Answering Imprecise Database Queries : A Novel Approach

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

Subbarao Kambhampati [*]
Dept. of Computer Science
Arizona State University
USA
rao@asu.edu

ABSTRACT

A growing number of databases especially those published on the Web are becoming available to external users. Users of these databases are provided simple form-based query interfaces that hide the underlying schematic details. Constrained by the expressiveness of the query interface users often have difficulty in articulating a precise query over the database. Supporting *imprecise queries* over such systems would allow users to quickly find relevant answers without iteratively refining their queries. For databases to support imprecise queries they must provide answers that closely match the query constraints. In this paper we focus on answering imprecise user queries without changing the existing database system. We propose to support imprecise queries over a database by identifying a set of related precise queries that provide answers that are relevant to the user given query. We present a domain independent approach based on information retrieval techniques to estimate the distance between queries. To demonstrate the utility and usefulness of our approach we perform usability tests and provide results.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Relational databases; H.3.3 [**Information Storage and Retrieval**]: Query formulation, Retrieval models

General Terms

Data Management

Keywords

Imprecise queries, relational database, query similarity

[*]This research is supported in part by Arizona State University grant ECR-A601.

1. INTRODUCTION

The rapid expansion of the Internet has made a variety of databases, including bibliographies, scientific databases, travel reservation systems and vendor databases 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”. Therefore most databases available on the Internet and Web provide easy to use form-based interface for users to interact with the database. The user requests are automatically converted to queries over the database. These form-based interfaces although easy to use come at a price: reduced expressibility of the queries, allowing only conjunctive queries with selection predicates to be issued over the database. The user queries are specified by filling in the form fields, imposing strict constraints (mostly equality) on the attribute values stored in the database. Such queries, however, can lead to unsatisfactory results since the users must express their information need as a conjunctive selection query. The difficulty is compounded by the increasing complexity of data available in the databases like hypertext documents and images. The variety of datatypes make it difficult to formulate exact queries on them. Often users must reformulate their queries a number of times before they can obtain a satisfactory answer. Thus the lack of knowledge about the contents of the database and the limited query capability of the interface can result in the user not obtaining satisfactory answers from the database. The above problem can be solved by providing similar answers if exact answers are not present. The answers can be ranked based on their relevance to the user query. We use the example below to further motivate the problem.

Example: Suppose Jane is searching a car database for SUVs. Most SUVs have unique model names. Hence Jane must convert the general query SUV to exact queries binding the attribute *model*. To find all SUVs in the database, a user must know the model names of all SUVs. Usually the user might know a couple of SUV models, say ”Nissan PathFinder”, ”Ford Escape” and would end up just searching for them. Thus, to get satisfactory results Jane must iteratively change her search criteria and submit queries to the database. But if the car database were to provide similar answers to a query apart from the exact answers, Jane could find all SUVs in the database by asking the query *model= "Ford Escape"*.

Challenges: We refer to queries that require tuples not exactly matching the constraints as *imprecise queries* while those requiring exactly matching tuples are referred to as *precise queries*. Supporting imprecise queries over databases necessitates a system that integrates similarity search paradigm over structured and semi-structured data. Given an imprecise query Q over such a system, the result should be a set of tuples ranked according to their similarity to the query. Estimating the similarity between the query and the tuples of the database is a difficult problem. To simplify the problem existing approaches [7, 5] for answering imprecise queries assume availability of a user given distance measure between every pair of values binding an attribute. Also these systems redefine the operators for selection, projection, join etc. to generate ranked set of tuples by using the distance measures. Thus enabling support for imprecise queries over existing databases involves modifying the database and is clearly a costly endeavor.

Our Contributions: In this paper we propose a domain-independent solution for answering imprecise queries over a database without modifying the existing database. We add a middleware system, *IQE (Imprecise Query Engine)*, between the existing database and the users to support imprecise queries. Given an imprecise query Q over a relation R, the result is a set of tuples ranked according to their similarity to Q. More precisely, apart from tuples exactly matching the query constraints, answers to imprecise queries also contain tuples that have values similar to the constraints. But existing databases only support precise queries i.e. queries requiring tuples that exactly match the query constraints. Therefore given Q, R will only return tuples exactly matching Q. Then to extract tuples similar to Q from R, one must extract additional tuples by *issuing other precise queries over R and determine their similarity to Q*.

The difficulty here is two-fold. First is identifying additional queries over R that have answers similar to Q. Second is to keep the cost of issuing queries low while ensuring enough tuples satisfying Q are obtained. This brings up another difficult problem of estimating inter-query similarity. To mitigate the problem of generating new queries with non-empty answersets, we assume a *workload* of the database containing queries over R is available. Given the workload, our strategy is to map the imprecise query Q to a precise query, say q_p, in the workload. Then we will determine the similarity of q_p to other queries in the workload. To estimate the similarity between two queries we use the document similarity metric, Jaccard Similarity [3], over the answersets of the two queries. The answer to query Q is a ranked set of tuples formed by taking a union of the answer tuples of queries similar to q_p.

In the next section, Section 2, we give an overview of our approach, define the needed terminology and describe a possible architecture for supporting imprecise queries. Section 3 describes a domain-independent approach to estimate query similarity. In Section 4, we conduct a user study and provide results indicating the high precision of the answers we provide to imprecise queries. Related work is discussed in Section 5. A discussion regarding assumptions made and future directions is done in Section 6. Finally we summarize our contributions in Section 7.

2. OVERVIEW

Imprecise Query: A user query that only requires data *closely* matching the query constraint is an imprecise query. For example, the query "Movie *like* Benhur" is an imprecise query.

Precise Query: A user query that requires data exactly matching the query constraint is a precise query. An imprecise query can be converted to a precise query by tightening the relation in the query constraint. For example, tightening the relation "like" to "equal-to" in the imprecise query above gives us the precise query "Movie = Benhur".

Let M be a relational database and R a relation of M that is published on the Internet. M provides a form-based interface for querying the data. Then, for any query Q, the answer will be a set of tuples exactly matching the query constraint or an empty set. Thus all queries over M are treated as precise queries. Hence to support imprecise queries over M we add an *imprecise query engine, IQE,* as middleware between the users and the database M. Figure 1 presents a schematic of the proposed solution. Given an imprecise query Q_{ipr}, the imprecise query engine must then identify all tuples of R that are similar to Q_{ipr}. That is,

$$Ans(Q_{ipr}) = \{x|Sim(Q,x) \in (0,1], x \in R\}$$

$Ans(Q_{ipr})$ can be divided into two sets, one containing tuples only showing similarity value of 1 with Q_{ipr} and the other containing all the remaining tuples. The first set contains all the tuples that satisfy a precise query Q_{pr} derived from Q_{ipr}. The second set can be approximated as containing tuples that are similar to the tuples in the first set. Thus, we can write $Ans(Q_{ipr})$ as

$$Ans(Q_{ipr}) = Tuples(Q_{pr}) + R'$$

$$where \quad R' = \{y|Sim(y,z) \in (0,1), y \in R, z \in Tuples(Q_{pr})\}$$

The database M accepts precise queries and hence we can obtain tuples for Q_{pr} from M. Then to find tuples similar to tuples of Q_{pr}, IQE must access the remaining tuples in M and determine the similarity with the tuples of Q_{pr}. Since accessing all tuples of M to answer each imprecise query encountered by IQE is not a feasible solution, we propose an alternate approach to answer Q_{ipr} by issuing a set of queries whose tuples may be similar to Q_{pr}.

$$Ans(Q_{ipr}) \approx Tuples(Q_{pr}) + Tuples(Q_{ot}) \sim Tuples(Q_{pr})$$

The difficulty arises in determining queries whose results will be similar to that of Q_{pr}. Issuing queries at random will lead to retrieval of a large number of tuples which are not similar to the user query. Also it would be expensive in terms of the cost of querying the sources and identifying the similar tuples. Moreover to generate queries other than Q_{pr}, the IQE will need to know the various values that satisfactorily bind the different attributes of the relation. The problem of

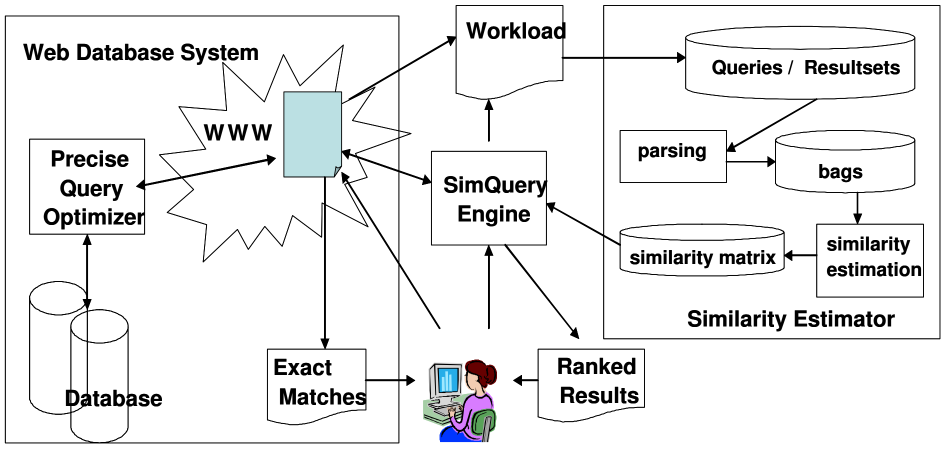


Figure 1: Imprecise Query Engine

additional queries can be resolved by maintaining a query log, Q_{log}, of frequent queries issued on the database. This still leaves open the question of finding the correct set of additional queries whose results will show similarity to Q_{pr}. For this we propose to identify queries that are *similar* to Q_{pr} from the queries appearing in the workload of the database M.

$$Ans(Q_{ipr}) \approx Tuples(Q_{pr}) + Tuples(Q')$$
$$where \quad Q' \sim Q_{pr}, Q' \in Q_{log}$$

Therefore, to answer an imprecise query Q_{ipr} over relation R, IQE will first identify a set of precise queries similar to Q_{ipr} from the workload and issue them over R. The answer to Q_{ipr} is an ordered union of the answers of the precise queries, with each tuple in the union inheriting the similarity of its generating precise query. The similarity between queries can be due to the data they share and/or due to a similarity perceived by the user. Depending on the similarity measure chosen, the query similarities would be calculated as under,

- Data Similarity: Given two queries Q_1 and Q_2 binding same attributes, the similarity shown by them over the unbound attributes is called as *data similarity*. The similarity is defined by a distance function associated with each attribute of the database relation. In this paper we will use equality as the distance measure for all the attributes.

- Semantic Similarity: We define *semantic similarity* between two queries Q_1 and Q_2, as a user perceived distance between the queries based on the semantic relationships between values binding the query attributes. Domain ontologies can be used to determine the semantic distances between values of an attribute. The semantic distance between two queries can then be estimated as a weighted sum of the semantic distance between their values.

Estimating semantic similarity between queries requires access to domain-specific information e.g. ontologies, value hierarchies etc. Our motivation is to propose a domain-independent approach for answering imprecise queries. Hence we use data similarity metric to compute the similarity among queries.

2.1 Architecture

The schematic diagram of a web database system extended to support imprecise queries is given in Figure 1. Although we chose a web database to explain our system, our approach is applicable to structured databases that are not Internet accessible. We start with an existing Web database accepting only precise queries over a form-based interface. Mapping from the form-based interface to database schema is done by the precise query optimizer of the database. To this existing architecture we add :

- A **SimQuery Engine**, that converts the imprecise query into equivalent precise query. If the precise query is not present in the workload seen by the database, the SimQuery Engine will determine queries that are close generalizations of the precise query. It then identifies the related precise queries, extracts the results and presents a ranked list of tuples for the imprecise query. All answer tuples of a precise query inherit the similarity shown by the query.

- A **Similarity Estimator**, to calculate the similarity between each pair of queries in the query log. As shown in Figure 1, the Content Similarity Estimator begins by extracting workload queries whose occurrence frequency is above a pre-defined threshold. If results of these queries were not materialized as part of the workload, Similarity Estimator will probe the database to extract and materialize the results. The resultsets are then parsed to extract frequently occurring keywords for every attribute in the resultset. For every query extracted from the workload a corresponding document containing the bags of keywords is created. The similarity between two queries is then estimated as the data similarity shown by their corresponding documents.

In the next section, we describe our approach for comput-

Author=Ullman	
Co-author	C. Li:5, R. Motwani:7, ...
Title	data-mining:3, optimizing:5, ...
Subject	integration:5, learning:2, ...
Conference	SIGMOD:5, VLDB:5, ...
Year	2000:6, 1999:5, ...

Table 1: SuperTuple for query Publications(Author=Ullman)

ing data similarity and provide two similarity coefficients to estimate the similarity between queries.

3. ESTIMATING QUERY SIMILARITY

Suppose query Q is an imprecise query over a database M and $\hat{Q}$ is a set of precise queries appearing in the workload of M. To answer the imprecise query Q, we must first identify all the queries in $\hat{Q}$ that are similar to Q. The similarity between two queries is calculated as the similarity between their answer tuples. Under the relational data model, two tuples are similar only if they show same values for all the attributes. Therefore using the relational model to determine similarity between queries would result in identifying overlapping queries (having same answer tuples) as being similar. But two queries can be considered similar even if their tuples only match partially, or if they show transitive correlations i.e. they are both similar to a common query. E.g., let Author=Ullman and Author=Widom be two queries on the relation Publications. The author names show no similarity, yet the authors may have publications that fall under the same Subject or appear in the same Conference or Year or a combination of all these. We are interested in capturing and using such relationships to identify the precise queries related to a given imprecise query. If we maintained the strict tuple structure for queries, queries would be similar only if they contained the same tuples. Hence only if, Ullman and Widom, were co-authors would they be seen as related. But Ullman and Widom are related because they work in the same area and write papers in related topics. We believe moving from the relational model to a vector space model will help in capturing additional relations between queries. Therefore we represent query results as a document of keywords and not as a set of tuples thereby moving from the relational model to a vector space model.

We convert the resultset for each query $q \in \hat{Q}$ to a structure called *supertuple*. The supertuple contains a bag of keywords for each attribute in the relation not bound by the query. The keywords are extracted from the resultset of the query. For categorical attributes, each distinct value the attribute takes is considered a keyword. The Table 1 shows the supertuple for the query ($Author = Ullman$) over the relation *Publications* 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 Conference in Table 1, we see 'Sigmod' with an occurrence count of 5, suggesting that Author has 5 tuples that bind the attribute Conference with value 'Sigmod'.

The similarity between two queries is estimated as the similarity between the supertuples of the queries. We use the *Jaccard Similarity metric* to estimate similarity between supertuples. The supertuples contain bags of keywords for each attribute in the relation R over which the queries are issued. Hence we use Jaccard Similarity [3] 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 Similarity to estimate the similarity between queries:

- *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 queries Q_1 and Q_2 is then determined as

  $$Sim_{doc-doc}(Q_1, Q_2) = Sim_J(ST_{Q1}, ST_{Q2})$$

- *Weighted-Attribute similarity:* Unlike pure text documents, supertuples would rarely share keywords across attributes. Also users may not give the same importance to all the attributes while determining similarity between tuples in a relation. E.g., even if two authors published papers in different years, their papers could be quite similar if the topic of the publications are related. Hence in this approach, given two queries, we first generate bags for each attribute in the corresponding supertuple. Next we estimate the similarity between bags of same attributes of the two supertuples. Finally the similarity between queries is computed as a weighted sum of the attribute bag similarities. Calculating the similarity in this manner allows us to give more importance to attributes of interest to the user. The weighted-attribute similarity between supertuples is calculated as

  $$Sim_{watr}(Q_1, Q_2) = \sum_{i=1}^{m} Sim_J(Bag_{Q1}(A_i), Bag_{Q2}(A_i)) \times W_i$$

  where Q1, Q2 have m attributes

Query Similarity Matrix

Using Algorithm 1 we compute a similarity matrix for the queries in query log. The similarity between every pair of queries in the query log is calculated using both the Doc-Doc and Weighted-Attribute similarity metrics. Since,

$$Sim_J(Q_1, Q_2) = Sim_J(Q_2, Q_1)$$

the similarity matrix is symmetric. Therefore we need to only compute the upper-half of the matrix. A minimal similarity threshold τ is used to prune the number of queries found similar to a query. The weight vector W, assigns a weight given to the similarity shown by each attribute while computing the Weighted-Attribute similarity. The sum of weights given to all the attributes is 1. Determining a good weighing scheme to accurately reflect the concerns of the user is a difficult task.

Algorithm 1 Creating Query Similarity Matrix

```
Require: Query log size - $N$, No of Attributes - $m$, Attribute Bags, Attribute Weights $W$, Similarity Threshold $\tau$
  BEGIN
  SimMatrix = null.
  for $i = 1$ to N-1 do
    iBags = Get_Attribute_Bags(i).
    for $j = i + 1$ to N do
      jBags = Get_Attribute_Bags(j).
      for $k = 1$ to m do
        iDoc = Append(iDoc,iBags[k]).
        jDoc = Append(jDoc,jBags[k]).
        AtSim[k] = $\frac{|iBags[k] \cap jBags[k]|}{|iBags[k] \cup jBags[k]|}$
      end for
      $Sim_{dd} = \frac{|iDoc \cap jDoc}{|iDoc \cup jDoc|}$
      If $(Sim_{dd} < \tau) Sim_{dd} = 0$.
      $Sim_{wa} = \sum_{k=1}^{m} AtSim[k] \times W[k]$
      If $(Sim_{wa} < \tau) Sim_{wa} = 0$.
      SimMatrix[i][j] = [$Sim_{dd}$, $Sim_{wa}$]
      SimMatrix[j][i] = SimMatrix[i][j].
    end for
  end for
  Return SimMatrix.
  END
```

4. EXPERIMENTS

4.1 Experimental Setup

To evaluate the effectiveness of our approach in answering imprecise queries, we set up a prototype database system that extends *BibFinder* [6]. *BibFinder* is a publicly-available Web data integration system, projecting a unified schema over multiple bibliography databases. BibFinder provides a formed-based interface and accepts queries over the relation

$$Publications(Author, Title, Conference, Journal, Year)$$

Several features of BibFinder validate the assumptions we make while developing our approach. Queries over BibFinder are conjuncts of attribute-value pairs. Even though BibFinder integrates multiple Web data sources with varying query capabilities, it displays the behavior similar to that of a relational database. BibFinder only returns tuples that exactly match the user query. Results generated by BibFinder contain values for all attributes in the relation. BibFinder can only access the underlying data using queries, hence any approach at answering imprecise queries that requires BibFinder to access all the tuples would not be feasible.

Our prototype system is designed over the architecture shown in Figure 1. We use the precise query optimizer and the query log of BibFinder to identify the queries from which to learn the similarity matrix. Table 2 lists the time taken and size of results produced at various stages of our algorithm. A Linux server running on Intel Celeron- 2.2 Ghz with 512Mb RAM was used to process the queries and to calculate the query similarities. We used 10000 queries from BibFinder's query log[1] in our prototype system.

Algorithm Step	Time	Size
SuperTuple Generation	126 sec	21 Mb
Similarity Estimation	10 hours	6.0 Mb

Table 2: Timing Results and Space Usage

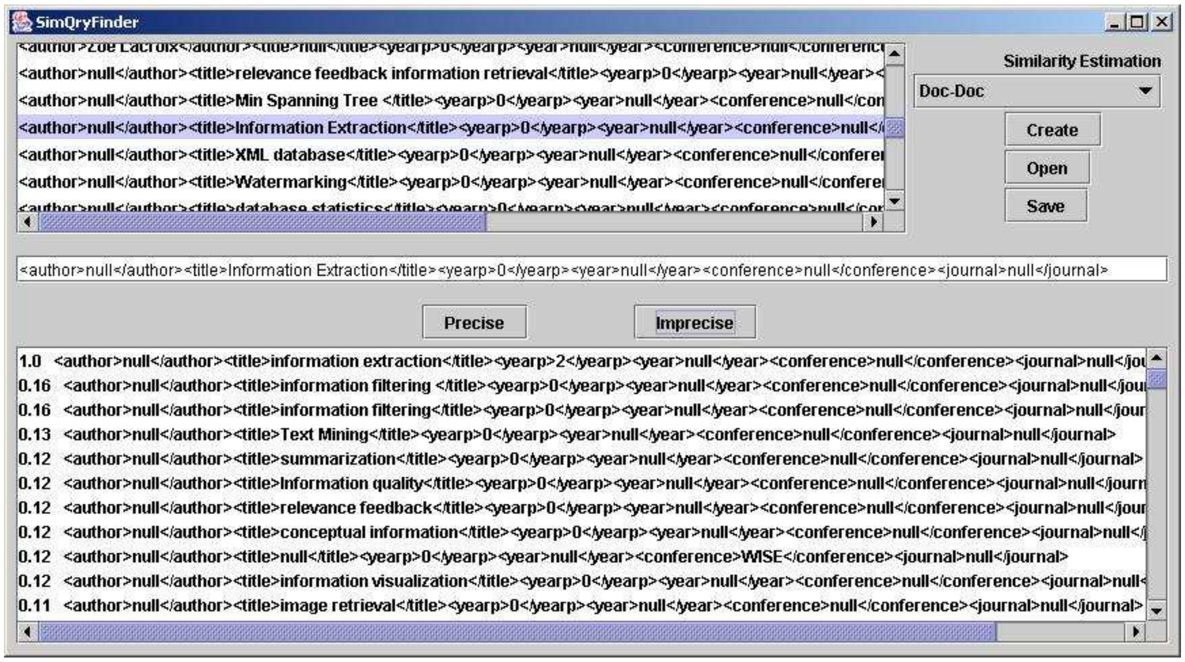
The complexity of the similarity estimation algorithm, Algorithm 1 is $O(N^2)$, where N is the number of queries in the query log. Hence the similarity matrix creation time is high, 10 hours, as we must compare each query with every other query in the query log as shown in Algorithm 1. We calculated both the *doc-doc* and *weighted-attribute* similarity between each pair of queries in the query log. To estimate the weighted-attribute similarity we assigned a weight of 0.3 for similarity over attribute Author, 0.4 for Title, 0.25 for Conference and 0.05 for year. We used a lower bound of 0.08 on the similarity between queries to reduce the number of similar queries. The same set of queries were used to compare the accuracy of similar queries identified by doc-doc similarity and weighted-attribute similarity.

To determine the correctness of the queries we suggest as being similar, we setup a user study. We asked 3 graduate students, who are frequent users of BibFinder to evaluate the relevance of the queries we suggest. Each student was provided with a GUI (see Figure 2) that allowed them to ask any query from among the 10000 queries in our prototype system. The GUI can execute both precise and imprecise queries. When a precise query is issued, only those tuples that exactly match the query are returned. For an imprecise query, a list of queries in descending order of similarity to the imprecise query is returned. The user can view the results of the related query by selecting the query. Each user was asked to pick 30 queries of their choice. For each imprecise query issued by the user, he/she had to determine how many among the *top*10 similar queries they considered similar. Also users were asked to report whether they found a query Q' similar to Q based on:

- Q' having related terms: The values binding attributes of Q' are related to the those in Q. E.g. term *e-learning* is relevant to *Web-based learning* and hence queries containing these terms would be considered similar to each other.
- Q' having related results: The results of Q' are relevant to Q, although no terms binding Q' are found relevant to Q. E.g. the query *Author="Jaiwei Han"* has results closely related to query *Title="Data Mining"* but the terms in the query itself are not related.

For the queries they found not relevant, the users were to describe the reason why they thought it was not relevant.

[1]BibFinder does not materialize the results of queries in its query log. Hence we extracted the results of the workload queries by probing BibFinder. Time to probe is approximately 1 hour if we exclude the delay between queries. The probing phase can be avoided by caching the results of the queries asked over the system.



```
SimQryFinder
<author>Zoe Lacroix</author><title>null</title><yearp>0</yearp><year>null</year><conference>null</conferenc
<author>null</author><title>relevance feedback information retrieval</title><yearp>0</yearp><year>null</year><
<author>null</author><title>Min Spanning Tree </title><yearp>0</yearp><year>null</year><conference>null</con
<author>null</author><title>Information Extraction</title><yearp>0</yearp><year>null</year><conference>null</
<author>null</author><title>XML database</title><yearp>0</yearp><year>null</year><conference>null</confere
<author>null</author><title>Watermarking</title><yearp>0</yearp><year>null</year><conference>null</confere
<author>null</author><title>database statistics</title><yearp>0</yearp><year>null</year><conference>null</con

Similarity Estimation: Doc-Doc
Create   Open   Save

<author>null</author><title>Information Extraction</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</journal>

Precise   Imprecise

1.0   <author>null</author><title>information extraction</title><yearp>2</yearp><year>null</year><conference>null</conference><journal>null</jou
0.16  <author>null</author><title>information filtering </title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</jou
0.16  <author>null</author><title>information filtering</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</jour
0.13  <author>null</author><title>Text Mining</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</journal>
0.12  <author>null</author><title>summarization</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</journal>
0.12  <author>null</author><title>Information quality</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</journ
0.12  <author>null</author><title>relevance feedback</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</jour
0.12  <author>null</author><title>conceptual information</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</j
0.12  <author>null</author><title>null</title><yearp>0</yearp><year>null</year><conference>WISE</conference><journal>null</journal>
0.12  <author>null</author><title>information visualization</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null<
0.11  <author>null</author><title>image retrieval</title><yearp>0</yearp><year>null</year><conference>null</conference><journal>null</journal>
```

Figure 2: User Interface

‘

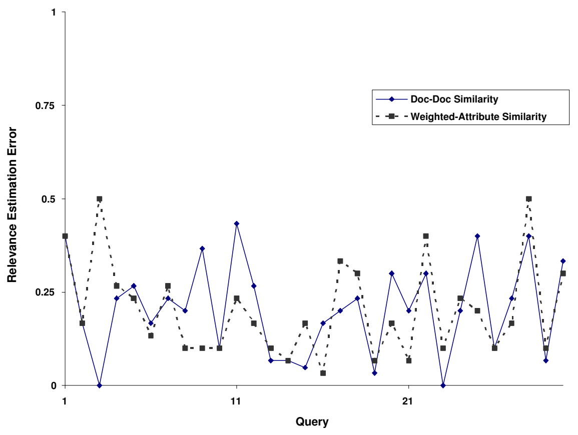


Figure 3: Error in Top-10 Estimation

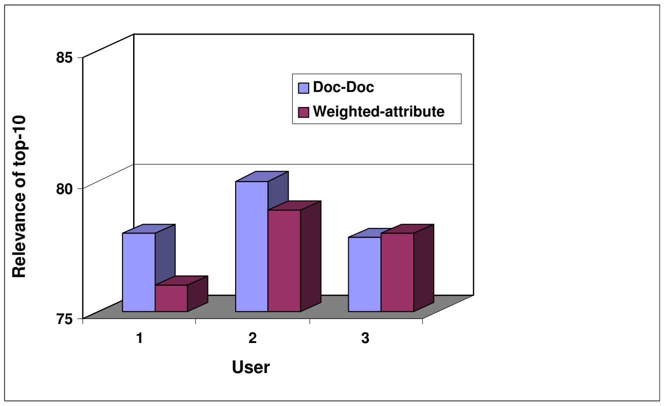


Figure 4: Precision of Top-10 Queries

4.2 Results of User Study

Table 3 contains a sample set of queries recommended as being similar to three imprecise queries. Figure 3 illustrates the error in estimating the top-10 queries to a user query. The error is calculated as the number of queries among the $top-10$ that were classified as not relevant by the user. The error is calculated as,

$$Error(Related(Q)) = \frac{|Not\_Relevant(Related(Q))|}{10}$$

$$= \frac{10 - Relevant(Related(Q))}{10}$$

$$= 1 - Precison(Related(Q))$$

Both doc-doc and weighted-average show less than 25% average loss of precision. The almost similar performance of doc-doc and weighted-average is surprising, even though weighted-attribute uses additional domain specific information in deciding the relevance. In fact, 2 out of 3 users found the weighted-attribute similarity to be less relevant than doc-doc as seen in Figure 4. This supports the hypothesis that converting the domain knowledge to a set of weights that capture the knowledge is a difficult task. The overall high relevance shown by both the approaches is encouraging. We believe that a larger corpus of queries would have improved the relevance numbers even further. Hence we plan to conduct a larger user study by extending BibFinder itself

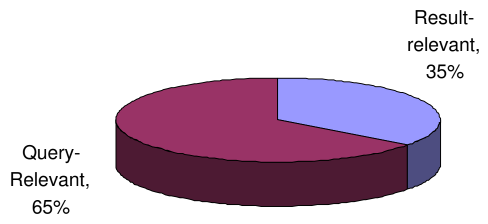


Figure 5: Relevance classification

Imprecise Query	**Title="web-based learning"**
Related Queries	0.19 Title="e Learning"
	0.16 Title="Web Technology"
	0.13 Conference="WISE"
Imprecise Query	**Title="Information Extraction"**
Related Queries	0.16 Title="information filtering"
	0.13 Title="Text Mining"
	0.12 Title="Relevance Feedback"
Imprecise Query	**Author="Abiteboul"**
Related Queries	0.3 Author="vianu"
	0.16 Author ="Dan Suciu"
	0.13 Author="Rakesh Agarwal"

Table 3: Relevant queries for 3 Imprecise queries

with the ability to support imprecise queries.

Figure 5 shows that on an average users found 65% of the queries that users found as relevant had related terms to the imprecise query they asked. For the remaining 35%, users had to execute the query and look at the results. Most similar queries that were classified as not relevant by users contained a widely used term present in the imprecise query or its answers. E.g. the query *Title="data warehouse"* is suggested as relevant to the query *Title="data integration"*, but users found it to be non relevant. This problem can be mitigated by giving weights to terms appearing in the query, with the common and widely used terms e.g. XML, data, mining etc getting lower weights. Information retrieval systems are faced with a similar problem of identifying query keywords that are rare. These systems use the IDF (inverse document frequency) value of keywords to identify and lower the weights to frequently occurring keywords.

5. RELATED WORK

Early approaches for retrieving answers to imprecise queries were based on theory of fuzzy sets. Fuzzy information systems [4] store attributes with imprecise values, like height= "tall" and color="blue or red", allowing their retrieval with fuzzy query languages. The WHIRL language [1] provides approximate answers by converting the attribute values in the database as vectors of text and ranking them using the vector space model. In [5], Motro extends a conventional database system to use data distances to interpret vague queries by adding a *similar-to* operator. The data metrics required by the similar-to operator must be provided by database designers. Binderberger [7] investigates methods to extend database systems to support similarity search and query refinement over arbitrary abstract data types. A number of similarity predicates and operators are defined and an algebra over ranked sets is presented. The similarity metrics to be used to compare various data types must be provided by the users of the system. In [2], Goldman et. al. propose to provide ranked answers to queries over Web databases but require users to provide additional guidance in deciding the similarity. Users must issue a *find query* and a *near query*. The find query is the one for which ranked results are desired. All the objects that are close to the set of objects returned by the near query are considered as the results for the find query. To make use of the proposed system users would have to know what objects in the database are closest to the objects they seek.

All the above systems, however are not applicable to existing databases since they assume non-standard data formats and require new operators to estimate the inter-tuple distance metrics. More specifically without re-modelling and restructuring the existing data and the operators, current approaches cannot be used on existing databases. Further they require large amounts of domain specific information either pre-estimated or given by the user of the query and may require access to all the tuples in the database to identify relevant queries. We proposed a domain independent approach to provide similar answers to a query that does not require the user to have in-depth knowledge of the system. Further, our solution does not warrant any re-modelling of the existing data or access to the entire database and hence is easy to implement over existing Web databases and Web mediation systems.

To control the cost of extracting related answers, we must reduce the overlap shown by related queries. Ideally we should execute the queries that will give high precision and are least overlapped with queries already executed. In [6], authors learn and use coverage and overlap statistics to execute a single query over multiple overlapping sources. On the other hand we execute multiple queries related to the imprecise query over a single database while keeping overlaps to a minimum. The difficulty is in maintaining a high relevance of the results while reducing overlaps.

6. DISCUSSION AND FUTURE DIRECTIONS

In our discussions in this paper we assume that a query is either completely precise or imprecise. But a user may want to issue a query where the imprecision is over a subset of the bound attributes, but has precise needs for the remaining attributes. In the current approach we do not consider such partially imprecise queries. Both precise and imprecise queries are represented as a conjunction of attribute value pairs. The user decides whether the query is precise or imprecise based on her need for ranked answers. We can easily extend our system to support partially imprecise queries. Given any partially imprecise query Q' we can split its bound attributes into two distinct sets: precisely bound and imprecisely bound. Using these two sets of attributes we can create two new queries, one completely precise and the other completely imprecise, each more general than Q'. The answer tuples satisfying Q' can be found by executing the precise general query over the resultset of the imprecise general query.

Only imprecise queries that are present in the query log or map to some query in the query log are answered by our system. Moreover to answer an imprecise query we need a corresponding precise query that has a non-empty resultset. We plan to extend our approach to answer queries that does not map to any query in the query log. A possible solution is to use co-occurrence analysis and scalar clustering techniques to identify terms that are related to terms appearing in the imprecise query [9]. Given an imprecise query Q over relation R, we can identify another imprecise query Q' from the query log whose terms are closest those of Q. The queries related to Q' can then be used to find queries related to Q. Another solution is to execute Q as a precise query over R and to identify rare keywords appearing in the answerset. Assuming we created a similar mapping for all the queries in the query log, we can measure the similarity between Q and other queries as the number of common keywords.

Large databases may contain thousands of queries in its query log. BibFinder itself contains over $30,000$ queries in its query log. Hence reducing the time for similarity estimation is of utmost importance. Figure 6 presents the frequency distribution of queries over BibFinder. 10% of the keyword queries were asked 60% of the time. Thus we can considerably reduce the time to estimate the similarity matrix by only calculating the similarity for only the frequently asked queries. Thus the algorithm complexity would become $O(K \times N)$, where $K << N$. The number of queries can be reduced even further by removing queries that were not recently issued over the system. Since queries with too few tuples would only be similar to small number of queries we can remove them by keeping a lower limit on the selectivity of queries stored in the query log. Reducing the queries used for comparison, might reduce the precision of our results.

7. CONCLUSION

In this work, we introduced a new approach for answering imprecise queries over a database. We use an information retrieval based approach to find the similarity among queries and use it to identify relevant precise queries to a given imprecise query. To evaluate the effectiveness of our approach, we performed experiments over a real Web database system, BibFinder. The experiments indicate that the proposed approach is able to answer imprecise queries with high levels of user satisfaction. The approach can be (and has been) implemented without affecting the internals of a database thereby showing that it could be easily implemented over many of the existing databases.

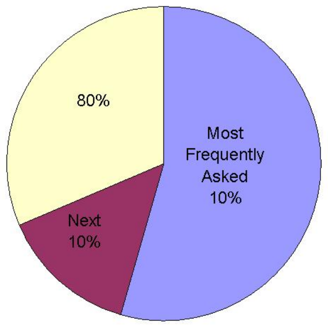


Figure 6: Query frequency distribution (taken from [8])

8. REFERENCES

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

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

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

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

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

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

[7] Micheal Ortega-Binderberger. *Integrating Similarity Based Retrieval and Query Refinement in Databases.* PhD thesis, UIUC, 2002.

[8] Z. Nie and S. Kambhampati. *A Frequency Based Approach for Mining Coverage Statistics in Data Integration.* to appear in ICDE 2004.

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