phrase their queries, they can often tell which tuples are of interest to them when presented with a mixed set of results having varying degrees of relevance to the query. Supporting a ranked query answering model requires database query processing models to embrace the IR systems' notion that *user only has vague ideas of what she wants*, is unable to formulate queries capturing her needs and would prefer getting a ranked set of answers. This shift in paradigm would necessitate supporting *imprecise queries*- queries that only require the answer tuples to match the constraints closely and not exactly.

We use the following example to motivate the need for supporting imprecise queries over databases.

[1]We use the term "Web database" to refer to a non-local autonomous database that is accessible only via a Web (form) based interface.

Example: Suppose a user wishes to search for *sedans* priced *around* $\$10000$ in a used car database, *CarDB(Make, Model, Year, Price, Location)*. Based on the database schema the user may issue the following query:

Q:- CarDB(Model = Camry, Price < 10000*)*

On receiving the query, CarDB will provide a list of *Camrys* that are priced below $\$10000$. However, given that *Accord* is a similar car, the user may also be interested in viewing all *Accords* priced around $\$10000$. The user may also be interested in a *Camry* priced $\$10500$. □

In the example above, the query processing model used by CarDB would not suggest the *Accords* or the slightly higher priced *Camry* as possible answers of interest as the user did not specifically ask for them in her query. This will force the user to enter the tedious cycle of iteratively issuing queries for all "similar" models before she can obtain a satisfactory answer. One way to automate this is to provide the query processor information about similar models (e.g. to tell it that *Accords* are 0.9 similar to *Camrys*). While such approaches have been tried, their Achilles heel has been the acquisition of the domain specific similarity metrics–a problem that will only be exacerbated as the publicly accessible databases increase in number.

The motivation behind developing the imprecise query answering system AIMQ (Nambiar & Kambhampati. 2006) is not to take the human being out of the loop, but to considerably reduce the amount of input she has to provide to get a satisfactory answer. Specifically, we want to test *how far we can go (in terms of satisfying users) by using only the information contained in the database: How closely can we model the user's notion of relevance by using only the information available in the database?* Therefore, in AIMQ we are interested in ranking an answer tuple t for a query Q using the *relevance* the user U would assign to the tuple. Ideally, our problem can be solved if we have access to the *relevance function* $R(t|Q, U)$. Since such a function is not readily available, past research has looked at either eliciting relevance models through user interactions; or requiring hard-coding relevance functions (e.g. through expert specified attribute weights). Since both are intrusive and hence conflicting with our desire to minimize user input, we developed techniques for assessing the relevance in a *non-intrusive* manner. In this paper, we describe two techniques for measuring $R(t|Q, U)$ - one where the relevance function is assessed from a sample of the database and another where

it is assessed with the help of query logs.

While the primary challenge we face is that of measuring $R(t|Q,U)$, for our solution to be practical given that the target databases are autonomous and on the Web, we also need to efficiently retrieve tuples that are likely to have high relevance scores given a query. This necessitates a query relaxation to identify possible high relevance neighbourhood of the query. A high level view of our solution is given on the righthand side of Figure 1. Specifically, AIMQ mines two types of information from a sample of the databases– *Approximate Functional Dependencies (AFDs)*, and *supertuples* (see below). AFDs are used to do query relaxation, while AFDs and supertuples are used in conjunction to develop a relevance measure. Below we illustrate our proposed solution and highlight the challenges raised by it. Continuing with the example given above, let the user's intended query be:

Q:- CarDB(Model like Camry, Price like 10000*)*

Note that for simplicity, we have used the relation `like` to represent all similarity relationships such as `like, around, contains etc`. We begin by assuming that the tuples satisfying some specialization of Q – called the *base query* Q_{pr}, are *indicative* of the answers of interest to the user. For example, it is logical to assume that a user looking for cars like *Camry* would be happy if shown a *Camry* that satisfies most of her constraints. Hence, we derive Q_{pr} by tightening the constraints from *"likeliness"* to *"equality"*:

Q_{pr}*:- CarDB(Model = Camry, Price =* 10000*)*

Our task then is to start with the answer tuples for Q_{pr} – called the *base set*[2], (1) find other tuples similar to tuples in the base set and (2) rank them in terms of similarity to Q. Our idea is to consider each tuple in the base set as a (fully bound) selection query, and issue relaxations of these selection queries to the database to find additional similar tuples.

Challenges: Given the above solution, our first challenge is: *Which relaxations will produce more similar tuples?* Once we handle this and decide on the relaxation queries, we can issue them to the database and get additional tuples that are similar to the tuples in the base set. However, unlike the base tuples, these tuples may have varying levels of relevance to the user. They thus need to be *ranked* before being presented to the user. This leads to our second challenge: *How to compute the similarity between the query and an answer tuple?* Our problem is complicated by our interest in making this similarity judgement not depend on user-supplied distance metrics.

The AIMQ Approach

In response to the above challenges, we developed the query processing framework AIMQ. Below we briefly explain how we learn attribute importance and value similarity measures from the database. Details of our solution are in (Nambiar & Kambhampati. 2006). The AIMQ system architecture is illustrated in the left half of Figure 1. The *Data Collector* probes the databases to extract sample subsets of the databases. *Dependency Miner* mines approximate dependencies from the probed data and uses them to determine a dependence based importance ordering among the attributes. The *Similarity Miner* uses an association based similarity estimation approach to approximate similarity between categorical values.

Finding Relevant Answers

Given an imprecise query, AIMQ begins by deriving a precise query (called base query) that is a specialization of the imprecise query. Then to extract other relevant tuples from the database it derives a set of precise queries by considering each answer tuple of the base query as *a relaxable selection query*.[3] Relaxation involves extracting tuples by identifying and executing new queries obtained by reducing the constraints on an existing query - a tuple t that is an answer to base query. In theory the tuples most similar to t will have differences only in the least important attribute. Therefore the first attribute to be relaxed must be the *least important attribute* - an attribute whose binding value, when changed, has minimal effect on values binding other attributes.

Estimating Attribute Importance: Identifying the least important attribute necessitates an ordering of the attributes in terms of their dependence on each other. A simple solution is to make a dependence graph between attributes and perform a topological sort over the graph. Functional dependencies can be used to derive the attribute dependence graph that we need. But, full functional dependencies (i.e. with 100% support) between all pairs of attributes (or sets encompassing all attributes) are often not available. Therefore we use approximate functional dependencies (AFDs) between attributes to develop the attribute dependence graph with attributes as nodes and the relations between them as weighted directed edges. However, the graph so developed often is strongly connected and hence contains cycles thereby making it impossible to do a topological sort over it. Constructing a DAG by removing all edges forming a cycle will result in much loss of information.

We therefore propose an alternate approach to break the cycle. We partition the attribute set into *dependent* and *deciding* sets, with the criteria being each member of a given group either depends or decides at least one member of the other group. A topological sort of members in each subset can be done by estimating how dependent/deciding they are with respect to other attributes. Then by relaxing all members in the dependent group ahead of those in the deciding group we can ensure that the least important attribute is relaxed first. We use the approximate key with highest support to partition the attribute set. All attributes forming the approximate key become members of the *deciding set* while the remaining attributes form the *dependent set*.

[2]We assume a non-null resultset for Q_{pr} or one of its generalizations. The attribute ordering heuristic we describe later in this paper is useful in relaxing Q_{pr} also.

[3]The technique we use is similar to the pseudo-relevance feedback technique used in IR systems.Pseudo-relevance feedback (also known as local feedback or blind feedback) involves using top k retrieved documents to form a new query to extract more relevant results.

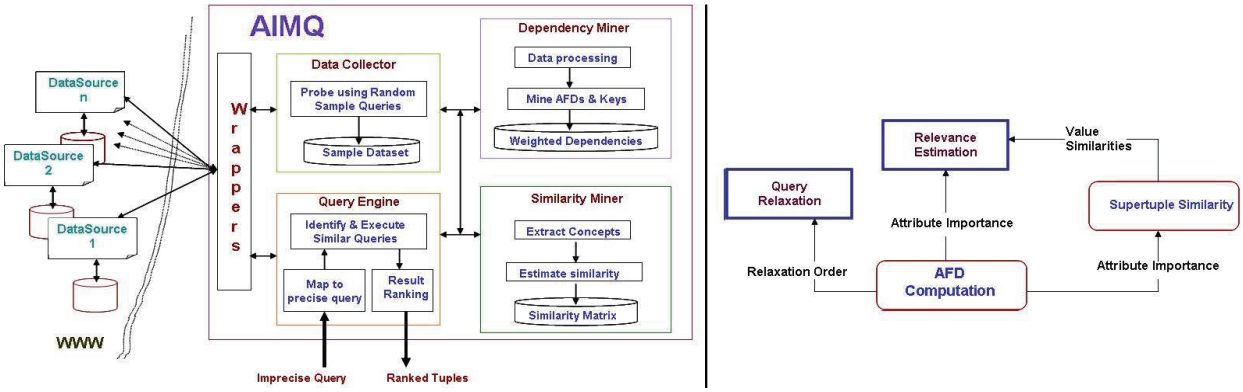

Figure 1: AIMQ system architecture

Measuring Relevance of Answers

The tuples obtained after relaxation must be ranked in terms of their relevance to the query. We measure the relevance of a tuple t to query Q as similarity between Q and an answer tuple t. Thus, $R(t|Q,U) = Sim(Q,t)$ and is measured as

$$Sim(Q,t) = \sum_{i=1}^{n} W_{imp}(A_i) \times \begin{cases} VSim(Q.A_i, t.A_i) & \text{if Domain}(A_i) = \text{Categorical} \\ 1 - \frac{Q.A_i - t.A_i}{Q.A_i} & \text{if Domain}(A_i) = \text{Numerical} \end{cases}$$

where $n = Count(boundattributes(Q))$, W_{imp} ($\sum_{i=1}^{n} W_{imp} = 1$) is the importance weight of each attribute, and *VSim* measures the similarity between the categorical values as explained below. If the numeric distances computed using $\frac{Q.A_i - t.A_i}{Q.A_i} > 1$, we assume the distance to be 1 to maintain a lowerbound of 0 for numeric similarity.

Categorical Value Similarity Estimation: The similarity between two values binding a categorical attribute, *VSim*, is measured as the percentage of common *Attribute-Value pairs(AV-pairs)* that are associated to them. An AV-pair consists of a distinct combination of a categorical attribute and a value binding the attribute. *Make=Ford* is an example of an AV-pair. An AV-pair can be visualized as a selection query that binds only a single attribute. By issuing such a query over a sample of the database we can identify a set of tuples all containing the AV-pair. We represent the answerset containing each AV-pair as a structure called the *supertuple*. The supertuple contains a bag of keywords for each attribute in the relation not bound by the AV-pair.

We measure the similarity between two AV-pairs as the similarity shown by their supertuples. We use *Jaccard Coefficient with bag semantics* (Baeza-Yates & Ribiero-Neto. 1999) to determine the similarity between two supertuples. However, all features (attributes of the relation) may not be equally important for deciding the similarity between two categorical values. For example, given two cars, their prices may have more importance than their color in deciding the similarity between them. Therefore we compute AV-pair similarity as a weighted sum of the attribute bag similarities.

$VSim(C_1, C_2) = \sum_{i=1}^{m} W_{imp}(A_i) \times Sim_J(C_1.A_i, C_2.A_i)$

where C_1, C_2 are supertuples with m attributes, A_i is the bag corresponding to the i^{th} attribute, $W_{imp}(A_i)$ is the importance weight of A_i and Sim_J is the Jaccard Coefficient and is computed as $Sim_J(A,B) = \frac{|A \cap B|}{|A \cup B|}$.

Evaluation

Advantages of the developed framework has been evaluated by applying it in the context of *two real-life* datasets: *(1) Yahoo Autos and the (2) US Census Dataset from UCI Machine Learning Repository.* Below we provide few evaluation results. For details of AIMQ system, algorithms developed and their evaluation please see (Nambiar & Kambhampati. 2006).

We designed two query relaxation algorithms *GuidedRelax* and *RandomRelax* for creating selection queries by relaxing the tuples in the base set. *GuidedRelax* makes use of the AFDs and approximate keys and decides a relaxation scheme. The *RandomRelax* mimics the random process by which users would relax queries by arbitrarily picking attributes to relax. To compare the relevance of answers we provide, we also set up another query answering system that uses the ROCK (Guha, Rastogi, & Shim 1999). Figure 2 provides a graphical representation of the estimated similarity between some of the values binding attribute *Make*.

In order to verify the correctness of the attribute and value relationships we learn and use, we setup a user study over the used car database CarDB. Figure 3 shows the average Mean Reciprocal Rank (MRR) - the metric for relevance estimation used in TREC QA evaluations (Voorhees 1999)) ascribed to both the query relaxation approaches. *GuidedRelax* has higher MRR than *RandomRelax* and ROCK. Thus, the attribute ordering heuristic used by AIMQ is able to closely approximate the importance users ascribe to the various attributes of the relation.

Exploiting Workloads to Estimate User Interest

The AIMQ system's primary intent was minimizing the inputs a user has to provide before she can get answers for her imprecise query. However, in doing so, AIMQ fails to include users' interest while deciding the answers. A naive

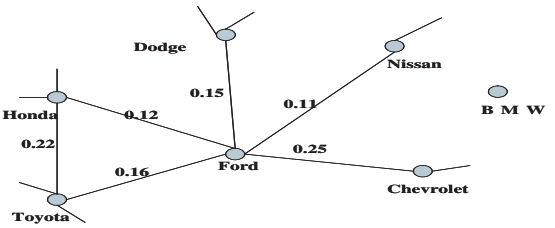

Figure 2: Similarity Graph for Make="Ford"

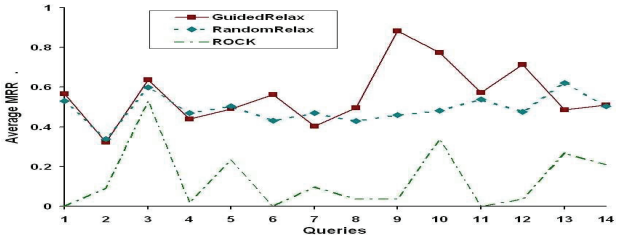

Figure 3: Average MRR over CarDB

solution would be to ask user to provide feedback about the answers she receives. But doing so would negate the benefits of AIMQ. Database *workloads - log of past user queries*, have been shown as being a good source for implicitly estimating the user interest (Agrawal *et al.* 2003). In a way, this may be viewed as a poor mans choice of relevance feedback and collaborative filtering where a users final choice of relevant tuples is not recorded. Despite its primitive nature, such workload information can help determine the frequency with which database attributes and values were often requested by users and thus may be interesting to new users. Therefore, we developed AIMQ-Log, a system that answers imprecise queries by using queries previously issued over the system.

AIMQ-Log differs from AIMQ in the way it identifies the set of precise queries that are used to extract answers from the database. AIMQ-Log identifies the set of relevant precise queries from the set of frequent queries appearing in the database workload. The idea is to use the collective knowledge of the previous users to help the new user. For example, an user looking for vacation rentals *around LA* would not know that a majority of such rentals are near *Manhattan Beach*, a popular tourist destination. However, since it is a popular destination, other experienced tourists may submit queries asking for vacation rentals *around Manhattan Beach, LA*. Thus, by identifying the relevant set of queries from the popular queries in the workload we are implicitly using user feedback. To determine the relevant queries, we compute the similarity between the base query and the popular queries in the workload. The similarity is determined as the *similarity among the answersets* generated by the queries. AIMQ-Log uses the same tuple ranking model as that of AIMQ.

Advantages of AIMQ-Log has been evaluated by applying it in the context of the *online bibliographic data source, BibFinder.* Details of the approach and the evaluation study are available in (Nambiar & Kambhampati. 2004).

Related Work

The problem of answering imprecise queries is related to three other problems. They are (1) *Empty answerset problem*- where the given query has no answers and needs to the relaxed (Muslea 2004; Gasterland 1997). (2) *Structured query relaxation* - where a query is relaxed using only the syntactical information about the query (S. Amer-Yahia & Srivastava 2002). (3) *Keyword queries in databases* - Recent research efforts (Aditya *et al.* 2002; Hristidis & Papakonstantinou. 2002) have looked at supporting keyword search style querying over databases. The imprecise query answering problem differs from the first problem in that we are not interested in just returning some answers but those that are likely to be relevant to the user. It differs from the second and third problems as we consider the semantic relaxations rather than the purely syntactic ones. More detailed evaluation of related research is available in (Nambiar & Kambhampati. 2006).

Conclusion

In this paper we first motivated the need for supporting imprecise queries over databases and presented a domain-independent approach we developed, AIMQ, for answering such queries. The efficiency and effectiveness of our system has been evaluated over two real-life databases, Yahoo Autos and Census database. To the best of our knowledge, AIMQ is the only domain independent system currently available for answering imprecise queries. It can be (and has been) implemented without affecting the internals of a database thereby showing that it could be easily implemented over any autonomous Web database.

References

Aditya, B.; Bhalotia, G.; Chakrabarti, S.; Hulgeri, A.; Nakhe, C.; Parag; and Sudarshan., S. 2002. BANKS: Browsing and Keyword Searching in Relational Databases. *In proceedings of VLDB*.

Agrawal, S.; Chaudhuri, S.; Das, G.; and Gionis, A. 2003. Automated Ranking of Database Query Results. *CIDR*.

Baeza-Yates, R., and Ribiero-Neto., B. 1999. *Modern Information Retrieval.* Addison Wesley Longman Publishing.

Gasterland, T. 1997. Cooperative Answering through Controlled Query Relaxation. *IEEE Expert*.

Guha, S.; Rastogi, R.; and Shim, K. 1999. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *In proceedings of ICDE*.

Hristidis, V., and Papakonstantinou., Y. 2002. Discover: Keyword Search in Relational Databases. *In proceedings of VLDB*.

Muslea, I. 2004. Machine Learning for Online Query Relaxation. *KDD*.

Nambiar, U., and Kambhampati., S. 2004. Providing Ranked Relevant Results for Web Database Queries. *In proceedings of WWW (Alternate Track Papers & Poster)*.

Nambiar, U., and Kambhampati., S. 2006. Answering Imprecise Queries over Autonomous Web Databases. *In proceedings of ICDE*.

S. Amer-Yahia, S. C., and Srivastava, D. 2002. Tree pattern relaxation. *EDBT*.

Voorhees, E. 1999. The TREC-8 Question Answering Track Report. *TREC 8*.