Mining Approximate Functional Dependencies and Concept Similarities to Answer Imprecise Queries

Ullas Nambiar
Dept of Computer Science & Engg
Arizona State University, USA
mallu@asu.edu

Subbarao Kambhampati
Dept of Computer Science & Engg
Arizona State University, USA
rao@asu.edu

ABSTRACT

Current approaches for answering queries with imprecise constraints require users to provide distance metrics and importance measures for attributes of interest. In this paper we focus on providing a domain and end-user independent solution for supporting *imprecise queries* over Web databases without affecting the underlying database. We propose a query processing framework that integrates techniques from IR and database research to efficiently determine answers for imprecise queries. We mine and use approximate functional dependencies between attributes to create precise queries having tuples relevant to the given imprecise query. An approach to automatically estimate the semantic distances between values of categorical attributes is also proposed. We provide preliminary results showing the utility of our approach.

Keywords

Imprecise Queries, Tuple Similarity, Approximate Functional Dependencies

1. INTRODUCTION

The rapid expansion of the World Wide Web has made a variety of databases like bibliographies, scientific databases, travel reservation systems, vendor databases etc. accessible to a large number of lay external users. The increased visibility of these systems has brought about a drastic change in their average user profile from tech-savvy, highly trained professionals to lay users demanding "instant gratification". Often such users may not know how to precisely express their needs and may formulate queries that lead to unsatisfactory results. Although users may not know how to phrase their queries, they can often tell which tuples are of interest when presented with a mixed set of results with varying degrees of relevance to the query.

Example: Suppose a user is searching a car database for "sedans". Since most sedans have unique model names, the user must convert the general query sedan to queries binding the attribute *Model*. To find all sedans in the database, the user must iteratively change her search criteria and submit queries to the database. Usually the user might know only a couple of models and would end up just searching for them. Thus limited domain knowledge combined with the inflexibility of the query interface can result in the user not obtaining all the relevant results to a query. But if the database were to provide similar answers to a query apart from the exact answers, the user could find all sedans in the database by asking a query such as *Model like "Camry"*. □

Problem Statement: Given a conjunctive query Q over a relation R projected by the autonomous relational database M, find all tuples of R that satisfy Q above a threshold of relevance ϵ. Specifically,

$$Ans(Q) = \{x | x \in R, Sim(Q, x) > \epsilon\} \quad where \quad \epsilon \in \{0, 1\}$$

The database M supports the boolean query processing model (i.e. a tuple either satisfies or does not satisfy a query). To access tuples of R one must issue structured queries over R. The answers to Q must be determined without altering the data model of M. Moreover the solution should require minimal domain-specific information. □

Challenges: Supporting imprecise queries over autonomous Web enabled databases requires us to solve the following problems:

- *Model of similarity:* Supporting imprecise queries necessitates the extension of the query processing model from binary (where tuples either satisfy the query or not) to a matter of the degree (to which a given tuple is a satisfactory answer).
- *Estimating Semantic similarity:* Expecting 'lay' users of the system to provide similarity metrics for estimating the similarity among values binding a given attribute is unrealistic. Hence an important but difficult issue we face is developing domain-independent similarity functions that closely approximate "user beliefs".
- *Attribute ordering:* To provide ranked answers to a query, we must combine similarities shown over distinct attributes of the relation into a overall similarity score for each tuple. While this measure may vary from user to user, most users usually are unable to correctly quantify the importance they ascribe to an attribute. Hence another challenging issue we face is to automatically (with minimal user input) determine the importance ascribed to an attribute.

1.1 Overview of our approach

In response to these challenges, we propose a query processing framework that integrates techniques from IR and database literature to efficiently determine answers for imprecise queries over autonomous databases. Below we begin by describing the query representation model we use and explain how we map from imprecise to precise queries.

Precise Query: A user query that requires data exactly matching the query constraint is a precise query. For exam-

Copyright is held by the author/owner.
Seventh International Workshop on the Web and Databases (WebDB 2004), June 17-18, 2004, Paris, France.

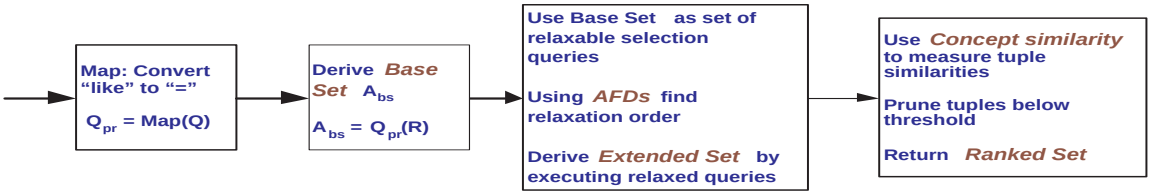

Figure 1: FlowGraph of our Approach

ple, the query

$$Q : -CarDB(Make = \text{“}Ford\text{”})$$

is a precise query, all of whose answer tuples must have attribute 'Make' bound by the value 'Ford'.

Imprecise Query: A user query that requires a close but not necessarily exact match is an imprecise query. Answers to such a query must be ranked according to their closeness/similarity to the query constraints. For example, the query

$$Q : -CarDB(Make \quad like \quad \text{“}Ford\text{”})$$

is an imprecise query, the answers to which must have the attribute 'Make' bound by a value similar to 'Ford'.

Algorithm 1 Finding Relevant Answers

Require: Imprecise Query Q, Relation R, Threshold T_{sim}, Concept Similarities, Approximate Functional Dependencies (AFDs)
begin
Let RelaxOrder = FindAttributeOrder(R, AFDs).
Let Q_{pr} = Map(Q) such that $A_{bs} = Q_{pr}(R)$ & $|A_{bs}| > 0$.
for k=1 to $|A_{bs}|$ **do**
  Q_{rel} = CreateQueries(A_{bs}[k], RelaxOrder)
  for j=1 to $|Q_{rel}|$ **do**
    $A_{rel} = Q_{rel}$[j](R).
    for n=1 to $|A_{rel}|$ **do**
      simval = MeasureSimilarity(A_{rel}[n],A_{bs}[k]).
      if $simval \geq T_{sim}$ **then**
        $A_{es} = A_{es} + A_{rel}$[n].
      end if
    end for
  end for
end for
Return Top-K(A_{es}).
end

Finding Relevant Answers: Figure 1 shows a flow graph of our approach for answering an imprecise query. Algorithm 1 gives further details of our approach. Specifically, given an imprecise query Q,

$$Q : -CarDB(Model \quad like \quad \text{“}Camry\text{”}, Price \quad like \quad \text{“}10k\text{”})$$

over the relation CarDB(Make, Model, Price, Year), we begin by converting Q to a precise *base query*[1](which is equivalent to replacing all "like" predicates with equality predicates) with non-null result set over the database, Q_{pr}. Thus the base query Q_{pr} is

$$Q_{pr} : -CarDB(Model = \text{“}Camry\text{”}, Price = \text{“}10k\text{”})$$

The tuples of CarDB satisfying Q_{pr} also satisfy the imprecise query Q. Thus answers to Q_{pr} form the *base set* of answers to Q, A_{bs}. Suppose A_{bs} contains the tuples

$$Make = \text{“}Toyota\text{”}, Model = \text{“}Camry\text{”}, Price = \text{“}10k\text{”}, Year = \text{“}2000\text{”}$$

$$Make = \text{“}Toyota\text{”}, Model = \text{“}Camry\text{”}, Price = \text{“}10k\text{”}, Year = \text{“}2001\text{”}$$

The tuples in A_{bs} exactly satisfy the base query Q_{pr}. But the user is also interested in tuples that may be similar to the constraints in Q. Assuming we knew that "Honda Accord" and "Toyota Camry" are similar cars, then we could also show tuples containing "Accord" to the user if these tuples had values of Price or Year similar to tuples in A_{bs}. Thus,

$$Make = \text{“}Honda\text{”}, Model = \text{“}Accord\text{”}, Price = \text{“}9.8k\text{”}, Year = \text{“}2000\text{”}$$

could be seen as being similar to the first tuple in A_{bs} and therefore a possible answer to Q. We could also show other Camrys whose Price and Year values are slightly different to those of the tuples in A_{bs}. Specifically, all tuples of CarDB that have one or more binding values close to some tuple in A_{bs} can be considered as potential answers to query Q.

By extracting tuples having similarity above a predefined threshold, T_{sim}, to the tuples in the A_{bs}, we can get a larger subset of potential answers called *extended set*,A_{es}. But to extract additional tuples we would require new queries. We can identify new queries by considering each tuple in the base set, A_{bs}, as a relaxable selection query. However randomly picking attributes of tuples to relax 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 affect the binding values of other attributes. *Approximate functional dependencies*(AFDs) [11] capture relationships between attributes of a relation and can be used to determine the degree to which a change in binding value of an attribute affects other attributes. Therefore we mine approximate dependencies between attributes of the relation and use them to determine a heuristic to guide the relaxation process. Details about our approach of using AFDs to create a relaxation order are in Section 2. Relaxation involves extracting tuples by identifying and executing new relaxed queries obtained by reducing the constraints on an existing query.

Identifying possibly relevant answers only solves part of the problem since we must now rank the tuples in terms of the similarity they show to the seed tuples. We assume that a similarity threshold T_{sim} is available and only the answers that are above this threshold are to be provided to the user. This threshold may be user given or decided by the system. The tuple similarity is estimated as a weighted sum of similarities over distinct attributes in the relation. That is,

$$Sim(t_1, t_2) = \sum_{i=1}^{n} Sim(t_1(A_i), t_2(A_i)) \times W_i$$

where $|attributes(R)| = n$ and $\sum_{i=1}^{n} W_i = 1$. In this paper we assume that attributes have either discrete numerical or categorical binding values. We assume that the Euclidean distance metric captures the semantic similarity between numeric values. But no universal measure is available for measuring the semantic distances between values binding a

[1]In this paper, we assume that the resultset of the base query is non-null. If the resultset of the base query is null, then by iteratively relaxing the base query we may obtain a query that has a non-null resultset. The attributes to be relaxed can be arbitrarily chosen.

categorical attribute. Hence in the Section 3 we present a solution to automatically learn the semantic similarity among values binding a categorical attribute. While estimating the similarity between values is definitely an important problem, an equally important issue is that of assigning weights to the similarity shown by tuples over different attributes. Users can be expected to assign weights to be used for similarity shown over a particular attribute. However, in [19, 18], our studies found users are not always able to map the amount of importance they ascribe to an attribute into a good numeric weight. Hence after determining the attribute order for query relaxation, we will automatically assign importance weights to attributes based on their order i.e. attribute to be relaxed first is least important and so gets lowest weight.

1.2 Organization

The rest of the paper is organized as follows. Section 2 explains how we use approximate functional dependencies between attributes to guide the query relaxation process. Section 3 describes our domain-independent approach for estimating the semantic similarity among concepts binding a categorical attribute. In Section 4, we provide preliminary results showing the accuracy of our concept learning approach and the effectiveness of the query relaxation technique we develop. In Section 5, we compare our work with research done in the areas of similarity search, cooperative query answering and keyword search over databases, all of which focus on providing answers to queries with relaxed constraints. Finally, in Section 6, we summarize our contributions and list the future directions for expansion of this work .

2. QUERY RELAXATION USING AFDS

Our proposed solution for answering an imprecise query requires us to generate new selection queries by relaxing the constraints of tuples in the initial set $\hat{t}_{IS}$. The underlying motivation there is to identify tuples that are closest to some tuple $t \in \hat{t}_{IS}$. 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 t will have differences only in the least important attribute. Therefore the first attribute to be relaxed must be the least important attribute. We define *least important attribute* as the attribute whose binding value, when changed, has minimal effect on values binding other attributes. *Approximate Functional Dependencies (AFDs)* [11] efficiently capture such relations between attributes. In the following we will explain how we use AFDs to identify the importance of an attribute and thereby guide the query relaxation process.

2.1 Definitions

Functional Dependency: For a relational schema R, an expression of the from $X \rightarrow A$ where $X \subseteq R$ and $A \in R$ is a functional dependency over R. The dependency is said to *hold* in a given relation r over R if for all pairs of tuples $t, u \in r$ we have

$$t[B] = u[B] \Rightarrow t[A] = u[A] \quad where \quad A, B \in X$$

Several algorithms [13, 12, 7, 17] have proposed various measures to approximate functional dependencies that hold in a database. Among them, the g_3 measure proposed by Kivinen and Mannila [12], has been widely accepted. The g_3 measure is defined as the minimum number of tuples that need be removed from relation r so that $X \rightarrow Y$ is an FD divided by the number of tuples in r. Huhtala et al [11] have developed an algorithm, *TANE*, for efficiently discovering all AFDs in a database whose g_3 approximation measure is below a user specified threshold. We use *TANE* to extract the AFDs and approximate keys. We mine the AFDs and keys using the a subset of the database extracted by probing.

Approximate Functional Dependency: The functional dependency $X \rightarrow A$ is an approximate functional dependency if it does not hold over a small fraction of the tuples. Specifically, $X \rightarrow A$ is an approximate functional dependency if and only if error$(X \rightarrow A)$ is at most equal to an error threshold ϵ $(0 < \epsilon < 1)$, where the error is measured as the fraction of tuples that violate the dependency.

Approximate Key: An attribute set X is a key if no two distinct tuples agree on X. Let error(X) be the minimum fraction of tuples that need to be removed from relation r for X to be a key. If error(X) is $\leq \epsilon$ then X is an approximate key.

Some of the AFDs determined in the used car database CarDB are: error(Model $\rightarrow$ Make) < 0.1, error(Model $\rightarrow$ Price) < 0.6, error(Make, Price $\rightarrow$ Model) $< .7$ and error(Make, Year $\rightarrow$ Model)< 0.7.

2.2 Generating the relaxation order

Algorithm 2 Query Relaxation Order

Require: Relation R, Tuples(R)
begin
for $\epsilon = 0.1$ to 0.9 **do**
$\hat{S}_{AFDs}$ = ExtractAFDs(R, ϵ).
$\hat{S}_{AKeys}$ = ExtractKeys(R, ϵ).
end for
K_{best} = BestSupport($\hat{S}_{AKeys}$).
NKey = Attributes(R)-K_{best}.
for k=1 to $|K_{best}|$ **do**
Wt_{Key}[k]=[K_{best}[k],Decides(NKey,K_{best}[k], $\hat{S}_{AFDs}$)].
end for
for j=1 to $|NKey|$ **do**
Wt_{NKey}[k]=[NKey[k],Depends(K_{best},NKey[k], $\hat{S}_{AFDs}$)].
end for
Return [Sort(Wt_{Key}), Sort(Wt_{NKey})].
end

The query relaxation technique we use is described in Algorithm 2. Given a database relation R and a dataset containing tuples of R, we begin by extracting all possible AFDs and approximate keys by varying the error threshold. Next we identify the approximate key (K_{best}) with the least error (or highest support). If a key has high support then it implies that fewer tuples will have the same binding values for the subset of attributes in the key. Thus the key can be seen as almost uniquely identifying a tuple. *Therefore we can assume that two tuples are similar if the values binding the key are similar.* All attributes of relation R not found in K_{best} are approximately dependent on K_{best}. Hence by relaxing the non-key attributes first we can create queries whose answers do not satisfy the dependency but have the same key.

We now face the issue of which key (non-key) attribute to relax first. We make use of the AFDs to decide the relaxation order within the two subsets of attributes. For each attribute belonging to the key we determine a weight depending on how strongly it influences the non-key attributes. The influence weight for an attribute is computed as

$$Weight(A_i) = \sum_{j=1}^{n} \frac{1 - error(\hat{A} \rightarrow A_j)}{|\hat{A}|}$$

$$where \quad A_i \in \hat{A} \subseteq R, \quad j \neq i \quad \& \quad n = |Attributes(R)|$$

If an attribute highly influences other non-key attributes then it should be relaxed last. By sorting the key attributes

in ascending order of their influence weights we can ensure that the least influencing attribute is relaxed first. On similar lines we would like to relax the least dependent non-key attribute first and hence we sort the non-key attributes in descending order of their dependence on the key attributes.

The relaxation order we produce using Algorithm 2 only provides the order for relaxing a single attribute of the query. Basically we use a greedy approach towards relaxation and try to create all 1-attribute relaxed queries first, then the 2-attribute relaxed queries and so on. The multi-attribute query relaxation order is generated by assuming independence among attributes and combining the attributes in terms of their single attribute order. E.g., if $\{a1, a3, a4, a2\}$ is the 1-attribute relaxation order, then the 2-attribute order will be $\{a1a3, a1a4, a1a2, a3a4, a3a2, a4a2\}$. The 3-attribute order will be a cartesian product of 1 and 2-attribute orders and so on. Attribute sets appearing earlier in the order are relaxed first. Given the relaxation order and a query Q, we formulate new queries from Q by removing the constraints (if any) on the attributes as given in the order. The number of attributes to be relaxed in each query will depend on order (1-attribute, 2-attribute etc). To ease the query generation process, we assume that the databases do not impose any binding restrictions. For our example database CarDB, the 1-attribute relaxation order was determined as {Make, Price, Year, Model}. Consequently the 2-attribute relaxation order becomes {(Make, Price), (Make, Year), (Make, Model), (Price, Year), (Price, Model), (Year, Model)}.

3. LEARNING CONCEPT SIMILARITIES

Below we provide an approach to solve the problem of estimating the semantic similarity between values binding a categorical attribute. We determine the similarity between two values as the similarity shown by the values correlated to them.

Concept: We define a concept over the database as any distinct attribute value pair. E.g. Make="Ford" is a concept over database CarDB(Make,Model,Price,Year).

Concept Similarity: Two concepts are correlated if they occur in the same tuple. We estimate the semantic similarity between two concepts as the percentage of correlated concepts which are common to both the concepts. More specifically, given a concept, the concepts correlated to a concept can be seen as the features describing the concept. Consequently, the similarity between two concepts is the similarity among the features describing the concepts.

For example, suppose the database CarDB contains a tuple t ={Ford, Focus, 15k, 2002}. Given t, the concept Make="Ford" is correlated to the concepts Model="Focus", Price="15k" and Year="2002". The distinct values binding attributes Model, Price and Year can be seen as features describing the concepts over Make. Similarly Make, Price and Year for Model and so on. Let Make="Ford" and Make="Toyota" be two concepts over attribute Make. Suppose most tuples containing the two concepts in the database CarDB have same Price and Year values. Then we can safely assume that Make="Ford" and Make="Toyota" are similar over the features Price and Year.

3.1 Semantics of a Concept

Databases on the web are autonomous and cannot be assumed to provide any meta-data such as possible distinct values binding an attribute. Hence we must extract this information by probing the database using sample queries. We begin by extracting a small subset of the database by sampling the database. From the extracted subset we can then identify a subset of concepts[2] over the relation.

[2]The number of concepts identified is proportional to the size of the database extracted by sampling. However we can incrementally add new concepts as and when they are encountered and learn similarities for them. But in this paper we do not focus on the issue of incremental updating of the concept space.

Model	Focus:5, ZX2:7, F150:8 ...
Mileage	10k-15k:3, 20k-25k:5, ..
Price	1k-5k:5, 15k-20k:3, ..
Color	White:5, Black:5, ...
Year	2000:6, 1999:5, ...

Table 1: Supertuple for Concept Make='Ford'

A concept can be visualized as a selection query called *concept query* that binds only a single attribute. By issuing the concept query over the extracted database we can identify a set of tuples all containing the concept. We represent the answerset containing each concept as a structure called the *supertuple*. The supertuple contains a bag of keywords for each attribute in the relation not bound by the concept. Table 1 shows the supertuple for the concept Make='Ford' over the relation CarDB as a 2-column tabular structure. To represent a bag of keywords we extend the semantics of a set of keywords by associating an occurrence count for each member of the set. Thus for attribute Color in Table 1, we see 'White' with an occurrence count of five, suggesting that there are five White colored Ford cars in the database that satisfy the concept-query.

3.2 Measuring Concept Similarities

The similarity between two concepts is measured as the similarity shown by their supertuples. The supertuples contain bags of keywords for each attribute in the relation. Hence we use Jaccard Coefficient [10, 2] with bag semantics to determine the similarity between two supertuples. The Jaccard Coefficient (Sim_J) is calculated as

$$Sim_J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

We developed the following two similarity measures based on Jaccard Coefficient to estimate the similarity between concepts:

Doc-Doc similarity: In this method, we consider each supertuple ST_Q, as a document. A single bag representing all the words in the supertuple is generated. The similarity between two concepts C_1 and C_2 is then determined as

$$Sim_{doc-doc}(C_1, C_2) = Sim_J(ST_{C1}, ST_{C2})$$

Weighted-Attribute similarity: Unlike pure text documents, supertuples would rarely share keywords across attributes. Moreover all attributes may not be equally important for deciding the similarity among concepts. For example, given two cars, their prices may have more importance than their color in deciding the similarity between them. Hence, given the answersets for a concept, we generate bags for each attribute in the corresponding supertuple. The similarity between concepts is then computed as a weighted sum of the attribute bag similarities. Calculating the similarity in this manner allows us to vary the importance ascribed to different attributes. The supertuple similarity will then be calculated as

$$Sim_{watr}(C_1, C_2) = \sum_{i=1}^{m} Sim_J(Bag_{C1}(A_i), Bag_{C2}(A_i)) \times W_i$$

where C1, C2 have m attributes

4. EVALUATION

To evaluate the effectiveness of our approach in answering imprecise queries, we set up a prototype used car search

Algorithm Step	Time	Size
SuperTuple Generation	181 sec	11 MB
Similarity Estimation	1.5 hours	6.0 MB

Table 2: Timing Results and Space Usage

database system that accepts precise queries over the relation

$CarDB(Make, Model, Year, Price, Mileage, Location, Color)$

The database was setup using the open-source relational database MySQL. We populated the relation CarDB using 30,000 tuples extracted from the publicly accessible used car database *Yahoo Autos* [20]. The system was hosted on a Linux server running on Intel Celeron- 2.2 Ghz with 512Mb RAM.

4.1 Concept Similarity Estimation

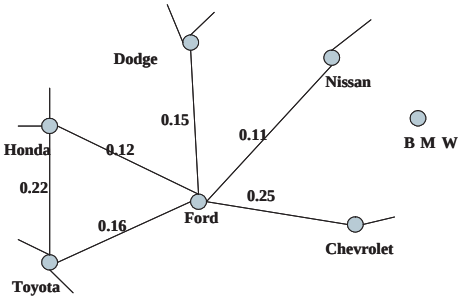

Figure 2: Concept Similarity Graph for Make

The attributes Make, Model, Location, Color in the relation CarDB are categorical in nature and contained 132, 1181, 325 and 100 distinct values. We estimated concept similarities for these attributes as described in Section 3. Time to estimate the concept similarity is high (see Table 2), as we must compare each concept with every other concept binding the same attribute. We calculated only the *doc-doc* similarity between each pair of concepts. The concept similarity estimation is a preprocessing step and can be done offline and hence the high processing time requirement for this process can be ignored. Figure 2 provides a graphical representation of the estimated semantic similarity between some of the values binding attribute Make. The concepts Make="Ford" and Make="Chevrolet" show high similarity and so do concepts Make="Toyota" and Make="Honda" while the concept Make="BMW" is not connected to any other node in the graph. We found these results to be intuitively reasonable and feel our approach is able to efficiently determine the semantic distances between concepts. Moreover in [19, 18] we used a similar approach to determine the semantic similarity between queries in a query log. The estimated similarities were validated by doing a user study and our approach was found to have above 75% accuracy.

4.2 Efficient query relaxation

To verify the efficiency of the query relaxation technique we propose in Section 2, we setup a test scenario using the CarDB database and a set of 10 randomly picked tuples. For each of these tuples our aim was to extract 20 tuples from CarDB that had similarity above some threshold T_{sim} ($0.5 \leq T_{sim} < 1$). We designed two query relaxation algorithms *GuidedRelax* and *RandomRelax* for creating selection queries by relaxing the tuples in the initial set. GuidedRelax makes use of the AFDs and approximate keys and decides a relaxation scheme as described in Algorithm 2. The RandomRelax algorithm was designed to somewhat mimic the

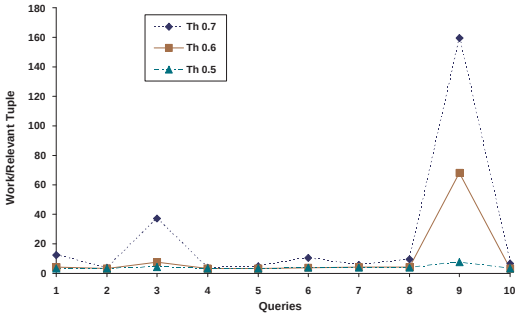

Figure 3: Work/RelevantTuple using GuidedRelax

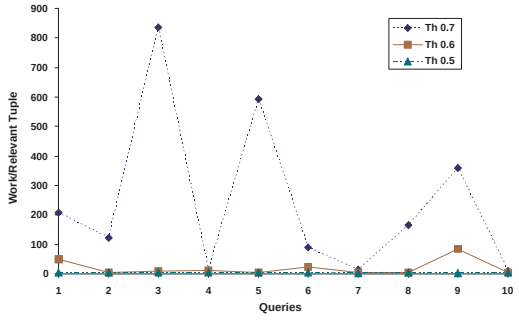

Figure 4: Work/RelevantTuple using RandomRelax

random process by which users would relax queries. The algorithm randomly identifies a set of attributes to relax and creates queries. We put an upper limit of 64 on the number of queries that could be issued by either algorithm for extracting the 20 similar answers to a tuple from the initial set.

To measure the efficiency of the algorithms we use a metric called *Work/RelevantTuple* defined as

$$Work/RelevantTuple = \frac{|T_{Extracted}|}{|T_{Relevant}|}$$

where $T_{Extracted}$ gives the total tuples extracted while $T_{Relevant}$ is the number of extracted tuples that were found as relevant. Specifically Work/RelevantTuple is a measure of the average number of tuples that an user would have to look at before finding a relevant tuple. Tuples that showed similarity above the threshold T_{sim} were considered relevant. Similarity between two tuples was estimated as the weighted sum of semantic similarities shown by each attribute of the tuple. Equal weightage was given to the similarity shown by all attributes.

The graphs in figures Figure 3 and Figure 4 show the average number of tuples that had to be extracted by GuidedRelax and RandomRelax respectively to identify a relevant tuple for the query. Intuitively the larger the expected similarity, the more the work required to identify a relevant tuple. While both algorithms do follow this intuition, we note that for higher thresholds RandomRelax (see Figure 4) ends up extracting hundreds of tuples before finding a relevant tuple. GuidedRelax is much more resilient to the variations in threshold and generally needs to extract about 4 tuples to identify a relevant tuple. Thus by using GuidedRelax, a user would have to look at much less number of tuples before obtaining satisfactory answers.

The initial results we obtained are quite encouraging. However for the current set of experiments we did not verify

whether the tuples considered relevant are truly relevant as measured by the user. We plan to conduct a user study to verify that our query relaxation approach not only saves time but also provides answers that are truly relevant according to the user. The evaluations we performed were aimed at studying the accuracy and efficiency of our concept similarity learning and query relaxation approaches in isolation. We are currently working on evaluating our approach for answering imprecise queries over *BibFinder* [1, 21], a fielded autonomous bibliography mediator that integrates several autonomous bibliography databases such as DBLP, ACM DL, CiteSeer. Studies over BibFinder will enable us to better evaluate and tune the query relaxation approach we use. We also plan to conduct user studies to measure how many of the answers we present are considered truly relevant by the user.

5. RELATED WORK

Early approaches for retrieving answers to imprecise queries were based on theory of fuzzy sets. Fuzzy information systems [14] store attributes with imprecise values, like height= "tall" and color="blue or red", allowing their retrieval with fuzzy query languages. The WHIRL language [6] provides approximate answers by converting the attribute values in the database to vectors of text and ranking them using the vector space model. In [16], Motro extends a conventional database system by adding a *similar-to* operator that uses distances metrics over attribute values to interpret vague queries . The metrics required by the similar-to operator must be provided by database designers. Binderberger [22] investigates methods to extend database systems to support similarity search and query refinement over arbitrary abstract data types. In [9], the authors propose to provide ranked answers to queries over Web databases but require users to provide additional guidance in deciding the similarity. These approaches however are not applicable to existing databases as they require large amounts of domain specific information either pre-estimated or given by the user of the query. Further [22] requires changing the data models and operators of the underlying database while [9] requires the database to be represented as a graph.

In contrast to the above, the solution we propose provides ranked results without re-organizing the underlying database and thus is easier to implement over any database. In our approach we assume that tuples in the base set are all relevant to the imprecise query and create new queries. The technique we use is similar to the pseudo-relevance feedback [3, 8] technique used in IR system. 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.

In [4, 5], authors explore methods to generate new queries related to the user's original query by generalizing and refining the user queries. The abstraction and refinement rely on the database having explicit hierarchies of the relations and terms in the domain. In [15], Motro proposes allowing the user to select directions of relaxation, thereby indicating which answers may be of interest to the user. In contrast, we automatically learn the similarity between concepts and use functional dependency based heuristics to decide the direction for query relaxation.

6. CONCLUSION AND FUTURE WORK

In this paper we first motivated the need for supporting imprecise queries over databases. Then we presented a domain independent technique to learn concept similarities that can be used to decide semantic similarity of values binding categorical attributes. Further we identified approximate functional dependencies between attributes to guide the query relaxation phase. We presented preliminary results showing the efficiency and accuracy of our concept similarity learning and query relaxation approaches.

Both the concept similarity estimation and AFDs and keys extraction process we presented heavily depend on the size of the initial dataset extracted by probing. Moreover the size of the initial dataset also decides the number of concepts we may find for each attribute of the database. A future direction of this work is to estimate the effect of the probing technique and the size of the initial dataset on the quality of the AFDs and concept similarities we learn. Moreover the data present in the databases may change with time. We plan to investigate ways to incrementally update the similarity values between existing concepts and develop efficient methods to compute distances between existing and new concepts without having to recompute the entire concept graph. In this paper we only looked at answering imprecise selection queries over a single database relation. Answering imprecise queries spanning multiple relations forms an interesting extension to our work.

Acknowledgements: We thank Hasan Davulcu for helpful discussions during the development of this work. This work is supported by ECR A601, the ASU Prop301 grant to ETI^3 initiative.

7. REFERENCES

[1] BibFinder: A Computer Science Bibliography Mediator. *Available at :http://kilimanjaro.eas.asu.edu/.*

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

[3] C. Buckley, G. Salton, and J. Allan. Automatic Retrieval with Locality Information Using Smart. *TREC-1, National Institute of Standards and Technology, Gaithersburg, MD*, 1992.

[4] W.W. Chu, Q. Chen, and R. Lee. Cooperative query answering via type abstraction hierarchy. *Cooperative Knowledge Based Systems*, pages 271–290, 1991.

[5] W.W. Chu, Q. Chen, and R. Lee. A structured approach for cooperative query answering. *IEEE TKDE*, 1992.

[6] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. *Proc. of SIGMOD*, pages 201–212, June 1998.

[7] M. Dalkilic and E. Robertson. Information Dependencies. *In Proc. of PODS*, 2000.

[8] N.E. Efthimiadis. Query Expansion. *In Annual Review of Information Systems and Technology, Vol. 31*, pages 121–187, 1996.

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

[10] T. Haveliwala, A. Gionis, D. Klein, and P Indyk. Evaluating strategies for similarity search on the web. *Proceedings of WWW, Hawai, USA*, May 2002.

[11] Y. Huhtala, J. Krkkinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. *Proceedings of ICDE*, 1998.

[12] J. Kivinen and H. Mannila. Approximate Dependency Inference from Relations. *Theoretical Computer Science*, 1995.

[13] T. Lee. An information-theoretic analysis of relational databases-part I: Data Dependencies and Information Metric. *IEEE Transactions on Software Engineering SE-13*, October 1987.

[14] J.M. Morrissey. Imprecise information and uncertainty in information systems. *ACM Transactions on Information Systems*, 8:159–180, April 1990.

[15] A. Motro. Flex: A tolerant and cooperative user interface to database. *IEEE TKDE*, pages 231–245, 1990.

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

[17] K. Nambiar. Some analytic tools for the Design of Relational Database Systems. *In Proc. of 6th VLDB*, 1980.

[18] U. Nambiar and S. Kambhampati. Providing ranked relevant results for web database queries. *To appear in WWW Posters 2004,*, May 17-22, 2004.

[19] U. Nambiar and S. Kambhampati. Answering imprecise database queries: A novel approach. *ACM Workshop on Web Information and Data Management*, November 2003.

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

[21] Z. Nie, S. Kambhampati, and T. Hernandez. BibFinder/StatMiner: Effectively Mining and Using Coverage and Overlap Statistics in Data Integration. *In Proc. of VLDB*, 2003.

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