Answering Imprecise Queries over Web Databases

Ullas Nambiar & Subbarao Kambhampati

Dept of Computer Science & Engineering
Arizona State University
Arizona, USA
{*ubnambiar,rao*}*@asu.edu*

1 Introduction

The rapid expansion of the World Wide Web has made a large number of databases like bibliographies, scientific databases etc. to become accessible to lay users demanding "instant gratification". Often, these users may not know how to precisely express their needs and may formulate queries that lead to unsatisfactory results.

For 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. This leads the user into a tedious cycle of iteratively issuing queries for all "similar" models and prices before she can obtain a satisfactory answer. Therefore, database query processing models must embrace the IR systems' notion that *user only has vague ideas of what she wants* and is unable to formulate queries capturing her needs precisely. This shift in paradigm would necessitate supporting *imprecise queries* - a direction given much importance in the Lowell report [2].

While the problem of supporting imprecise database queries has received some attention [8, 10, 6] over the years, most proposed approaches require users and/or database designers to provide domain specific distance metrics and importance measures for attributes of interest. Unfortunately, such information is hard to elicit from the users. In this demo, we present the **AIMQ** system [9] - a domain and user independent solution for supporting imprecise queries over autonomous Web databases.

The Problem: Given a conjunctive query Q over an autonomous Web database projecting relation R, find all tuples of R that show similarity to Q above a threshold $T_{sim} \in (0, 1)$:

$Ans(Q) = \{x | x \in R,\ Similarity(Q,x) > T_{sim}\}$

Constraints: (1) R supports the boolean query processing model (i.e. a tuple either satisfies or does not satisfy a query). (2) The answers to Q must be determined without altering the data model or requiring additional guidance from users. □

The AIMQ Approach: Continuing with the example given above, let the user's intended query be:

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

Given such a query, 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. We derive[1] Q_{pr} by tightening the constraints from "likeliness" to "equality":

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

Starting with the answer tuples for Q_{pr} – called the *base set*, AIMQ (1) finds other tuples similar to tuples in the base set and (2) ranks 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: The first challenge in realizing the AIMQ approach is: *Which relaxations will produce more similar tuples?* Once we handle this we can get additional tuples that are similar to the tuples in the base set by issuing the relaxed queries. However, 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 the answer tuple?* Our problem is complicated by our interest in making this similarity judgement not be dependent on user-supplied distance metrics.

Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

[1] We assume a non-null resultset for Q_{pr} or some generalization of Q_{pr}. This generalization (relaxation) can also be guided by the heuristic we developed as part of AIMQ.

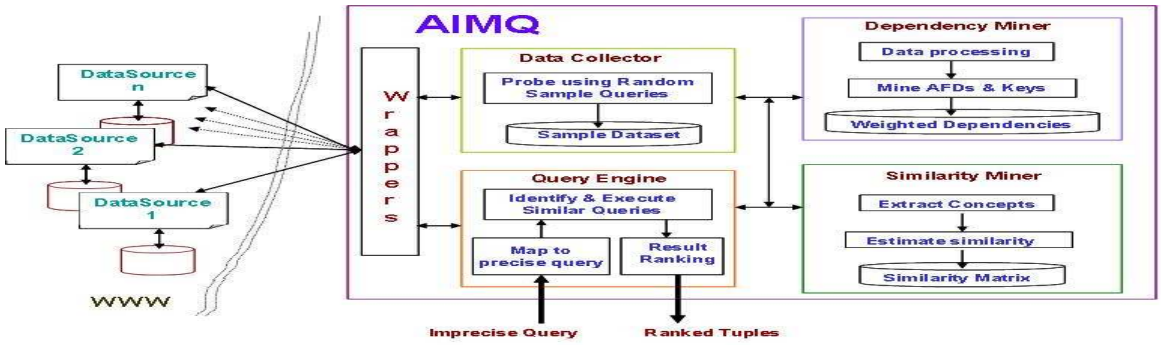

Figure 1: AIMQ system architecture

Solutions: One way of handling the first challenge above is to arbitrarily pick attributes for relaxation. However, this could generate a large number of tuples of possibly low relevance. In theory, the tuples closest to a tuple in the base set will have differences in the attribute that least affects the binding values of other attributes. Such relationships are captured by *approximate functional dependencies* (AFDs). Therefore, AIMQ makes use of AFDs between attributes to determine the degree to which a change in the value of an attribute affects other attributes. Using the mined attribute dependency information AIMQ obtains a heuristic to guide the query relaxation process. To the best of our knowledge, there is no prior work that automatically learns attribute importance measures (required for efficient query relaxation). Hence, the *first contribution* of AIMQ is a domain and user independent approach for learning attribute importance. AIMQ makes use of approximate functional dependencies (AFDs) between attributes to derive a heuristic to guide the query relaxation process.

The tuples obtained after relaxation are not equally relevant to the query and this leads to our second challenge. To estimate the query-tuple similarities we will need distance functions for both numerical and categorical attributes. We can safely use Euclidean distance to capture numeric value similarity. But, while some effort at estimating categorical value similarity [3, 7, 4, 5] has been done, the solutions suggested were inefficient, assumed attribute independence or required the users to provide attribute importance measures. Therefore, the *second contribution* of the AIMQ system is an association based domain and user independent approach for estimating similarity between values binding categorical attributes.

2 The AIMQ System

The AIMQ system as illustrated in Figure 1 consists of four subsystems: *Data Collector*, *Dependency Miner*, *Similarity Miner* and the *Query Engine*. The Data Collector probes the databases to extract sample subsets of the databases. AIMQ also contains wrappers to access the Web databases. However, we do not focus on challenges involved in generating and maintaining the wrappers. We hand coded the wrappers for our demo system. Dependency Miner mines AFDs and approximate keys from the probed data and uses them to determine a dependence based importance ordering among the attributes. This ordering is used by the Query Engine in query relaxation. The Similarity Miner uses an association based similarity mining approach to estimate similarities between categorical values.

2.1 Generating the Relaxation Order

As described above, our solution for answering an imprecise query requires us to generate new selection queries by relaxing the constraints of the tuples in the base set. The underlying motivation there is to identify tuples that are closest to some tuple in the base set. Randomly relaxing constraints and executing queries will produce tuples in arbitrary order of similarity thereby increasing the cost of answering the query. In theory, the tuples most similar to a given tuple will have differences only in the least important attribute. Therefore the first attribute to be relaxed must be the *least important attribute*. We define the least important attribute as the attribute whose binding value, when changed, has minimal effect on values binding other attributes.

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 is often 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 are assigned to the deciding set and the remaining to the dependent set. Given the attribute order, we compute the weight to be assigned to each attribute $k \in R$ as $W_{imp}(k)=\frac{RelaxOrder(k)}{|Attributes(R)|}$, where RelaxOrder returns the position at which k will be relaxed. For a prototype implementation of the database CarDB, we found the relaxation order to be Year → Price → Model → Make → Location.

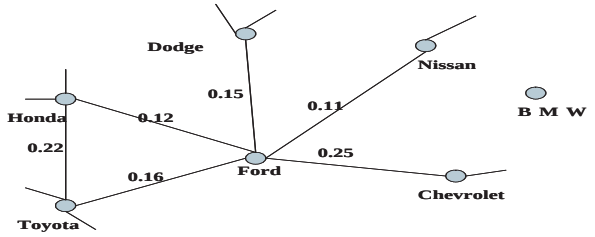

Figure 2: Value Similarity Graph for Make="Ford"

2.2 Query-Tuple Similarity Estimation

AIMQ estimates the similarity between an imprecise query Q and tuple t as

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

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 categorical values as explained below.

The similarity between two values binding a categorical attribute, VSim, is measured as the percentage of distinct Attribute-Value pairs (AV-pair) common to both. An AV-pair can be visualized as a selection query that binds only a single attribute. By issuing such a query over the extracted 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. The similarity between two attribute values (AV-pairs) is measured as the similarity shown by the supertuples. The supertuples contain bags of keywords for each attribute in the relation. Hence we use *Jaccard Coefficient with bag semantics* to determine the similarity between two supertuples. Figure 2 displays the graph showing the values that show similarity above a predefined threshold to Make="Ford".

3 Demonstration

In this demo we will showcase AIMQ's domain and user independent approach for (1) automatically learning the importance of an attribute and (2) measuring the similarity between values binding a categorical attribute. We will give an end-to-end demonstration of imprecise query answering approach of AIMQ using the Yahoo Autos database [1]. Specifically, our demonstration will focus on the following aspects:

Learning Attribute Importance & Value Similarities: AIMQ estimates the similarity of a tuple to a query as the weighted sum of the similarity over the individual attributes. Hence, AIMQ requires attribute importance weights and similarity measures between values binding the attributes. Users are often unable to provide these measures. We will show how the Data Miner learns the attribute importance from AFDs and approximate keys it mines from a probed sample of the database. Also we will demonstrate the ability of the Similarity Miner to automatically build similarity graphs for values binding categorical attributes in a relation. In the context of Yahoo Autos database we will show how we learn value similarities for the attributes Make, Model, Year and Location.

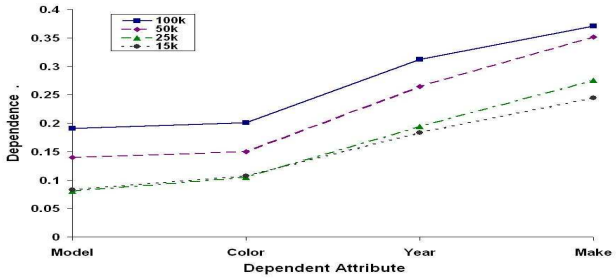

Figure 3: Robustness of mined attribute ordering

Robustness of Estimated Statistics: AIMQ learns attribute importance and value similarities by mining over a probed sample of the database. We will show that the attribute importance and values similarities obtained by sampling do capture the distributions occurring in the database. Specifically, the loss of accuracy incurred due to sampling may not be a critical issue for us as it is the *relative* rather than the *absolute* values of the attribute importance and value similarities that are more important in ranking the answers. Figure 3 shows the robustness of estimated attribute dependencies over Yahoo Autos database for different sample datasets.

Efficiency & Accuracy of AIMQ: AIMQ provides similar answers to an imprecise query by identifying

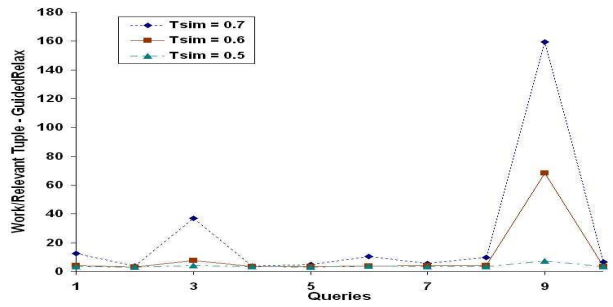

Figure 4: Work/RelevantTuple using GuidedRelax

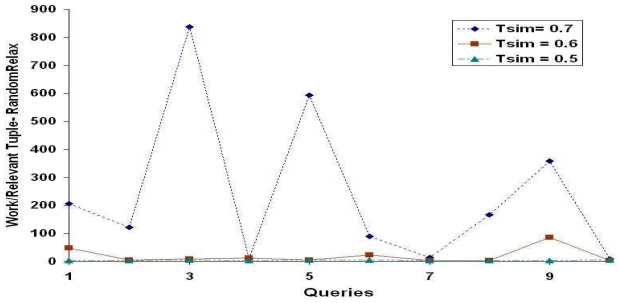

Figure 5: Work/RelevantTuple using RandomRelax

and executing a number of relevant precise queries. Since executing additional queries incurs cost in terms of time and memory one could wonder about the efficiency of AIMQ in answering an imprecise query. Hence, in this demonstration we will use two query relaxation algorithms *GuidedRelax* and *RandomRelax* used by AIMQ system for creating selection queries by relaxing the tuples in the base set. GuidedRelax makes use of the attribute order determined by AIMQ and decides a relaxation scheme. The RandomRelax algorithm was designed to somewhat mimic the random process by which users would relax queries. We measure the relaxation efficiency using the metric *Work/RelevantTuple* defined as the average number of tuples that an user would have to look at before finding a relevant tuple i.e. a tuple showing similarity above threshold T_{sim}. Intuitively the larger the expected similarity, the more the work required to identify a relevant tuple. While both algorithms do follow

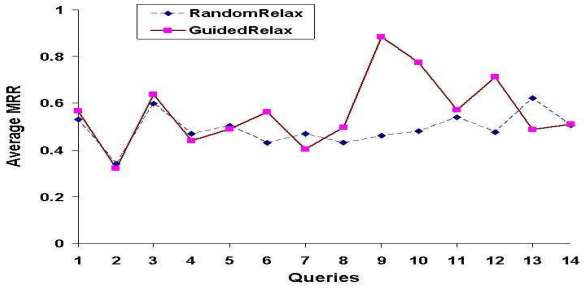

Figure 6: Average MRR over test queries

this intuition, we note that for higher thresholds RandomRelax (Figure 5) ends up extracting hundreds of tuples before finding a relevant tuple. GuidedRelax (Figure 4) is much more resilient to the variations in threshold and generally needs to extract about 4 tuples to identify a relevant tuple.

We conducted a user study and measured the MRR (mean reciprocal rank) [11] - the metric for relevance estimation used in TREC QA evaluations, to compare the relevance of the answers provided by RandomRelax and GuidedRelax. Figure 6 shows that GuidedRelax has higher average MRR for most of the sample queries.

4 Summary

In this demonstration, we presented AIMQ - a domain and user independent system for providing ranked answers to imprecise queries. AIMQ's contributions include - (1)an approach for automatically estimating attribute importance and (2)an association based efficient approach for estimating categorical value similarities.

Acknowledgements: We thank Gautam Das and Kevin Chang for their valuable suggestions during the development of this work. This work is supported by *ECR A*601, the *ASU Prop*301 grant to the ETI^3 initiative.

References

[1] Yahoo! Autos. *Available at http://autos.yahoo.com/.*

[2] The Lowell Database Research Self Assessment. June 2003.

[3] G. Das, H. Mannila, and P. Ronkainen. Similarity of Attributes by External Probes. *In Proceedings of KDD*, 1998.

[4] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS-Clustering Categorical Data Using Summaries. *In Proceedings of KDD*, 1999.

[5] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. *In Proceedings of VLDB*, 1998.

[6] R. Goldman, N .Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity search in databases. *VLDB*, 1998.

[7] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *In Proceedings of ICDE*, 1999.

[8] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, 1998.

[9] U. Nambiar and S. Kambhampati. Mining Approximate Funcitonal Dependencies and Concept Similarities to Answer Imprecise Queries. *WebDB*, June 17-18, 2004.

[10] Micheal Ortega-Binderberger. *Integrating Similarity Based Retrieval and Query Refinement in Databases.* PhD thesis, UIUC, 2003.

[11] E. Voorhees. The TREC-8 Question Answering Track Report. *TREC 8*, November 17-19, 1999.