ANSWERING IMPRECISE QUERIES OVER AUTONOMOUS DATABASES

by

Ullas Nambiar

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

December 2005

ANSWERING IMPRECISE QUERIES OVER AUTONOMOUS DATABASES

by

Ullas Nambiar

has been approved

August 2005

APPROVED:

____, Chair

____

____

____

____

Supervisory Committee

ACCEPTED:

____

Department Chair

____

Dean, Division of Graduate Studies

ABSTRACT

Current approaches for answering queries with imprecise constraints require users to provide distance metrics and importance measures for attributes of interest - metrics that are hard to elicit from lay users. Moreover they assume the ability to modify the architecture of the underlying database. Given that most Web databases are autonomous and may have users with limited expertise over the associated domains, current approaches for answering imprecise queries are not applicable to autonomous databases such as those accessible on the Web.

This dissertation presents *AIMQ* - a *domain independent* framework for supporting imprecise queries over autonomous databases with *minimal input* from users and *no modifications* to the underlying database. AIMQ provides answers satisfying an imprecise query by identifying and executing a set of precise queries similar to the imprecise query. AIMQ begins by mapping the given imprecise query to a precise query with non-null resultset. Then using a approximate functional dependency (AFD) based query relaxation heuristic AIMQ identifies a set of precise queries similar to the initially mapped precise query.

AIMQ measures the similarity between a tuple and the imprecise query as the weighted summation of similarity over attributes. However, measuring similarity requires distance metrics to be provided by the users or an expert; often quite difficult even for experts. Hence, a *context-sensitive domain-independent semantic similarity estimation* technique has also been developed as part of AIMQ.

Results of empirical evaluation conducted using multiple real-life databases demonstrate both the domain independence and the efficiency of AIMQ's learning algorithms. User study results presented in this thesis demonstrate the high relevance of answers given by AIMQ. AIMQ is the only domain independent system currently available for answering imprecise queries over autonomous databases. It can be (and has been) implemented without affecting the internals of a database or requiring extensive domain specific inputs from the user, thereby demonstrating that AIMQ can be implemented over any autonomous database.

To

Lakshmi, Mom and Dad

for their unconditional love, unwavering support and unlimited patience.

ACKNOWLEDGMENTS

No one who achieves success does so without acknowledging the help of others.

The wise and confident acknowledge this help with gratitude.

- *Alfred North Whitehead*, Mathematician & Philosopher (1861-1947)

The past few years of my life have been memorable, both professionally and personally, due to the interactions I have had with a number of people. Among them, the person who has most influenced my life is my advisor, Subbarao (Rao) Kambhampati. I am deeply grateful to Rao for all the criticisms and suggestions on the early ideas that lead to my defining the scope of this thesis. He showed immense patience during the long and frustrating problem identification phase of my research. When my interests lead me away from his areas of research, he wholeheartedly supported my search for topics which eventually resulted in this thesis. He always found time to review drafts usually given just hours before the deadline and to provide constructive criticism. I am grateful for his insistence on pursuing only the most critical problems and comparing only with the best in the field; a factor that has and will continue to motivate me to aim high.

I would also like to thank the other members of my committee, Selcuk Candan, Huan Liu and Hasan Davalcu for valuable advice and encouragement. I would specially like to acknowledge my external committee member, Gautam Das of University of Texas, Arlington for giving many insightful suggestions and for taking time from his busy schedule to participate in my dissertation. Many thanks to Kevin Chang of UIUC for providing helpful comments. I also thank my industry mentors, Wen

Syan Li and Prasan Roy for providing exciting research opportunities and invaluable advice.

My special thanks to Zaiqing Nie for being a friend and collaborator. I will always cherish the long discussions I had with him at Gold Water Center. Special thanks also to Biplav Srivastava for his invaluable advice and occasional long-distance mentoring. I also thank Zoe Lacroix for introducing me to the world of XML databases. A special thanks to Romeo Sanchez for providing emotional support and much cherished friendship.

To my friends, Thomas Hernandez, Menkes van den Briel, Binh Minh Do, Xuanlong Nguyen, Tuan Le, Lei Yu, Jianchun Fan, and all other members of the AI lab, thanks for their invaluable companionship. I also acknowledge all the AI lab members who volunteered to take part in user studies that form an important part of this thesis.

I am and will always be very grateful to my parents, Bhargavi and Balan, who have supported me with their love and encouragement throughout my education. My most sincere thanks to my closest friend and fiance, Lakshmi, who came into my life early on during my Ph.D and has lovingly and patiently supported me since then.

My research was mostly supported by the ASU ET-I^3 initiative grant ECR A601.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Everything is vague to a degree (that) you do not realize till you have tried to make it precise.

- Bertrand Russell

The Philosophy of Logical Atomism

The rapid expansion of the World Wide Web[1] has made a variety of autonomous 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 *Web databases*[2] (450,000 and growing [CHL+04]) has brought about a drastic change in their average user profile from tech-savvy, highly trained professionals to lay users demanding "instant gratification". Moreover, most users on the Web now prefer the easy-to-use keyword based query interface and the ranked retrieval model used by *search engines*. However, unlike Web search engines that take a few keywords, look up the index and provide a listing of best-matching

[1]The World Wide Web (the "Web" or "WWW" for short) is a hypertext system that operates over the Internet - a publicly available internationally interconnected system of computers and services provided by them.

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

web pages (usually based on relevance to keyword query combined with popularity of the web page), the Web database systems expect users to know the name of relation to query, the field to look in, and at times even the field type. Moreover, database query processing models have always assumed that the *user knows what she wants* and is able to formulate a query that accurately expresses her needs. Therefore, database systems have always used *a boolean model of query processing where there is a set of answer tuples that exactly satisfy all the constraints of the query and thus are equally relevant to the query.* Hence, to obtain a satisfactory answer from a Web database, the user must formulate a query that accurately captures her information need; often a difficult endeavor. Often users must reformulate their queries a number of times before they can obtain a satisfactory answer. Thus, a lack of knowledge about the schema and contents of the database combined with the boolean query model can often result in the user not obtaining satisfactory answers from a database.

Although users may not know how to phrase their queries, they can often tell which tuples are of interest to them when presented with a mixed set of results having varying degrees of relevance to the query. Thus database query processing models must embrace the IR systems' notion that *user only has vague ideas of what she wants*, is unable to formulate queries capturing her needs and would prefer getting a ranked set of answers. This shift in paradigm would necessitate supporting *imprecise queries*[3]- queries that only require the answer tuples to match the constraints closely and not exactly. This sentiment is also reflected by several database researchers in a recent database research assessment [Low03].

[3] *A formal definition is given in Chapter 3*

1.1. Imprecision in Database Systems

A query against incomplete or imprecise data in a database, or a query whose search conditions are imprecise can both result in answers that do not satisfy the query completely. Such queries can be broadly termed as *imprecise queries.* Eventhough there has recently been much interest in looking at problems arising in storing and retrieving data that is incompletely specified (hence imprecise), such systems have not gained widespread acceptance yet. The popular querying and data storage models (both in database and IR systems) still work with data that is precise. Hence, in this thesis, *we only focus on the problem of supporting queries with imprecise constraints over databases containing precise data and supporting the boolean query answering model.*

1.1.1. Why Support Imprecision in Queries?

Todays database systems are designed largely for *precise queries* against a database of *precise and complete* data. Range queries (e.g., *Age BETWEEN 20 AND 30*) and disjunctive queries (e.g., *Name="G. W. Bush" OR Name="George Bush"*) do allow for some imprecision in queries. However, these extensions to precise queries are unable to completely capture the expressiveness of an imprecise query. We use the following two examples to motivate the need for supporting imprecise queries over databases.

Example 1.1: *Suppose, we wish to express the query -Find a person whose last name is* `like Napalu`*, is* `perhaps middle aged`*, and who drives an* `old white-ish car` *with license plate that* `contains TR`*, over a demographics database. We* *can represent the query formally as*

*Q:- Demographics(LastName like Napalu, Age like 50, VehicleColor like White, LicensePlate like ''*TR*'')*

Note that for simplicity, we have used the relation `like` *to represent all similarity relationships such as* `white-ish, contains etc`*. Also we have used some domain-knowledge in converting* `perhaps middle-aged` *to* `Age like 50`*. While we can assume lay users to be able to do such trivial transformations, we cannot expect them to come up with alternate formulations of the above query that can be expressed using precise queries, such as range and disjunctive queries, and thus extract relevant answers from existing databases. In fact, we believe the task would be quite difficult even for experts.*

Example 1.2: *Suppose a user wishes to search for* sedans *priced* around $10000 *in a used car database,* `CarDB(Make, Model, Year, Price, Location)`*. Based on the database schema the user may issue the following query:*

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

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

In the above example, Example 1.2, the query processing model used by CarDB would not suggest the *Accords* or the slightly higher priced *Camry* as possible answers of interest as the user did not specifically ask for them in her query. This will force the user to enter the tedious cycle of iteratively issuing queries for all "similar" models before she can obtain a satisfactory answer. This is further complicated by the fact that in many cases the user may not know what the similar models are to begin with.

One way to automate this is to provide the query processor information about similar models (e.g. to tell it that *Accords* are 0.9 similar to *Camrys*). While such approaches have been tried, their achilles heel has been the acquisition of such domain specific similarity metrics–a problem that will only be exacerbated as the publicly accessible databases increase in number.

1.1.2. Difficulty in Adapting Current Database Systems. Supporting imprecise queries over databases necessitates a system that integrates similarity search paradigm over structured and semi-structured data. Todays relational database systems, as they are designed to support precise queries against precise data, use such precise access support mechanisms as indexing, hashing, and sorting. Such mechanisms are used for fast selective searches of records within a table and for joining two tables based on precise matching of values in join fields in the tables. The imprecise nature of the search conditions in queries will make such access mechanisms largely useless. Thus, supporting imprecise queries over existing databases would require adding support for imprecision within the query engine and meta-data management schemes like indexes. Moreover, we will require access to domain specific metadata, e.g. an *object thesaurus* that provides all possible synonyms for various objects in the domain, descriptions about characteristics of the objects along with rules for matching similar names and descriptions. Recent surveys [NH97, KS05] of existing attempts at ontology design show that there is great diversity in the way ontologies are designed and in the way they represent the real-world facts. Thus, much work needs to be done before mature, usable and universally acceptable ontologies are made available. Therefore, at this time a solution based on adding domain ontologies to existing databases cannot be considered feasible.

Another challenge we must overcome is that the database is remotely located and will be autonomous in its behaviour i.e. the database may undergo frequent updates and may not be willing to support imprecise queries. Supporting imprecise queries would involve changing the query processing and data storage models being used by the database. Such a transformation would be a time consuming and costly procedure and may affect a number of other systems that use the database. For example, changing an airline reservation database will necessitate changes to other connected systems including travel agency databases, partner airline databases etc. Hence, assuming that Web databases will themselves be inclined to support imprecise queries would be a fallacy. Therefore, we must contend with being able to access the underlying data by using the existing query interfaces i.e. by issuing precise queries.

In fact, the problem of answering imprecise queries is equally difficult even if we assume that the query processing framework of the underlying database can be modified to support imprecise queries. Even if the database is modifiable, we would still require a domain expert and/or end user to provide the necessary distance metrics and a domain ontology. Domain ontologies do not exist for all possible domains and the ones that are available are far from being complete. Even if we were to assume availability of distance metrics and domain ontologies (given by some expert) that is true over a given database, most real-life databases undergo frequent updates and therefore the provided metrics and relationships will also have to be updated frequently - a non-trivial endeavour by itself. Thus, the problem of answering imprecise queries over local databases also brings forth most of the challenges that we have when looking at autonomous sources.

Based on the above discussions, we can conclude that a feasible solution for answering imprecise queries should neither assume the ability to modify the properties of the database nor require users (both lay and expert) to provide much domain specific information. Therefore, in this thesis, *our focus is enabling support for imprecise queries without changing the behaviour of the existing database and with minimal input (the imprecise query and similarity threshold) from the users.* We assume the databases are autonomous in nature and support the boolean query model where both queries and data are precise. The solution we propose in this thesis is a middle-ware that sits between the user issuing the imprecise query and the database that only supports precise queries.

1.2. Outline of the Thesis

Below we formally define the problem at hand, list the key challenges faced in answering imprecise queries and then motivate and describe the solution we present in this dissertation.

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

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

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. □

1.2.2. Challenges. Supporting imprecise queries over autonomous Web databases brings forth the following challenges:

Supporting Imprecision: 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). Thus, to support an imprecise query Q over a database R (that supports the boolean query answering model), we require the ability to identify all tuples of R that are similar to Q. A naive solution would be to compare each tuple in the relation R against the query. But accessing all tuples of R to answer each imprecise query is neither feasible nor practical. Since, only a query based access to the tuples of the database is available and the underlying query answering model is binary, extracting tuples from R necessitates probing the database using precise queries whose answers fall in the neighbourhood of interest i.e. are highly relevant to Q.

Techniques like query relaxation and generalization [CCL91, CCL92, Mot90] have been attempted by researchers to generate new queries related to the user's original query. The new queries can then be used to find answers which may be of interest to the user but not in the scope of the original query. However the abstraction and refinement rely on the database having explicit hierarchies of the values in the domain. A generalized query is created by replacing values in the given query with corresponding values higher up in the hierarchy while replacing with values lower in the hierarchy gives a refined query. Since, as shown earlier in the section, domain specific object hierarchies generally known as ontologies are often not available and when available are far from being complete, we consider the query generalization approach as not a promising approach to solving the imprecise query answering problem.

On the other hand, query relaxation is very much a feasible solution but requires a initial query with a number of attributes bound. Assuming we are able to derive such a query from the imprecise query, further relaxation would still bring up the problem of what to relax first. Depending on the attributes relaxed we may retrieve few or many results with varying relevance. Randomly creating precise queries may result in queries that have too few answers or have answers that are not relevant to the imprecise query Q. In either case we may end up creating and executing a large number of precise queries thereby increasing cost of answering the query. Therefore to efficiently answer an imprecise query, we must create and execute only those precise queries that are likely to return answers relevant to Q i.e. only those Q' who answers have high precision. Precision of Q' is the fraction of answers that are relevant to Q.

$$Precision(Q') = \frac{|Answers(Q) \bigcap Answers(Q')|}{|Answers(Q')|}$$

Estimating Query-Tuple Similarity: A database system supporting imprecise queries must provide information about how close an answer tuple is to the given imprecise query. Two tuples (a selection query can be seen as a tuple with few missing values) are considered similar if they have syntactical similarity (e.g. same subset of attributes are bound in both queries, stems of a common word bind an attribute, etc) or if the binding values are *semantically similar*.

Semantic similarity between two values is the similarity perceived by the user. Since our motivation is to provide answers that are acceptable to the users, we must use semantic similarity between values binding the query and the tuples to decide the relevance of the answers. Semantic similarity, also called *semantic closeness/proximity/nearness*, is a concept whereby a set of words (attribute values) are assigned a metric based on the closeness of their meaning. An intuitive way of displaying terms according to their semantic similarity is by grouping together closer related terms and spacing more distantly related ones wider apart. This is commonly achieved by using ontologies. However, as mentioned earlier, ontologies describing relationships between all possible concepts for every domain is not available. Thus, developing a domain independent solution for measuring the semantic similarity between the query and the answer tuples becomes vital to answering imprecise queries over autonomous databases.

Measuring Importance of an Attribute: Often users would like to see only the *top-k* answers to a query. 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. However, not all attributes may be equally important in determining the similarity of the answer to the query. Specifically, a measure of importance for the similarity shown over any attribute in the context of a given query may be necessary to determine the best k matches. 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 that of computing the importance to be ascribed to an attribute.

1.2.3. Motivation behind our approach. The problem of supporting imprecise queries has already attracted considerable interest from researchers including those in fuzzy information systems [Mor90], cooperative query answering [JWS81, Jos82, Mot86] and query generalization [CCL92, Mot90]. More recent efforts have focussed at supporting imprecise queries over relational databases by introducing abstract data types and extending the query processor with similarity functions [OB03, GSVGM98, ABC$^+$02] (Chapter 9 has more details). However, all the proposed approaches for answering imprecise queries require large amounts of domain specific information either pre-estimated or given by the user of the query. Unfortunately, such information is hard to elicit from the users. Further some approaches require changing the data models and operators of the underlying database. Recently much work has been done on providing ranked answers to queries over a relational database [BGM02, FLN01, IAE03]. However, they assume complete access to the indexes of the databases. To summarize, the solutions attempted so far require both extensive input from the users and the capability to change the underlying database - requirements that are difficult to satisfy given the autonomous nature of the databases on the Web and the limited expertise of their users.

This is the motivation for the *AIMQ system* [NK05, NK04b, NK04a, NK03] - a domain independent solution for supporting imprecise queries over autonomous Web databases that we present in this thesis. Rather than shifting the burden of providing the value similarity functions and attribute orders to the users, we propose a domain independent approach that requires no additional input from the users and does not necessitate any changes to be made to the underlying database. Our motivation is to mimic the interfaces provided by search engines, in that the users need only provide the queries and should get ranked answers that satisfy their needs. Our solution is a paradigm shift that unites the database and information retrieval technologies: it brings the similarity searching/ranked retrieval paradigm from IR systems into the structured, type-rich access paradigm of databases, thereby enabling the database systems to support flexible query interfaces. Thus unlike the relational database systems we retrieve answers that are ranked according to the degree of relevance to the user query. The degree of relevance of an answer to a query is automatically estimated using domain-independent similarity functions that can closely approximate the subjective interpretation of the user.

Specifically, our intent is to mine the semantics inherently present in the tuples (as they represent real-world objects) and the structure of the relations projected by the databases. Our intent is not to take the human being out of the loop, but to considerably reduce the amount of input she has to provide to get a satisfactory answer. In short, we wish to test *how far we can go (in terms of satisfying users) by* *using only the information contained in the database? How closely can we model the user's notion of relevance by using only the information available in the database?* In fact, our effort should be seen as being similar in spirit to that of Web search engines which try to model the user's notion of relevance of a document to the given keyword query based on information such as link structure, authorities, hubs etc. that is mined from the corpus itself.

1.2.4. Contributions. In response to the above challenges, we propose the query processing approach, *AIMQ*, that integrates techniques from IR and database research to efficiently determine answers for imprecise queries over autonomous databases supporting a boolean query processing model.

AIMQ: Given an imprecise query, AIMQ begins by deriving a precise query (called base query) that is a specialization of the imprecise query. Then to extract other relevant tuples from the database it derives a set of precise queries by considering each answer tuple of the base query as *a relaxable selection query.*[4] Relaxation involves extracting tuples by identifying and executing new queries obtained by reducing the constraints on an existing query. However, randomly picking attributes to relax could generate a large number of tuples with 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. Such relationships are captured by approximate functional dependencies (AFDs). Therefore, AIMQ makes use of AFDs between attributes to

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

determine the degree to which a change in the value of an attribute affects other attributes. Using the mined attribute dependency information AIMQ determines the importance of each attribute and derives 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 both for efficient query relaxation and measuring similarity of answers). Hence, the *first contribution of this dissertation is a domain independent approach for learning attribute importance.*

The tuples obtained after relaxation must be ranked in terms of their semantic similarity to the query. While we can by default use a L_p distance metric[5] such as Euclidean distance to capture similarity between numerical values, no such widely accepted measure exists for categorical attributes. Therefore, the *second contribution of this dissertation is an association based domain independent approach for estimating semantic similarity between values binding categorical attributes.*

Advantages of the developed framework are presented by applying it in the context of *two real-life* datasets: *(1) Yahoo Autos and the (2) US Census Dataset from UCI Machine Learning Repository.*

AIMQ-Log: The AIMQ system's primary intent was minimizing the inputs a user has to provide before she can get answers for her imprecise query. However, in doing so, AIMQ fails to include users' interest while deciding the answers. A naive solution would be to ask user to provide feedback about the answers she receives. But doing so would negate the benefits of AIMQ. The ideal solution would be to obtain and use user

[5]AIMQ uses a L_1 distance metric or Manhattan distance to capture similarity between numeric values.

feedback implicitly. Database workloads - log of past user queries, have been shown as being a good source for implicitly estimating the user interest [ACDG03]. In a way, this may be viewed as a poor mans choice of relevance feedback and collaborative filtering where a users final choice of relevant tuples is not recorded. Despite its primitive nature, such workload information can help determine the frequency with which database attributes and values were often requested by users and thus may be interesting to new users.

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

AIMQ.

Advantages of AIMQ-Log are presented by applying it in the context of the *online bibliographic data source, BibFinder.*

1.3. Expected Impact

This dissertation enables autonomous databases to efficiently answer imprecise queries with minimal user guidance and without modifying the existing database. The solutions developed build on popular techniques developed in IR and database communities to estimate the distance between values of categorical attributes, to automatically determine the order in which to relax the attributes and to support a ranked retrieval model over databases.

As described earlier, often the lay users of autonomous database systems are unable to precisely express their query. We found this to be especially true in our experience with archaeological (KADIS, [Kin04]) and biological sources (BioHavasu, [HK04]), where we need to deal with a broad range of scientific users, many of whom lack a specific knowledge to frame precise queries. Supporting imprecise queries over such sources would greatly enhance the ability of end users to efficiently extract critical information necessary for their research. Existing approaches for supporting similarity search over databases are not applicable here as we do not have access to internals of the data sources, and the users may not be able to articulate domain specific similarity metrics. A recent report on improving Homeland Security [Kim02] also points out the need for supporting imprecise queries over databases for efficient extraction of critical information.

The AIMQ system presented in this thesis, being domain independent and easily implementable over existing databases, would be quite useful in supporting imprecise queries in many scenarios such as those mentioned above.

1.4. Thesis Organization

Chapter 2 starts by briefly reviewing the theory of query processing in relational databases and information retrieval systems. We also look at several recent attempts to combine techniques from databases and information retrieval systems.

Chapter 3 explains the data and query models used in this dissertation and gives an overview of the AIMQ approach for answering imprecise queries. Formal definitions of precise and imprecise queries are provided. Also the query-tuple similarity estimation model used by AIMQ is described.

Chapter 4 presents the semantic similarity estimation approach developed as part of the AIMQ system. The chapter begins by pointing out that semantic similarity is context sensitive. Then the IR style model of identifying the context of a value based on the associated values (features) is presented. A weighted attribute model for estimating similarity is described.

Chapter 5 describes how AIMQ uses approximate functional dependencies between attributes to guide the query relaxation process. We describe the process of extracting a representative sample of the database for mining the dependencies using probing queries and also highlight possible affects of sampling on our solution.

Chapter 6 presents results showing the efficiency and effectiveness of the AIMQ system in answering imprecise queries. Specifically, we investigate the robustness of the estimated attribute importance and value similarities, evaluate the efficiency of our query relaxation process and verify the relevance of answers we suggest by conducting a user study.

Chapter 7 compares the efficiency and accuracy of AIMQ with a system providing similar answers by using the ROCK categorical value clustering algorithm. User study results comparing the relevance of the answers provided by both the systems are presented.

Chapter 8 describes how AIMQ-Log, a system that extends AIMQ by implicitly adding user relevance to the imprecise query answering model of AIMQ. We describe how AIMQ-Log identifies the set of precise queries relevant to the given imprecise query from a workload of the database. Results of a user study showing the relevance of the identified queries and the extracted answer tuples are presented.

Chapter 9 briefly describes several recent research efforts that have attempted to integrate IR and database techniques and/or tried to ease the difficulties faced by lay users when trying to extract information from a database.

Finally, in Chapter 10, we summarize our contributions and discusses potential extensions for the techniques developed in this dissertation.

CHAPTER 2

BACKGROUND

This section discusses work that is relevant and/or has influenced the ideas put forth by this thesis. By nature this thesis is interdisciplinary as it adapts techniques from information retrieval research into database query processing. Hence we begin by doing a high level overview of the underlying theory in (1) Relational database systems, (2) Web data integration and (3) Information Retrieval. Then we briefly look at some existing systems that (4) merge database and information retrieval systems.

2.1. Relational Database Systems

A *database* is an information set with a regular structure. Databases resembling modern versions were first developed in the 1960s. We can easily classify databases by the programming model associated with the database. Historically, the hierarchical model was implemented first, then came the network model and finally ever-popular, relational model[1]. Databases based on the relational model became known as *relational databases*. The relational data model permits the designer to create a consis-

[1]The object-oriented data model is a more expressive extension of the relational model but has not been widely accepted [CD96].

tent logical model of the information to be stored. This logical model can be refined through a process of database normalization. The basic relational building block is the *domain or data type.* A *tuple* is a set of *attributes*, which are ordered pairs of domain and value. A *relation* is an unordered set of tuples. Although these relational concepts are mathematically defined, they correspond loosely to traditional database concepts. A relation is similar to the traditional concept of table. A tuple is similar to the concept of row. To perform queries under this model, two mathematically equivalent paradigms exist:

Relational Calculus is a purely declarative means of specifying the desired result.

Relational Algebra is a procedural language consisting of a set of (unary and binary) operators that are applied to tables.

The Structural Query Language (SQL) to support relational systems is largely based on relational calculus, although it incorporates several aspects of relational algebra operators.

2.2. Multi Database Systems

All computer systems have limits. These limitations can be seen in the amount of memory the system can address, the number of hard disk drives which can be connected to it or the number of processors it can run in parallel. In practice this means that, as the quantity of information in a database becomes larger, a single system can no longer cope with all the information that needs to be stored, sorted and queried. Although it is possible to build bigger and faster computer systems, a more affordable solution is to have several database servers that appear to the users to be a single system, and which split the tasks between themselves. These are called *distributed databases*, and have the common characteristics that they are stored on two or more computers, called nodes, and that these nodes are connected over a network.

2.2.1. Distributed Databases. Depending on the level of autonomy given to each of the component databases, we can further classify distributed databases into two subsets.

- *Homogeneous databases*: Homogeneous databases all use the same DBMS software and have the same applications on each node. They have a common schema (a file specifying the structure of the database), and can have varying degrees of local autonomy. Local autonomy specifies how the system appears to work from the user's and the programmer's perspective. For example, we can have a system with little or no local autonomy, where all requests are sent to a central node, called the gateway. From here they are assigned to whichever node holds the information or application required. It has the disadvantage that the gateway into the system has to have a very large network connection and a lot of processing power to keep up with requests and routing the data back from the nodes to the users.
- *Heterogeneous databases*: At the other end of the scale we have heterogeneous databases, which have a very high degree of local autonomy. Each node in the

system has its own local users, applications and data and dealing with them itself, and only connects to other nodes for information it does not have. This type of distributed database is often just called a *database federation system* [SL90]. Such systems are also known as *database mediation systems* due to their use of a mediation component that connects the various heterogeneous databases.

Database mediation systems are popular due to their scalability, reduced cost in adding extra nodes, and the ability to include different database management systems in the system. This makes them appealing to organizations since they can incorporate legacy systems and data into new systems. The rapid spread of the Internet and the WWW has opened up new opportunities to make databases accessible via the web (Web-enabled databases). Further, federated systems can now be derived by connecting autonomous databases over the Web. In particular, this thesis focuses on supporting queries over web-enabled databases and mediation systems integrating them. Therefore, in the following we will look at mediation systems over web-enabled databases popularly known as *data integration systems* [CGMH$^+$94, ACPS96, TRV98, HKWY97, LRO96, KLN$^+$04, KNNV02].

2.2.2. Data Integration. A data integration system is an automated method for querying across multiple heterogeneous databases in a uniform way. In essence, a mediated schema is a uniform set of relations serving as the domain of the application and is used to provide the user with a uniform interface to a multitude of heterogeneous data sources that store the actual data. Figure 1 shows the architecture of a

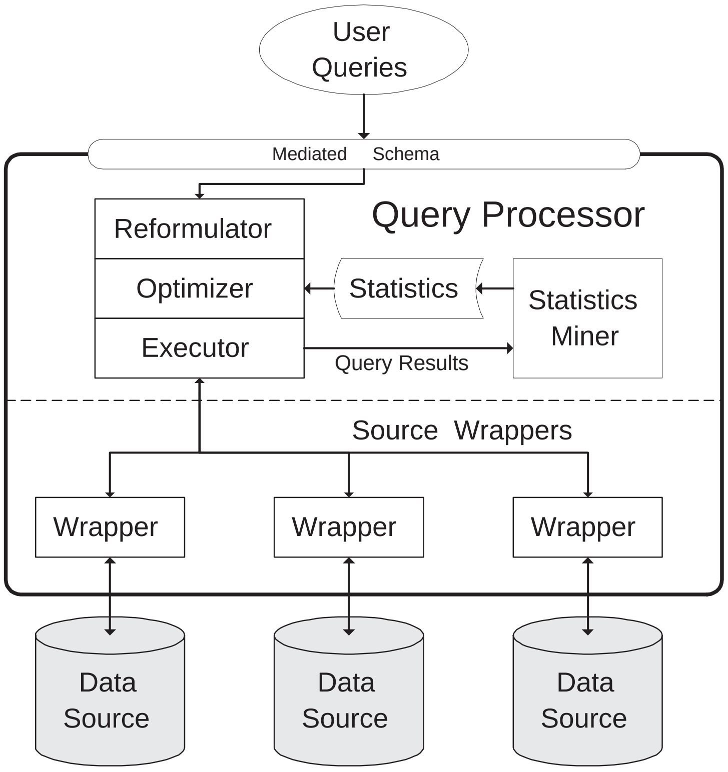


Figure 1. Data Integration System (taken from [Nie04])

data integration system. In a data integration system, the user asks a query over the mediated schema and the data integration system reformulates this into a query over the data sources. The query optimizer will find a high-quality query plan using the necessary statistics obtained from the statistics engine. The query executor will then execute the plan by calling the wrappers for the integrated data sources.

Data integration systems can be further classified depending on whether the sources are cooperating in the integration process. Systems such as Garlic [HKWY97], TSIMMIS [CGMH$^+$94] and HERMES [ACPS96] assume that databases are "aware of" and participate in building the integration system. At the opposite end are systems like Information Manifold [LRO96], DISCO [TRV98], Emerac [KLN$^+$04, LKG99] and Havasu [KNNV02, NKH03] which integrate databases that are autonomous and do not provide any support in forming the integrated system.

2.3. Information Retrieval

Information retrieval (IR) is the art and science of searching for information from free-form natural language text. An alternate definition of IR as a process of identifying *unstructured* records satisfying a user query, is more suitable to the work done in this thesis. Traditionally IR systems refer to a record as a *document* and an organized repository of documents as a *collection*. This section gives a background on modern IR [BYRN99] briefly exploring models in IR and the criteria used to compute similarity between documents.

2.3.1. Goal of Information Retrieval.

IR focuses on retrieving documents based on the content of their unstructured components. Documents are represented as a collection of features (also called "terms"). An IR request (typically called a "query") may specify desired characteristics of both the structured and unstructured components of the documents to be retrieved (e.g. The documents should be about "Information retrieval" and their author must be "Smith"). In this example, the query asks for documents whose body (the unstructured part) is about a certain topic and whose author (a structured part) has a specified value. IR typically seeks to find documents in a given collection that belong to a user given topic or that satisfy an information need as expressed by the query. Documents that satisfy the given query in the judgment of the user are said to be *relevant.* An IR engine may use the query to classify the documents in a collection (or in an incoming stream), returning to the user a subset of documents that satisfy some classification criterion. Naturally, the higher the proportion of documents returned to the user that she judges as relevant, the better the classification criterion. Alternatively, an IR engine may "rank" the documents in a given collection. To say that document D_1 has higher ranking than document D_2 with respect to a given query Q may be interpreted as D_1 is more likely to satisfy Q than D_2.

Traditionally, success of an IR system has been evaluated using two popular measures, both based on the concept of relevance of answers to a given query. They are:

- *Precision*: Precision is defined as the ratio of relevant items retrieved to all items retrieved, or the probability given that an item is retrieved, it will be relevant [BYRN99]. Measuring precision is easy; if a set of competent users or judges agree on the relevance or non-relevance of each of the retrieved documents, then calculating the precision is straightforward. Of course, this assumes that the set of retrieved documents is of manageable size, as it must be if it is to be of value to the user. If the retrieved documents are ranked, one can always reduce the size of the retrieved set by setting the threshold higher (e.g., only look at the top 100, or the top 20).

- *Recall*: Recall is defined as the ratio of relevant items retrieved to all relevant items, or the probability given that an item is relevant, it will be retrieved. Measuring recall is much more difficult than measuring precision because it depends on knowing the number of relevant documents in the entire collection, which necessitates assessment of all documents in the collection.

However an obvious trade-off exists here. If one retrieves all of the documents in a collection, then one is sure of retrieving all the relevant documents in the collection in which case the recall will be "perfect", i.e., one. On the other hand, in the common situation where only a small proportion of the documents in a collection are relevant to the given query, retrieving everything will give a very low precision (close to zero). The usual plausible assumption is that the user wants the best achievable combination of good precision and good recall, i.e., ideally she would like to retrieve all the relevant documents and no non-relevant documents. However, in practice, some users attach greater importance to precision, i.e., they want to see some relevant documents without wading through a lot of junk. Others attach greater importance to recall, i.e., they want to see the highest possible proportion of relevant documents. Hence, Van Rijsbergen [Rij79] offers the *E (for Effectiveness) measure*, a weighted harmonic mean of precision and recall that takes a high value only when both precision and recall are high. The E measure is computed as

$$E \approx 1 - \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

$$where \quad P = precision, R = recall \quad \& \quad 0 < \alpha < 1$$

2.3.2. Models of IR. Broadly, there are two major categories of IR technology and research: semantic and statistical. Semantic approaches attempt to implement some degree of semantic analysis; in other words, they try to reproduce to some degree the understanding of the natural language text that a human user would provide. In statistical approaches, the documents that are retrieved or that are highly ranked are those that match the query most closely in terms of some statistical measure. By far the greatest amount of work to date has been devoted to statistical approaches. In this thesis, I extend popular statistical models and apply them for determining similarity over structured records given by databases. Hence in the following we will briefly look at the various statistical approaches used in IR. Statistical approaches fall into a number of categories: *boolean, vector space, and probabilistic.* Statistical approaches break documents and queries into terms. These terms are the population that is counted and measured statistically. Most commonly, the terms are words that occur in a given query or collection of documents. Many techniques for extracting relevant terms have been developed over time, and they include:

- Stop-word elimination: Make a list of stop-words i.e. words that are too common to be meaningful e.g. prepositions and connectors. Pick only terms that do no appear in the list for representing the document.

- Stemming: Reduce words to their morphological root using a stemmer. The stemming algorithm written by Martin Porter [Por80, JKW97] is popularly used for stemming. This reduces the number of terms that are indexed and also allows for easier term to term comparisons (e.g. “child” and “childish” are considered as the same term).

- Thesauri lookup: Use synonyms of terms to reduce the total number of terms and can be used to give some degree of flexibility to users in posing queries.

Other sophisticated techniques include using noun-phrases instead of single terms in the hope that phrases will contain more semantic meaning.

A user specifies her information need to the system in the form of a query. Given a representation of the user's information need and a document collection, an IR system estimates the likelihood of a document in the collection matching the user's information need. The representation of documents and queries, and the metrics use to compute the similarity among them constitute the *retrieval model* of the system. Existing retrieval models can be broadly classified as:

- *Boolean Model*: Systems based on boolean retrieval partition the set of documents into either being relevant or irrelevant but do not provide degrees of relevance of the document. In this model, each document is represented as a binary-valued vector of length k, where k is the number of terms in the collection. The i^{th} element of the vector is assigned "true" if the document contains the corresponding term. For all terms not present in the document the corresponding element in the vector is set to "false". A query is represented as a Boolean expression in which operands are terms. A document whose set of terms satisfies the Boolean expression is relevant to the user.

- *Vector Space model*: Vector space model considers a document as a collection of words (terms). A term may appear multiple times in a document. Hence the notion of frequency of a term, called *term frequency (tf)* is used to determine the importance of a word in a document. Further, a term may appear in a number of documents of the collection. *Document frequency (df)* of a term measures the number of documents in which the term appears at least once. A high term frequency value shows that the term is important in a given document whereas a high document frequency indicates that the term is not useful to discriminate among documents and may not be useful for retrieval. Hence a more useful metric called *inverse document frequency (idf)* is popularly used to determine the discrimination capability of a term and is computed as $idf = log\frac{size(collection)}{df}$. All terms in a document are assigned weights based on a combination of the term and document frequency of the term. The product of tf and idf has proved to be a good estimation of the weights. Thus the weight for term i in a document is denoted as $w_i = tf_i \times idf_i$. Specifically, if there are N distinct terms in the collection of documents, then each document is viewed as a point in the N dimensional space. A query is also represented as a document with weights for terms appearing in the query. To determine which documents are close to the query a similarity function is defined. A number of measures have been identified [Rij79, BYRN99] to measure the similarity among documents. The most common similarity measure is the *cosine similarity measure* [Rij79, BYRN99] where the cosine of the angle between the document and query vectors in the N dimensional space and the origin is measured. The cosine of the angle between query and document vector is computed as

$$D.Q = |D||Q|cos\theta$$

Rearranging the above formula gives us:

$$Similarity(D,Q) = \frac{D.Q}{|D||Q|} \quad where \quad D.Q = \sum_{i=1}^{i=N} D_i \times Q_i$$

An alternate but equally popular similarity metric is the *Jaccard Coefficient* [Rij79, BYRN99], and is measured as:

$$Similarity(D,Q) = \frac{|D \cap Q|}{|D \cup Q|}$$

Vector models are suited for situations where all terms used to describe the document content are of the same type, i.e. homogeneous.

- *Probabilistic Retrieval Models*: In these models the system estimates the probability of relevance of a document to the user's information need specified as a query. Documents are ranked in decreasing order of probability relevance estimate. Given a document and a query, the system computes $P(R/d,q)$ which represents the probability that the document d will be relevant to the user's query q. These probabilities are computed and used to rank the documents using the Bayes' theorem and a set of independence assumptions about the distribution of terms in the documents. INQUERY [CCH92] is an example of this model.

Comparison of different models of IR is done by assessing their performance on standard benchmarks like the TREC collections [NIS05]. Performance is qualitative unlike in databases (execution speed) and is measured using precision and recall. IR techniques are the driving force behind most the Internet search engines like Altavista [Eng05a], Google [Eng05b], Yahoo [Eng05c], etc. Even though most of these systems have simple features, their success is suggested by the high popularity they enjoy among Internet users.

CHAPTER 3

ANSWERING IMPRECISE QUERIES

In this chapter we present details about our domain independent approach, AIMQ, developed for efficiently extracting and automatically ranking tuples satisfying an imprecise query. Our proposed approach for generating relevant results to an imprecise query requires neither domain specific information nor changes to the architecture of the underlying database. Our solution involves mapping the given imprecise query to a precise query (called *base query*) that is a specialization of the imprecise query. Next AIMQ derives a set of precise queries by considering each answer tuple of the base query as *a relaxable selection query.* The heuristic for guiding the relaxation process is based on the approximate dependencies mined from the underlying database. Chapter 5 has more details about our heuristic relaxation approach.

The tuples obtained after the relaxation are not all equally relevant to their respective queries and hence must be ranked in terms of their similarity to the query. For numerical attributes we can use L_p distance metrics to capture similarity. However no such widely accepted measure exists for categorical attributes. Therefore, as part of AIMQ, we developed a context-sensitive domain independent approach for estimating the semantic similarity between values binding categorical attributes. More details about the similarity estimation approach are in Chapter 4.

A key factor motivating our approach is our desire to minimize the input a user must provide to get an answer to her imprecise query. Specifically, we wish to provide an interface similar to that given by the increasingly popular Web search engines, where the user only provides a set of keywords that vaguely specify her query and the search engine returns a ranked list of documents that satisfy the user query to some degree. On similar lines, *our quest is to determine whether we can provide relevant answers to a imprecise query by using only the information contained in the database.* Given that the database contains tuples generated by humans, they must capture some amount of real-world semantics e.g. relationships between attributes (features of the domain), similarity between values etc. Hence in developing AIMQ, the question we wish to answer the question *How closely can we model users' notion of relevance by mining meta-data from the database?*

Below we begin by giving the architecture of the AIMQ system, explain the data and query model we use and finally describe AIMQ's approach for answering imprecise queries.

3.1. AIMQ Architecture

The AIMQ system as illustrated in Figure 2 consists of four subsystems: Data Collector, Dependency Miner, Similarity Miner and the Query Engine. The data collector probes the autonomous databases to extract sample subsets of the databases.

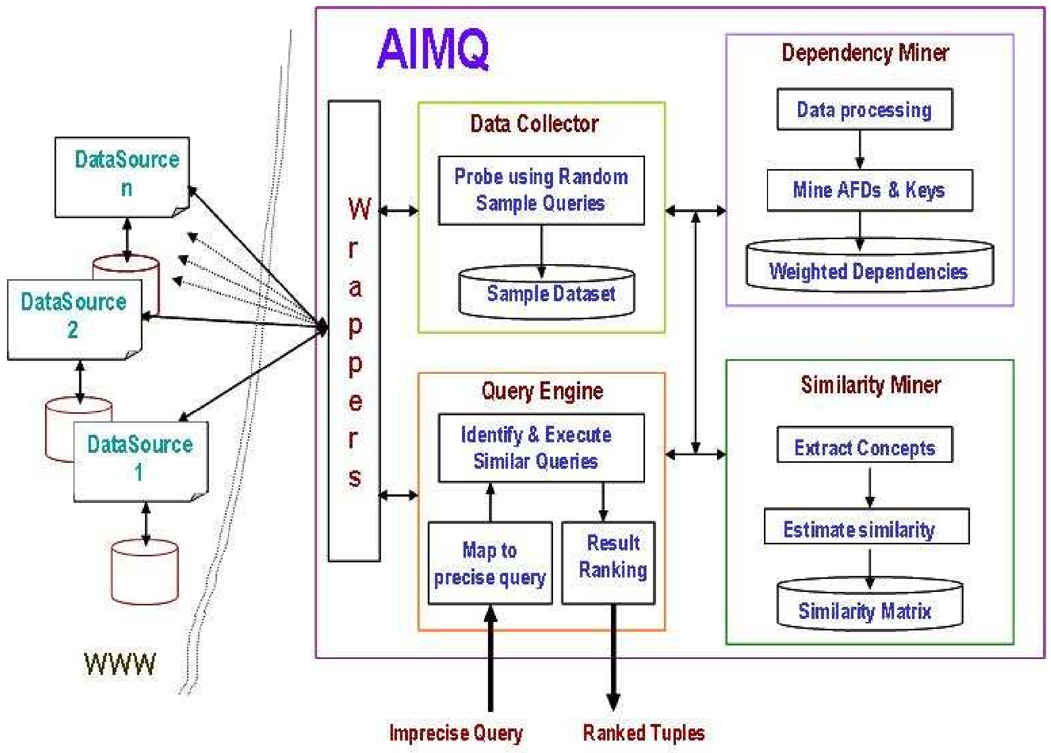
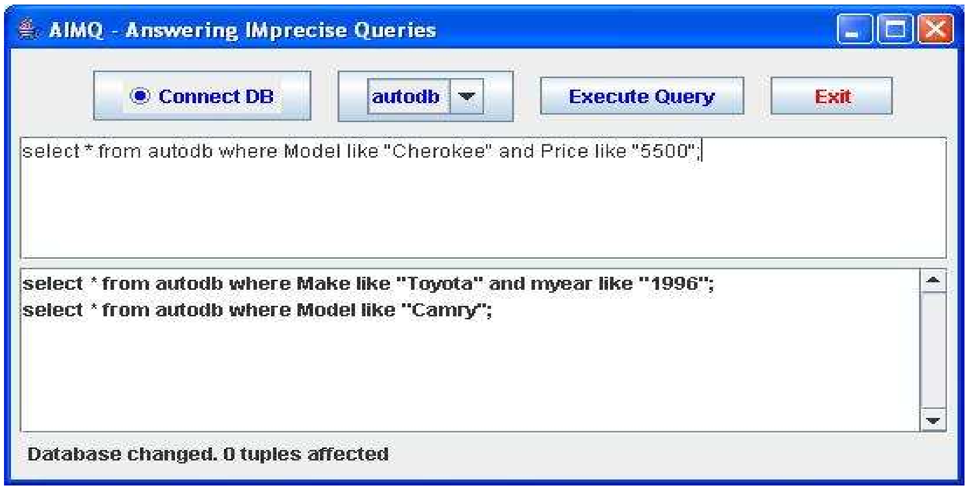


Figure 2. AIMQ system architecture

The extracted sample is processed for efficient extraction of attribute dependencies and value similarities. Dependency Miner mines approximate dependencies 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 as well as to ascribe weights to similarities shown by each attribute. The Similarity Miner uses an association based similarity mining approach to estimate similarities between categorical values. AIMQ also contains wrappers to access the Web databases. However, in this thesis we do not focus on challenges involved in generating and maintaining the wrappers[1]. Figure 3 is the interface for asking imprecise queries provided by AIMQ.



```
select * from autodb where Model like "Cherokee" and Price like "5500";
```

```
select * from autodb where Make like "Toyota" and myear like "1996";
select * from autodb where Model like "Camry";
```

```
Database changed. 0 tuples affected
```

Figure 3. AIMQ Query Interface

3.2. Data and Query Model

Attribute: An attribute is a template for possible values and set of functions, operators that operate on these values and define the behavior. All operators and functions except the similarity function are assumed to be built-in functions for any attribute. The domain of an attribute is the set of all values following this template.

Attribute-value: An attribute-value (also called an instance of the attribute) is one of all possible values in the domain of the attribute.

Tuple: Given a set of n pairs (A_i, D_i), $0 \leq i \leq n$, where A_i is an attribute and D_i is its domain. A *tuple* t is an element of the cartesian product of $D_1, D_2, ..., D_n$.

[1]Recent research done as part of the MetaQuerier[ZZC04] system has shown promising results in automatic interface extraction (wrapper generation).

An attribute supports a number of related operators that are applicable to instances of the attribute. If the domain of the attribute has a total order (e.g. is numeric in nature) then the set of operators is $\{<, \leq, =, \geq, >\}$. The built-in predicates in boolean model are based on crisp (boolean) retrieval semantics. These predicates are used to represent the conditions in traditional database queries. A similarity version of each predicate is represented by suffixing $like$ to the predicate. In this thesis we focus on the similarity predicate *like*, as we are also interested in determining similarity over attributes who domains are not ordered (e.g. categorical attributes). However, for the ordered domain, *like* can be trivially extended to obtain $like(=), like(<), like(>), like(\leq)$ and $like(\geq)$.

Query conditions can be of two types, crisp conditions which are exact matches and similarity expressions which serve to rank the results. The crisp conditions follow the traditional boolean model of true and false.

Precise Query: A user query that requires data exactly matching the query constraint is a precise query. A precise query contains only crisp conditions over the attributes. For example, the query

```
Q:- CarDB(Make = Ford)
```

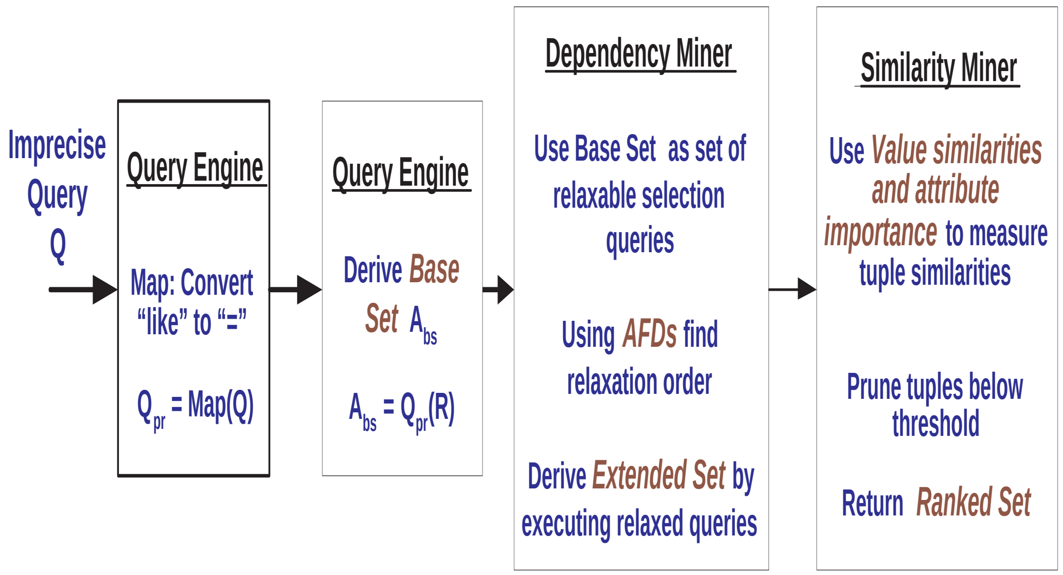
is a precise query, all of whose answer tuples must have attribute *Make* bound by the value *Ford*.

Imprecise Query: A user query that does not insist on exact match (and only requires data *closely* matching the query constraint) is an imprecise query. Thus an imprecise query contains similarity expressions that rank the results. Answers to such a query must be ranked according to their closeness/similarity to the query constraints. For example, the query

`Q:- CarDB(Make like Ford)`

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

3.3. The AIMQ Approach



Imprecise Query Q
Query Engine
Map: Convert "like" to "="
Q_pr = Map(Q)
Query Engine
Derive Base Set A_bs
A_bs = Q_pr(R)
Dependency Miner
Use Base Set as set of relaxable selection queries
Using AFDs find relaxation order
Derive Extended Set by executing relaxed queries
Similarity Miner
Use Value similarities and attribute importance to measure tuple similarities
Prune tuples below threshold
Return Ranked Set

Figure 4. FlowGraph of the AIMQ approach

Below we give an overview of AIMQ's imprecise query answering approach. Continuing with the scenario in Example 1.2, let the user's intended query be:

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

We begin by assuming that the tuples satisfying some specialization of Q – called the *base query* Q_{pr}, are *indicative* of the answers of interest to the user. For example, it is logical to assume that a user looking for cars like Camry would be happy if shown a Camry that satisfies most of her constraints. Hence, we derive Q_{pr}[2] by tightening the constraints from *"likeliness"* to *"equality"*:

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

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

```
t1 = [Make=Toyota, Model=Camry, Price=10000, Year=2000]
t2= [Make=Toyota, Model=Camry, Price=10000, Year=2001]
```

The tuples $t1$ and $t2$ completely satisfy the constraints of the base query Q_{pr}. But the user is also interested in tuples that have binding values 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 of Q_{pr}. Thus,

```
t3= [Make=Honda, Model =Accord, Price=9800, Year=2000]
```

could be seen as being similar to the tuple $t1$ and therefore a possible answer to Q. We could also show other *Camrys* whose *Price* and *Year* values are slightly different to those of tuples in $t1$ and $t2$. Specifically, all tuples of *CarDB* that have one or

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

more binding values close to some answer tuple of Q_{pr} can be considered as potential answers to query Q. Thus by considering each tuple in the base set as a relaxable selection query we can extract additional tuples from the database that are similar to tuples in the base set. These new tuples form the *extended set* of answers that are relevant to the imprecise 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 affects the binding values of other attributes. *Approximate functional dependencies* (AFDs) 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. After the relaxation process, a large number of tuples may be found as being possibly relevant to the tuples of the base query. But not all tuples will be equally relevant. Therefore we use Equation 3.1, described below, to measure the semantic similarity of each tuple in extended set to the corresponding tuple in base set and provide a ranked list of answers that closely match the given imprecise query.

3.3.1. Extracting Relevant Answers. A formal description of the AIMQ approach for answering an imprecise selection query over a database is given in Algorithm 1. Given an imprecise query Q to be executed over relation R, the threshold of similarity T_{sim} and the attribute relaxation order $\widehat{A}_{relax}$, we begin by mapping the

Algorithm 1 Extracting Relevant Tuples

```
Require: Q, R, $\widehat{A}_{relax}$, $T_{sim}$
1: Let $Q_{pr} = \{Map(Q)|A_{bs} = Q_{pr}(R), |A_{bs}| > 0\}$
2: $\forall\ t \in A_{bs}$
3:    $Q_{rel}$ = CreateQueries(t, $\widehat{A}_{relax}$)
4:    $\forall\ q \in Q_{rel}$
5:       $A_{rel} = q(R)$
6:       $\forall\ t' \in A_{rel}$
7:          if $Similarity(t,t') > T_{sim}$
8:             $A_{es} = A_{es} \bigcup t'$
9: Return Top-k($A_{es}$).
```

imprecise query Q to a precise query Q_{pr} having a non-null answerset (Step 1). The set of answers for the mapped precise query forms the *base set* A_{bs}. By extracting tuples having similarity above a predefined threshold, T_{sim}, to the tuples in A_{bs} we can get a larger subset of potential answers called *extended set* (A_{es}). Every tuple $t \in A_{bs}$ can be seen as a precise selection query with values binding all the attributes. Therefore by relaxing the constraints of tuple t we can generate new queries whose answers will be similar to t and consequently relevant to the imprecise query Q. Randomly relaxing constraints can lead to queries having no answers or that have many irrelevant tuples. To ensure more relevant tuples are retrieved after relaxation, we use the Algorithm 2 to determine an attribute relaxation order $\widehat{A}_{relax}$. Using $\widehat{A}_{relax}$, we generate a set of precise queries Q_{rel} from each tuple in A_{bs} (Step 3). Executing a query $q \in Q_{rel}$ over R will give us a set of tuples, A_{rel}, that are relevant to the corresponding tuple $t \in A_{bs}$ (Step 5). 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 tuple t. Therefore we measure the similarity of each tuple $t' \in A_{rel}$ to the tuple $t \in A_{bs}$ (Step 7). Only if t' shows similarity above the threshold T_{sim} do we add it to the set of relevant answers A_{es} for Q (Step 8). The algorithm returns only the top-k[3] tuples (in terms of similarity to Q) to the user.

3.3.2. Estimating Query-Tuple Similarity. AIMQ measures the similarity between an imprecise query Q and an answer tuple t as a *weighted summation* of the similarity scores of corresponding attributes in Q and t. Thus the similarity is estimated as

$$Sim(Q,t) = \sum_{i=1}^{n} W_{imp}(A_i) \times \begin{cases} VSim(Q.A_i, t.A_i) \\ \text{if Domain}(A_i) = \text{Categorical} \\ \\ 1 - \frac{absolute(Q.A_i - t.A_i)}{Q.A_i} \\ \text{if Domain}(A_i) = \text{Numerical} \end{cases} \tag{3.1}$$

where $n = Count(boundattributes(Q))$, W_{imp} ($\sum_{i=1}^{n} W_{imp} = 1$) is a factor corresponding to the importance of an attribute and the function *VSim* measures the similarity between the categorical values. If the distances computed using $\frac{absolute(Q.A_i - t.A_i)}{Q.A_i}$ is greater than 1, we assume the distance to be 1 to maintain a lowerbound of 0 for numeric similarity. AIMQ assumes the attributes to have either discrete numerical or categorical values. As mentioned earlier in Chapter 1, our motivation is to provide answers that are acceptable to the users with minimal input from the users themselves and hence we do no wish to burden the users by having to provide the necessary

[3]Algorithm 3 assumes that similarity threshold T_{sim} and the number of tuples (k) to be returned to the user are tuned by the system designers.

distance metrics. Moreover, lay users will often find it difficult to come up with distance metrics that capture their notions about similarity. Hence we must compute the similarity between values binding the query and the tuples using automated solutions.

There are many ways in which we can determine the similarity between two objects. For example, for numeric data, the data values can be viewed as direct arguments to a calculation function. For non-numeric data, some form of similarity procedure can be developed that correlates non-numeric instances with numeric values (e.g. represent non-numeric data using Unicode or ASCII representation). Such methods commonly produce a numeric value indicating the closeness of the values according to some accepted convention and scale. However, in this respect, it can be argued that the reduction of non-numeric data to numeric proximity values can, for some applications, be improved, particularly when the value itself has no meaning except as a comparison. Even for numeric data, in many cases the useful distance between two values may not simply be the numeric difference between them. A useful measure of similarity for some spatial applications, for example, may be a measure of the time taken to get from *point A* to *point B* rather than any of the numerous methods of measuring physical proximity. The similarity between *two objects may also be a function of their behavior with respect to other attributes in the relation.* Often, users perception of similarity called *semantic similarity* is not based on the (syntax of the) values themselves but on their behaviour as described by their interactions with other values. For example, finding cars similar to *Toyota Camry* will require comparing every feature of *Camry* to those of other cars. Note that semantic similarity always subsumes the syntactical similarity. Since our motivation is to provide answers acceptable to users, we must measure the semantic similarity between the query and the answer tuples.

The user perception of similarity between numeric values is based mostly on the (physical or syntactic) distances between the values themselves. A L_p distance metrics such as Manhattan distance and Euclidean distance are widely accepted measures for computing distances between numeric values irrespective of the domain in which these values appear. Hence in this dissertation we assume that L_p distance captures the semantic (user perceived) similarity between numeric values. But no such domain and user-independent measure is available for measuring the similarity between values binding a categorical attribute.

In the context of AIMQ, the crucial first step is in deciding the set of precise queries using which relevant tuples can be extracted from the database. While the task is important, the challenge there is not in extracting the tuples but doing so efficiently. Irrespective of the efficiency of the tuple extraction phase, we do require a distance metric for measuring the semantic similarity between the tuples and the given imprecise query. Therefore, deciding the similarity between categorical values becomes very vital to the task of answering an imprecise query.

Hence in the following, we first present a domain independent solution for estimating semantic similarity between categorical values in Chapter 4. Then in Chapter 5 we present a heuristic for deciding which attributes to relax first such that the tuples that are likely to be more relevant are extracted earlier from the database.

CHAPTER 4

ESTIMATING SEMANTIC SIMILARITY AMONG CATEGORICAL VALUES

The need to determine *semantic similarity* between two lexically expressed concepts is a problem that pervades much of natural language processing. Measures of relatedness are used in such applications as word sense disambiguation, determining the structure of texts, text summarization and annotation, information extraction and retrieval, automatic indexing, lexical selection, and the automatic correction of word errors in text. We must note that semantic relatedness is a more general concept than *semantic similarity.* In natural language processing systems *synonymy* was considered as the basic semantic relation. *Similar words were considered semantically related by virtue of their synonymy* (e.g. bank – trust company), but dissimilar entities may also be semantically related by lexical relationships such as *meronymy*[1] (e.g. car - wheel) and *antonymy*[2] (e.g. hot - cold), or just by any kind of functional relationship or frequent association (e.g. pencil - paper, penguin - Antarctica, rain - flood). Computational applications typically require relatedness rather than just

[1]A meronym denotes a constituent part of, or a member of something.
[2]Antonymy holds between two words that can (in a given context) express opposite meanings.

similarity. However, in this thesis, we measure only the semantic similarity between words (attribute values) such that a ranked set of answers can be presented to a given imprecise query. Therefore, our focus is only on learning the synonymy between words.

According to (a statement usually attributed to) Gottfried Liebniz, "*two words are synonyms if one can be used in place of the other without changing the meaning of the statement*". However, linguists consider that for all synonyms there is at least one statement whose meaning is changed by the substitution of one word for another. Hence, a weaker definition of synonym by Liebniz - "*synonyms are words that are interchangeable in some contexts*" is considered more realistic. This definition makes synonymy (and thus similarity) relative to context.

The semantic similarity estimation approach we developed is as under. Given a database of tuples, we assume that binding values that are semantically similar have similar distributional behaviour. With this assumption, we can treat the values that co-occur near a value as constituting features that describe the context in which the given value appears in the database. The semantic similarity between two values is then computed in terms of how similar is their contexts i.e. by measuring the commonality of their features. An alternate interpretation is to consider the sets of features co-occurring (associated) with a given value as representing the context in which the value appears. Then measuring the similarity between the features sets of two values will tell us the degree of similarity of the context in which the value appears in the database or corpus. The closer this similarity, the more semantically related would be the values. Simply put, *two values are (semantically) similar if* *they share a number of common features.* This model forms the basis of our domain independent approach for measuring the semantic similarity between values binding categorical attributes.

Below we present a association based context sensitive semantic similarity estimation technique. In keeping with the discussion above, we determine the *similarity between two values as their behavioral similarity or the commonality in their contexts.*

4.1. AV-Pair

We call the combination of an attribute and a distinct value binding it as an *AV-pair.* E.g. *Make=Ford* is an AV-pair.

We consider two values as being associated if they occur in the same tuple. Two AV-pairs are associated if their values are associated. The similarity between two AV-pairs can be measured as the percentage of associated AV-pairs common to them. More specifically, given a categorical value, all the AV-pairs associated to the value can be seen as the features describing the value. The set (bag in our case) of features together can be considered as capturing a specific context in which the given AV-pair appears. Consequently, the similarity between two values can be estimated by the commonality in the features (AV-pairs) describing them and hence the closeness of the context in which they appear. For example, given tuple t ={*Ford, Focus, 15k, 2002*}, the AV-pair *Make=Ford* is associated to the AV-pairs *Model=Focus, Price=15k* and *Year=2002.* The AV-pairs formed by combining distinct values binding the attributes *Model, Price* and *Year* can be seen as the features describing the AV-pairs over *Make.*

Similarly AV-pairs of *Make, Price* and *Year* for those of *Model* and so on.

4.2. Supertuple

Databases on the web are autonomous and cannot be assumed to provide meta-data such as possible distinct values binding an attribute. Hence we must extract this information by probing the database using sample queries. From the data extracted by probing we can identify a subset of all AV-pairs in the database [3] over the relation.

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 Make=Ford

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. Table 1 shows the supertuple for *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*

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

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 AV-pair query.

4.3. Measuring Categorical Value Similarity

We measure the similarity between two AV-pairs as the similarity shown by their supertuples. We use the *Jaccard Similarity metric* [BYRN99] to estimate similarity between the supertuples. The supertuples contain bags of keywords for each attribute in the relation. Hence we use Jaccard Similarity [HGKI02] with bag semantics to determine the similarity between two supertuples. The Jaccard Coefficient (Sim_J) is calculated as

$$Sim_J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

Unlike pure text documents, supertuples would rarely share keywords across attributes. Moreover all attributes (features) may not be equally important for deciding the similarity between two categorical values. For example, given two cars, their *prices* may have more importance than their *color* in deciding the similarity between them. Hence, given the answersets for an AV-pair, we generate bags for each attribute in the corresponding supertuple. The value similarity 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. Thus, similarity between two values is calculated as

$$VSim(V_1, V_2) = \sum_{i=1}^{m} W_{imp}(A_i) \times Sim_J(St_1.A_i, St_2.A_i) \tag{4.2}$$

where V_1 and V_2 are values binding a categorical attribute and St_1, St_2 are the corresponding supertuples with m attributes, A_i is the bag corresponding to the i^{th} attribute in the supertuple, $W_{imp}(A_i)$ is the importance weight of A_i. However, determining a good weighing scheme to accurately reflect the concerns of the user is a difficult task. While users may have an idea of what features (attributes) are more or less important given a AV-pair, it is not certain that they can accurately model their preferences in terms of importance weights. This further motivates the need to learn the importance of the attributes automatically. In the next chapter, Chapter 5, we present a domain independent approach for estimating the importance to be ascribed to an attribute.

4.4. Summary and Discussion

In this chapter we presented the context sensitive similarity estimation technique we use to compute the similarity between categorical values. As already defined, we measure the similarity between two categorical values as the similarity of their contexts or supertuples. However, by computing the similarity between the supertuples as the Jaccard Similarity between attribute bags, we are only measuring whether the same exact AV-pair is co-occurring with both the values. Thus, we are approximating the similarity between values by the *degree of equality* between their contexts. Measuring only equality among categorical AV-pairs is acceptable since we do not have prior access to a distance metric for measuring similarity among categorical values. However, for numeric values this assumption contradicts with our earlier claim of L_p

distance metrics like Euclidean or Manhattan being able to measure similarity. Below we highlight the challenges involved in computing the similarity between numerical AV-pairs in the context of measuring bag similarity.

We can measure the similarity between numeric elements (AV-Pairs) of bags as described in Equation 3.1, i.e.

$$Sim(P_1.i, P_2.j) = min(Wt_{P_1.i}, Wt_{P_2.j}) \times 1 - \frac{absolute(P_1.i - P_2.j)}{P_1.i} \tag{4.3}$$

Moreover, as in Equation 3.1, if the distance computed using $\frac{absolute(P_1.i - P_2.j)}{P_1.i}$ is greater than 1, we can assume the distance to be 1 to maintain a lowerbound of 0 for numeric similarity. However, using Equation 3.1 we cannot capture similarity between all pairs of numeric values. Specifically, given some $P_1.i$, Equation 3.1 will only measure the true similarity of $P_2.j$ with values in the range $[0, 2 \times P_1.i]$. The similarity of all $P_2.j$'s with values greater than $2 \times P_2.i$ will be considered as equal to that of $2 \times P_2.i$ irrespective of how large the actual value is from $2 \times P_2.i$. Thus, while using Equation 4.3 is better than assuming equality, it does not always capture true similarity between any two numeric values. While the similarity measured by Equation 4.3 for values not lying in the range $[0, 2 \times P_1.i]$ is an approximation, this approximation is acceptable when testing whether a given tuple is having values close to the user given value in the query. Specifically, by using the Equation 3.1 we are putting a bound,$[0, 2 \times Q.value]$ on the neighbourhood of interest within which we will correctly rank the tuples.

A more critical issue that arises when using Equation 4.3 is that of deciding the possible values for $P_2.j$ for a given $P_1.i$. If we assign all values in P_2 to $P_2.i$ we will get a similarity vector with size equal to P_2 for every $P_1.i$. While a single number can be obtained by summing up each such vector, it is not clear what such number will represent and at times can lead to false conclusions. We will illustrate the problem using an example. Suppose we are interested in measuring the similarity between three bags of car *Prices*, P_1, P_2 and P_3. Specifically, we want to decide whether P_1 or P_3 is more similar to P_2. Let $P_1 = \{10 :, 20 : 1, 100 : 1\}$, $P_2 = \{10 : 1, 25 : 2, 95 : 5\}$ and $P_3 = \{1 : 5, 90 : 4\}$. Here the representation semantics is same as that of Figure 1. If we use Equation 4.2 to measure similarity we will get $Sim(P_2, P_1) = 0.09$ as only one element 10 is common while $Sim(P_2, P_3) = 0$ as no elements are common between them. Thus, P_1 will be considered as more similar to P_2 than P_3 - a result that is intuitive given the closeness in distribution of elements in P_1 and P_2. Now let us use Equation 4.3 to compute the similarity of each element of P_2 to all elements in P_1 and P_3 and sum up the values to obtain a single number of similarity. We will get $Sim(P_2, P_1) = \{1 + 0.75 + 0.95\} = 2.7$ and $Sim(P_2, P_3) = \{4 \times 0.95\} = 3.8$. Thus, by using Equation 4.3 we would conclude that both P_3 is more similar to P_2. However, such a conclusion does not seem intuitive given that P_1 has more distinct elements similar with P_2 than P_3. Thus, measuring similarity between bags as the similarity of their elements instead of equality of elements does not guarantee measurement of a more intuitive and user-acceptable similarity measure.

The above problem may have been introduced by computing similarity between all pairs of values in the bags. Instead, we can restrict computing the similarity of $P_1.i$ to only the closest value possible as $P_2.j$. However, finding the closest value entails additional computational cost ($O(n^2)$ additional comparisons for each pair of bags) and cannot provide any more guarantees about the computed similarity measure. In comparison, computing bag similarity by only considering equality among AV-pairs has clean semantics - that of measuring commonality among the features. Therefore, when computing the categorical value similarities we do not differentiate between numerical and categorical valued features and only look for equality of AV-pairs.

CHAPTER 5

LEARNING ATTRIBUTE IMPORTANCE MEASURES FOR EFFICIENT QUERY RELAXATION

The approach used by AIMQ for answering imprecise queries requires generation of new selection queries by relaxing the constraints of the tuples in the base set A_{bs}. The underlying motivation there is to identify tuples that are closest to some tuple $t \in A_{bs}$. 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 the least important attribute as the attribute whose binding value, when changed, has minimal effect on values binding other attributes. *Approximate Functional Dependencies (AFDs)*[HKPT98] efficiently capture such relations between attributes. The underlying assumption in using AFDs is the belief that the database instances reflect the real world and hence the significance models held by users. In the following we will explain how we use AFDs to identify the importance of an attribute and thereby guide the query relaxation process.

5.1. Mining AFDs for Query Relaxation

The attribute relaxation heuristic we developed for supporting efficient query relaxation is described below. We begin by giving necessary definitions and then explain our approach for estimating attribute importance.

5.1.1. 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 $t, u \in r$ and $\forall B \in X$ we have $t.B = u.B \Rightarrow t.A = u.A$.

Approximate Functional Dependency (AFD): The functional dependency $X \rightarrow A$ over relation r 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 $error(X \rightarrow A) \leq T_{err}$, where the error threshold $T_{err} \in (0, 1)$ and the error is measured as a ratio of the tuples that violate the dependency to the total number of tuples in r.

Approximate Key (AKey): An attribute set $X \subset R$ is a key over relation r if no two distinct tuples in r agree on X. However, if the uniqueness of X does not hold over a small fraction of tuples in r, then X is considered an *approximate key*. Specifically, X is an approximate key if $error(X) \leq T_{err}$, where $T_{err} \in (0, 1)$ and $error(X)$ is measured as the minimum fraction of tuples that need to be removed from relation r for X to be a key.

Several authors [Lee87, KM95, DR00] have proposed various measures to approximate the functional dependencies and keys that hold in a database. Among them, the g_3 measure proposed by Kivinen and Mannila [KM95], is widely accepted. The g_3 measure is defined as the ratio of minimum number of tuples that need be removed from relation R to make $X \rightarrow Y$ a functional dependency to the total number of tuples in R. This definition is consistent with our definition of approximate dependencies and keys given above. Hence we use *TANE* [HKPT98], the algorithm developed by Huhtala et al for efficiently discovering AFDs and approximate keys whose g_3 approximation measure is below a given error threshold. A brief overview of the TANE algorithm is given in Appendix A.

We mine the AFDs and keys using a subset of the database extracted by probing. Some of the AFDs and approximate keys mined from a probed sample of a user car database (used for evaluating our approach) are shown in Table 2. Specifically, the tuple showing *Make, Price* → *Model* with support 0.65 implies that if you know the *Make* and *Price* of a car in the database then with 0.65 probability you can guess the *Model* of the car. Similarly, the 0.78 support for approximate key *Model, Mileage* gives the probability of uniquely identifying a tuple given that approximate key.

AFD	**Support**
Model → Make	0.96
Make, Price → Model	0.65
Approximate Key	**Support**
Model, Mileage	0.78
Make, Price	0.54

Table 2. Sample AFDs and Approximate Keys mined from CarDB

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

```
Algorithm 2 Attribute Relaxation Order
Require: Relation R, Dataset r, Error threshold $T_{err}$
 1: $S_{AFD}$={$x|x \in$ GetAFDs(R,r), $g_3(x) < T_{err}$}
 2: $S_{AK}$={$x|x \in$ GetAKeys(R,r), $g_3(x) < T_{err}$}
 3: $AK$={$k|k \in S_{AK}$, $\forall k' \in S_{AK}$ support(k) $\geq$ support(k')}
 4: $\overline{AK} = \{k|k \in R - AK\}$
 5: $\forall\ k \in AK$
 6:     $Wt_{decides}(k)$=$\sum \frac{support(\hat{A} \rightarrow k')}{size(\hat{A})}$
                where $k \in \widehat{A} \subset R$, $k' \in R - \widehat{A}$
 7:     $Wt_{AK} = Wt_{AK} \bigcup [k, Wt_{decides}(k)]$
 8: $\forall\ j \in \overline{AK}$
 9:     $Wt_{depends}(j)$ = $\sum \frac{support(\hat{A} \rightarrow j)}{size(\hat{A})}$ where $\hat{A} \subset R$
10:     $Wt_{\overline{AK}} = Wt_{\overline{AK}} \bigcup [j, Wt_{depends}(j)]$
11: Return [Sort($Wt_{\overline{AK}}$), Sort($Wt_{AK}$)].
```

We therefore propose an alternate approach to break the cycle. We partition the attribute set into *dependent* and *deciding* sets, with the criteria being each member of a given group either depends or decides at least one member of the other group. A topological sort of members in each subset can be done by estimating how dependent/deciding they are with respect to other attributes. Then by relaxing all members in the dependent group ahead of those in the deciding group we can ensure that the least important attribute is relaxed first. We use the approximate key with highest support to partition the attribute set. All attributes forming the approximate key become members of the *deciding set* while the remaining attributes form the *dependent set*. Details of our attribute ordering approach is described in Algorithm 2.

Given a database relation R and error threshold T_{err}, Algorithm 2 begins by extracting all possible AFDs and approximate keys (AKeys). As mentioned earlier, we use the TANE algorithm to extract AFDs and AKeys whose g_3 measures are below T_{err} (Step 1,2). Next we identify the approximate key with the highest support (or least error), AK, to partition the attribute set into the deciding group (attributes belonging to AK) and those that are dependent on AK (belong to $\overline{AK}$)(Step 3,4). Then for each attribute k in deciding group we sum all support values for each AFD where k belongs to the antecedent of the AFD (Step 5-7). Similarly we measure the dependence weight for each attribute j belonging to the dependent group by summing up the support of each AFD where j is in the consequent (Step 8-10). The two sets are then sorted in ascending order and a totally ordered set of attributes in terms of their importance (i.e. how deciding an attribute is) is returned (Step 11). Given the attribute order, we compute the weight to be assigned to each attribute

$k \in Wt_{AK} \bigcup Wt_{\overline{AK}}$ as

$$Wt(k) = \frac{RelaxOrder(k)}{|Attributes(R)|} \times \begin{cases} \frac{WT_{decides}(k)}{\sum Wt_{decides}(j)} \\ \text{if } k \in Wt_{AK} \\ \\ \frac{WT_{depends}(k)}{\sum Wt_{depends}(j)} \\ \text{if } k \in Wt_{\overline{AK}} \end{cases} \tag{5.1}$$

where RelaxOrder returns the position at which k will be relaxed. The position ranges from 1 for least important attribute to *count(Attributes(R))* for the most important attribute. The dependance and decidability measures, $WT_{depends}$ and $WT_{decides}$ are computed as

$$\begin{aligned} &Wt_{depends}(j) = \sum \frac{support(\hat{A} \rightarrow j)}{size(\hat{A})} \\ &Wt_{decides}(k) = \sum \frac{support(\hat{A} \rightarrow j)}{size(\hat{A})} \\ &\text{where } \hat{A} \subset R, k \in \hat{A}, j \in R - \hat{A} \end{aligned} \tag{5.2}$$

The relaxation order we produce using Algorithm 2 only provides the order for relaxing a single attribute of the query at a time. Given the single attribute ordering, we greedily generate multi-attribute relaxation assuming the multi-attribute ordering strictly follows the single attribute ordering. For example, suppose the 1-attribute relaxation order is

$$a_1 \rightarrow a_3 \rightarrow a_4 \rightarrow a_2$$

then the 2-attribute order will be

$$a_1, a_3 \rightarrow a_1, a_4 \rightarrow a_1, a_2 \rightarrow a_3, a_4 \rightarrow a_3, a_2 \rightarrow a_4, a_2$$

The 3-attribute order will be a cartesian product of 1 and 2-attribute orders and so on. Figure 5 shows the 1 and 2-attribute relaxation order we derived for the prototype database CarDB we designed to evaluate AIMQ(see Chapter 6 for details).

```
Make → Price → Year → Model

Make, Price → Make, Year → Make, Model → Price, Year →
Price, Model → Year, Model
```

Figure 5. Attribute Relaxation Order in CarDB

5.2. Sampling the Databases

In order to learn the attribute importance and value similarities , we need to first collect a representative sample of the data stored in the sources. Since the sources are autonomous, this will involve "probing" the sources with a representative set of "probing queries". Below we describe the process of selecting probe queries and also highlight possible affects of sampling on our solution.

5.2.1. Generating Probe Queries. There are two possible ways of generating "representative" probing queries. We could either (1) pick our sample of queries from a set of *spanning queries* - i.e., queries which together cover all the tuples stored in the data sources or (2) pick the sample from the set of actual queries that are directed at the system over a period of time. Although the second approach is more sensitive to the actual queries that are encountered, it has a chicken-and-egg problem as no statistics can be learned until the system has processed a sufficient number of user queries. In this thesis, we assume that the probing queries are selected from a set of spanning queries (the second approach can be used for refining statistics once sufficient queries are issued over the system). In Chapter 8, we present an extension of AIMQ that suggests similar queries by assuming availability of a query log. Spanning queries can be generated by considering a cartesian product of the values binding all attributes, and generating selection queries that bind attributes using the corresponding values of the members of the cartesian product. Assuming the availability of binding values for all attributes projected by an autonomous Web database is not practical. A more feasible solution is to assume the availability of few binding values for some attributes of the projected relation. Given a small seed set of values, we can generate a subset of the spanning queries by considering a cartesian product of the values. New values can be identified from the results of these spanning queries. The process stops when no new binding values are identified or if we have extracted obtained the sample size of our choice.

In this thesis we assume that the system designer/ domain expert is able to provide a set of seed binding values for some attributes found in the relation. For example, in the used car database, it is fairly easy to come up with seed values for almost all attributes. Although a query binding single attribute will generate larger resultsets, most often such queries will not satisfy the binding restrictions of Web sources as they are too general and may extract a large part of the sources data. The less general the query (more attributes bound), more likely it will be accepted by autonomous Web sources. But reducing the generality of the query does entail an increase in the number of spanning queries leading to larger probing costs if sampling is not done. Once we decide the issue of the space from which the probing queries are selected (in our case, a set of spanning queries), the next question is how to pick a representative sample of these queries. Clearly, sending all potential queries to the sources is too costly. We use sampling techniques for keeping the number of probing queries under control. Two well-known sampling techniques are applicable to our scenario: (a) Simple Random Sampling and (b) Stratified Random Sampling [Coc77]. Simple random sampling gives equal probability of being selected to each query in the collection of sample queries. Stratified random sampling requires that the sample population be divisible into several subgroups. However, grouping categorical attributes would require access to domain specific information (e.g. ontologies) - information, as we showed earlier, that is often not available. Hence, we only use the simple random sampling technique to obtain probe queries.

5.2.2. Issues Raised by Sampling. We note at the outset, that the details of the dependency mining and value similarity estimation tasks do not depend on how the probing queries are selected. However, we are approximating the model of attribute dependencies and value similarities found in the database by using a small sample of the database. Therefore, we may end up learning dependencies and value similarities that do not reflect the actual distribution of the database. Intuitively, the larger the sample obtained, the better our approximation of the database. However, as our experimental evaluations will show, assuming a uniform distribution of values in the database our approach will be able to model[1] the relative ordering among attributes and values from a sample of the database. The loss of accuracy due to sampling is not a critical issue for us as it is the *relative* rather than the *absolute* values of the dependencies and value similarities that are more important in query relaxation and result ranking.

5.3. Summary and Discussion

In this chapter, we presented an AFD based query relaxation approach developed as part of AIMQ. Effective query relaxation is very essential for AIMQ to efficiently identify potential answers to an imprecise query. Several approaches for efficient query relaxation/refinement have been investigated under the aegis of cooperative query answering [Gas97, CCL91, CCL92, Mot86]. The solutions developed assumed presence of user provided meta information about relationships between attributes that could be exploited to extend the scope of a given query. Some solutions assume availability of concept hierarchies that could be used to suggest alternate binding values for the query. A detailed discussion about these solutions is in Section 9.2. Given our intent of minimizing user input, we could not adapt any of the suggested user dependent solutions. Moreover, as pointed out earlier in this dissertation, concept hierarchies and ontologies are not readily available and hence could not be used as a basis for driving query relaxation under AIMQ.

[1]In [AC99], authors use uniformity of data assumption to build and maintain histograms using answers of queries issued over the database. They show that such histograms are able to learn the underlying data distributions with little loss of accuracy even for distributions with moderate skew.

The solution therefore was to try and learn a query relaxation heuristic by mining the database. The underlying motivation was to automatically learn an approximation of the users' notion of relevance of an attribute. Once an approximate model is identified, we could refine it by accepting implicit feedback from users. At the outset, the idea of mining patterns from data that reflect real world models might seem to be wishful thinking. On the contrary, *functional dependencies* as a form of real-world relevance among attributes is widely accepted and used in databases. In [RN03], Russell and Norvig point out that functional dependencies express a strict form of relevance that can be learned from observations of the real world - as in tuples in a real-world database. They argue that *dependencies provide sufficient information to allow construction of hypotheses concerning the target attribute.* Therefore, in AIMQ we use functional dependencies between attributes to determine a relevance (importance) based relaxation heuristic. However, full functional dependencies, dependencies that are true for every tuple in the database, do not cover all attributes of the relation. Hence, we use approximate functional dependencies - dependencies that are true over a majority of the tuples, to identify the query relaxation heuristic. A recent work in query optimization [IMH$^+$04] also learns approximate functional dependencies from the data but uses it to identify attribute sets for which to remember statistics. In contrast, we use it for capturing semantic patterns from the data.

In AIMQ, we used AFDs to identify the attribute that is least likely to *cause* other attributes in a relation. However, AFDs are not the only tool that can represent causal relationships. Causal Bayesian Networks [Coo97, CH] and Causal Association

Rules [SsBMU98] are alternative techniques that are useful in learning causal relationships among attributes. Indeed, before deciding to use AFDs, we looked at the feasibility of using the above mentioned techniques in AIMQ.

A *Bayesian network* is a valuable tool for reasoning about probabilistic (casual) relationships. A Bayesian network for a set of attributes $X = X1, , Xn$ is a directed acyclic graph with a network structure S that encodes a set of conditional independence assertions about attributes in X, and a set P of local probability distributions associated with each attribute. A *causal Bayesian network* is a Bayesian network in which the predecessors of a node are interpreted as directly causing the variable associated with that node. However, the possible causal networks are exponential in the number of variables and so practical algorithms must use heuristics to limit the space of networks. The process of identifying a good causal model can be helped by providing prior distribution . In general it is considered that good heuristics combined with prior information could lead to practical causal Bayesian systems. But developing good heuristics and providing prior information are tasks that can be performed by a domain expert. Since our motivation was to avoid the use of domain specific information in developing AIMQ, using causal Bayesian networks for determining attribute causality was not a feasible solution.

In [SsBMU98], authors look at the applicability of constraint-based causal discovery in identifying causal relationships in market basket data. They build on ideas presented in [Coo97] to determine a subset of causal relationships. They argue that *causal Bayesian networks* is impossible to to infer in large scale data mining applications but the constraint-based techniques are feasible. Specifically, they use *information about dependence and independence among set of variables to constrain the number of causal relationships* among a subset of the variables. Simply put, if it is know that attributes A and B are independent, then one can easily infer that no causal relationships exist between them. In general, the constraint-based causal network learners also attempt to form a complete causal model and have exponential-time complexity. However, by only looking at relationships occurring between sets of three variables that are pairwise correlated, authors are able to provide a polynomial time algorithm in [SsBMU98]. In doing so, the algorithm is only able to find a very small subset of the relations between the attributes. Moreover, authors assume several other constraints on the underlying data the most important being the applicability of Markov Condition - *If A and B are nodes in a Bayesian Network and B is not a descendent of A in the network, then the Markov condition is said to hold if A and B are independent conditioned on the parents of A.* Since the algorithm does not find all relationships between attributes, any attribute relevance estimate obtained using such an algorithm would be highly erroneous to begin with and therefore we do not use it in AIMQ.

CHAPTER 6

EVALUATING ROBUSTNESS, EFFICIENCY AND ACCURACY OF AIMQ

In this chapter we present evaluation results showing the efficiency and effectiveness of AIMQ in answering imprecise queries. Specifically, we investigate the robustness of the estimated attribute importance and value similarities, evaluate the efficiency of the query relaxation process and verify the relevance of answers we suggest by conducting a user study. We used the online used car database *Yahoo Autos*[1] to evaluate our system.

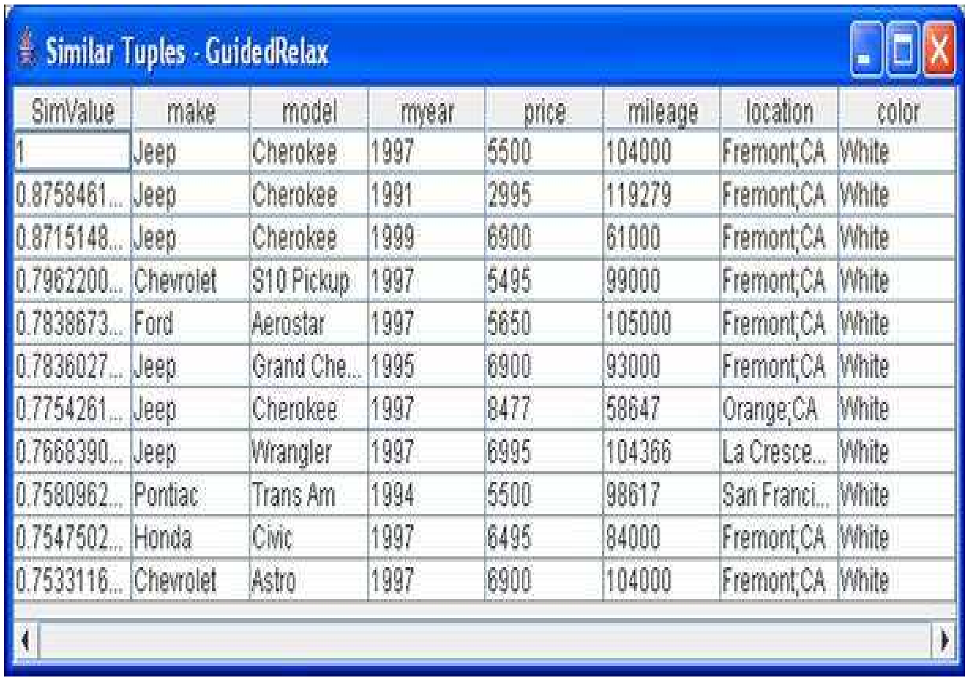
6.1. Experimental Setup

We set up a MySQL based used car search system that projects the relation *CarDB(Make, Model, Year, Price, Mileage, Location, Color)* and populated it using $100,000$ tuples extracted from *Yahoo Autos*. To populate CarDB we probed the *Yahoo Autos* database by generating probe queries as described in Section 5.2. The

[1]Available at http://autos.yahoo.com.

probe queries over Yahoo Autos bound the attributes *Make* and *Location*. We considered the attributes *Make, Model, Year, Location* and *Color* in the relation CarDB as being categorical in nature. Additional details about the testbed are given in the Appendix C. Figure 6 presents a ranked set of answers returned by AIMQ. The AIMQ system is written in Java. The evaluations were conducted on a Windows based system with 1.5GHz CPU and 768MB RAM.

Implemented Algorithms: We designed two query relaxation algorithms *GuidedRelax* and *RandomRelax* for creating selection queries by relaxing the tuples in the base set. *GuidedRelax* makes use of the AFDs and approximate keys and decides a relaxation scheme as described in Section 5.1.2. The *RandomRelax* algorithm was designed to mimic the random process by which users would relax queries. The algorithm randomly identifies a set of attributes to relax and creates queries.



SimValue	make	model	myear	price	mileage	location	color
1	Jeep	Cherokee	1997	5500	104000	Fremont;CA	White
0.8758461...	Jeep	Cherokee	1991	2995	119279	Fremont;CA	White
0.8715148...	Jeep	Cherokee	1999	6900	61000	Fremont;CA	White
0.7962200...	Chevrolet	S10 Pickup	1997	5495	99000	Fremont;CA	White
0.7838673...	Ford	Aerostar	1997	5650	105000	Fremont;CA	White
0.7836027...	Jeep	Grand Che...	1995	6900	93000	Fremont;CA	White
0.7754261...	Jeep	Cherokee	1997	8477	58647	Orange;CA	White
0.7668390...	Jeep	Wrangler	1997	6995	104366	La Cresce...	White
0.7580962...	Pontiac	Trans Am	1994	5500	98617	San Franci...	White
0.7547502...	Honda	Civic	1997	6495	84000	Fremont;CA	White
0.7533116...	Chevrolet	Astro	1997	6900	104000	Fremont;CA	White

Figure 6. Ranked Answers

6.2. Robustness of AIMQ

As pointed out in Section 5.2, both the similarity estimation and attribute importance estimation processes depend on the sample dataset extracted by probing. Hence, one could argue that our approach is susceptible to variations in the distribution of the probing queries and consequently on the amount of data extracted. Below we empirically show that while the absolute support for the AFDs and approximate keys does vary over different data samples, their relative ordering is not considerably affected.

6.2.1. Robust Attribute Importance Estimation.

Using simple random sampling without replacement we constructed three subsets of CarDB containing 15k, 25k and 50k tuples. Then we mined AFDs and approximate keys from each subset and also from the 100k tuples of CarDB. Using only the AFDs we computed the dependence of each attribute on all other attributes in the relation (see $Wt_{depends}$ in Equation 5.2).

The attributes *Price, Mileage* and *Location* did not appear in any consequent. Figure 7 shows the dependence of remaining attributes in CarDB. We can see that *Model* is the least dependent among the dependent attributes while *Make* is the most dependent. The dependence values are highest when estimated over the 100k sample and lowest when estimated over 15k sample. This variation (due to sampling) is expected, however the change in the dataset size does not affect the relative ordering of the attributes and therefore will not impact our attribute ordering approach.

Figure 8 compares the quality of approximate keys mined from the sample datasets to that mined over the entire CarDB database (100k). Quality of an approximate key is defined as the ratio of support over size (in terms of attributes) of the key. The quality metric is designed to give preference to shorter keys. Specifically, given two keys with same support we would pick the key with less number of attributes. In Figure 8, the approximate keys are arranged in increasing order of their quality in the database. Only 4 of the 26 approximate keys in the database are not present in the sampled datasets. These 4 keys have low quality and would not have been useful in query relaxation. The approximate key with the highest quality in the database also has the highest quality in all the sampled datasets. Thus, even with the smallest sample (15k) of the database we would have picked the right approximate key during the query relaxation process.

Value	**Similar Values**	**25k**	**100k**
`Make=Kia`	`Hyundai`	`0.17`	`0.17`
	`Isuzu`	`0.15`	`0.15`
	`Subaru`	`0.13`	`0.13`
`Model=Bronco`	`Aerostrar`	`0.19`	`0.21`
	`F-350`	`0`	`0.12`
	`Econoline Van`	`0.11`	`0.11`
`Year=1985`	`1986`	`0.16`	`0.18`
	`1984`	`0.13`	`0.14`
	`1987`	`0.12`	`0.12`

Table 3. Comparison of Value Similarities computed using 25k and 100k samples of CarDB

6.2.2. Robust Similarity Estimation. We estimated value similarities for the attributes *Make, Model, Year, Location* and *Color* using both the 100k and 25k

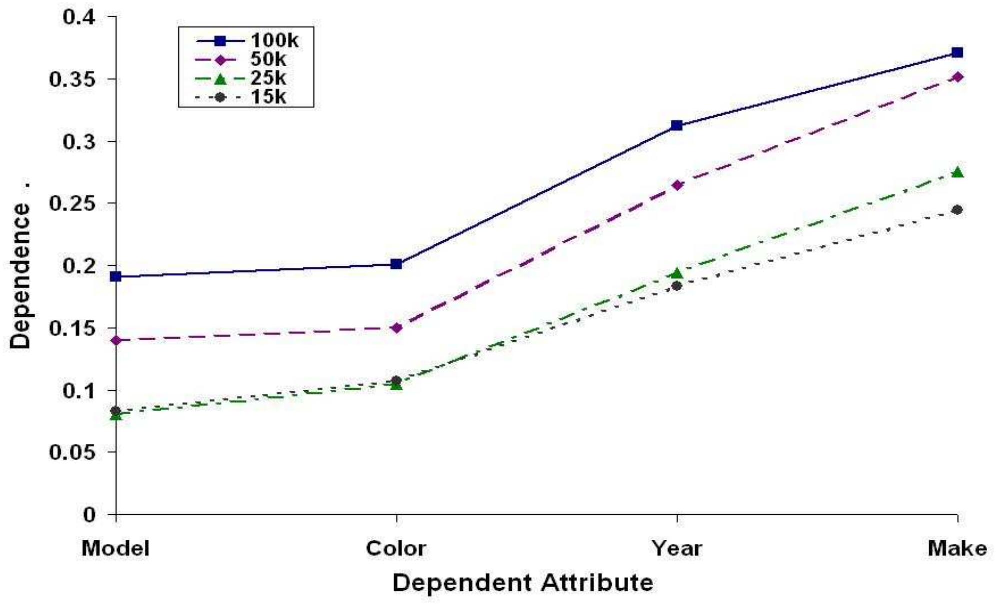


Figure 7. Robustness of Attribute Ordering

Algorithm Step	**Time-25k**	**Time-100k**
SuperTuple Generation	3 min	4 min
Similarity Estimation	15 min	26 min

Table 4. Computation Time for Value Similarity Estimation over CarDB

datasets. Time required for similarity estimation directly depends on the number of AV-pairs extracted from the database and not on the size of the dataset. This is reflected in Table 6 where the time required to estimate similarity over 100k dataset is only twice that of the 25k dataset even though the dataset size increased four times. Figure 9 provides a graphical representation of the estimated similarity between some of the values binding attribute *Make*. The values *Ford* and *Chevrolet* show high similarity while *BMW* is not connected to *Ford* as the similarity is below threshold.

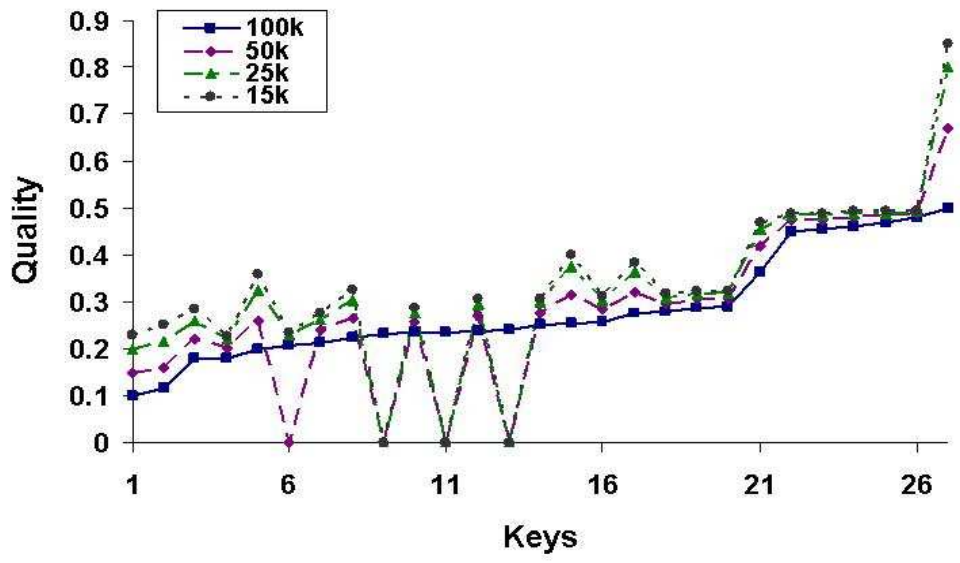


Figure 8. Robustness in mining Keys

We found these results to be intuitively reasonable and feel our approach is able to efficiently determine the distances between categorical values. Later in the section we will provide results of a user study that show our similarity measures as being acceptable to the users.

As shown above, the number of AV-pairs may vary depending on the size of the dataset used to learn the value similarities. Even though the missing values in smaller samples does affect the answers we suggest, we are able to correctly rank the values occurring in the sample dataset. Table 3 shows the top-3 values similar to *Make=Kia, Model=Bronco* and *Year*=1985 that we obtained from the 100k and 25k datasets. Even though the actual similarity values are lower for the 25k dataset,

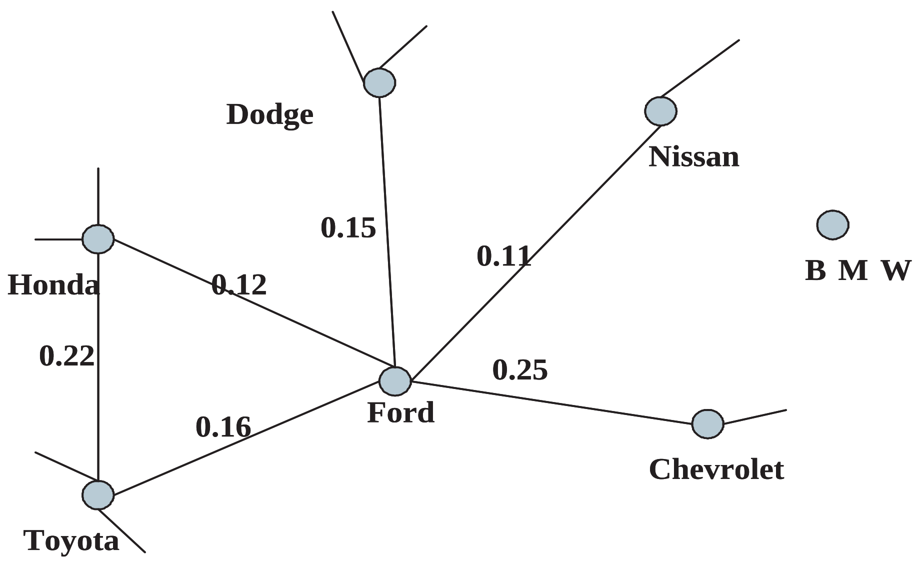


Figure 9. Similarity Graph for Make=Ford

the relative ordering among values is maintained. Similar results were also seen for other AV-pairs. Once again, we reiterate the fact that it is the *relative* and not the *absolute* value of similarity (and attribute importance) that is crucial in providing ranked answers.

6.3. Efficiency of query relaxation

To verify the efficiency of the query relaxation technique we presented in Chapter 5, 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 threshold T_{sim} ($0.5 \leq T_{sim} < 1$). To measure the efficiency

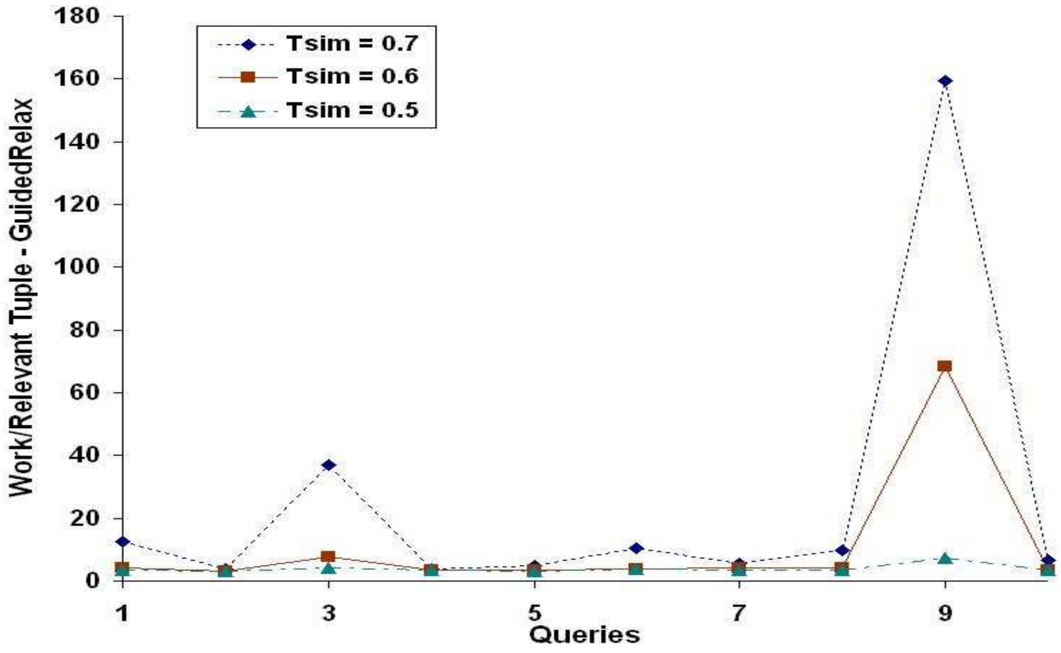


Figure 10. Efficiency of GuidedRelax

of our relaxation algorithms we used the metric

$$Work/RelevantTuple = \frac{|T_{Extracted}|}{|T_{Relevant}|} \tag{6.1}$$

where $T_{Extracted}$ gives the total tuples extracted while $T_{Relevant}$ is the number of extracted tuples showed similarity above the threshold T_{sim}. 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.

The graphs in Figure 10 and Figure 11 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

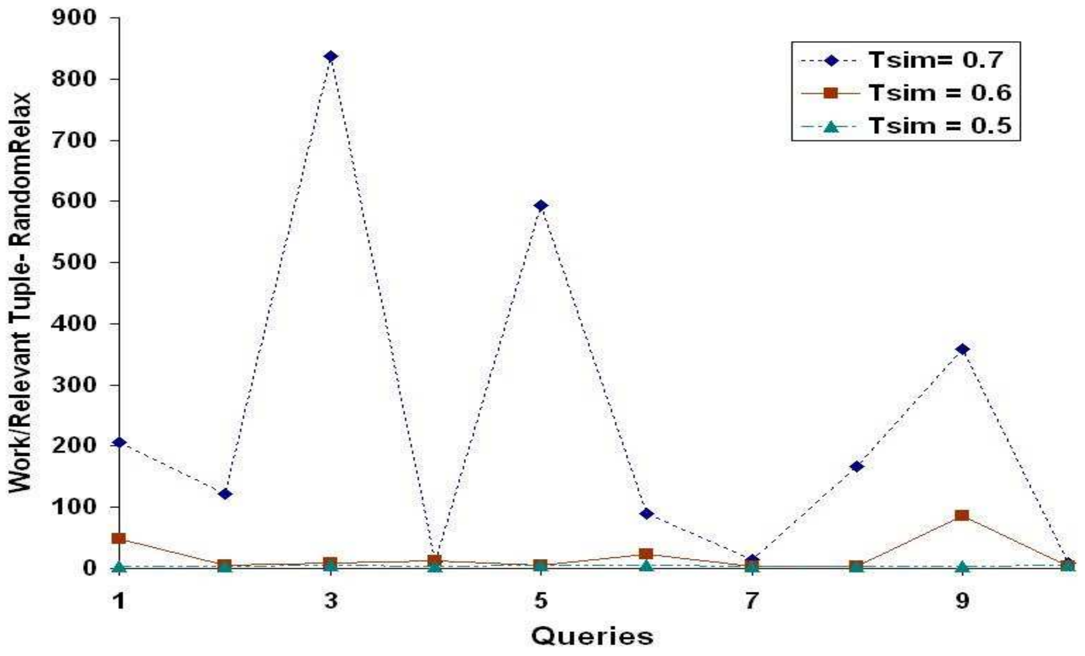


Figure 11. Efficiency of RandomRelax

this intuition, we note that for higher thresholds *RandomRelax* (Figure 11) ends up extracting hundreds of tuples before finding a relevant tuple. *GuidedRelax* (Figure 10) is much more resilient to the variations in threshold and generally needs to extract about 4 tuples identify a relevant tuple. Thus by using *GuidedRelax*, a user would have to look at considerably less number of tuples before obtaining satisfactory answers.

The evaluations presented above aimed to study the accuracy and efficiency of our query relaxation approach. However these experiments did not verify whether the tuples returned were acceptable to the user. In the next section we present results from a user study showing the high relevance of the answers we suggest.

6.4. Relevance of answers given by AIMQ

The results presented so far only verify the robustness and efficiency of the imprecise query answering model we propose. However these results do not show that the attribute importance and similarity relations we capture are acceptable to the user. Hence, in order to verify the correctness of the attribute and value relationships we learn and use, we setup a small user study over the used car database CarDB. We randomly picked 14 tuples from the 100*k* tuples in CarDB to form the query set. Next, using both the RandomRelax and GuidedRelax methods, we identified 10 most similar tuples for each of these 14 queries. For tuples extracted by RandomRelax we gave equal importance all attribute similarities. We used the 25k dataset to learn the attribute importance weights used by GuidedRelax. The categorical value similarities were also estimated using the 25k sample dataset. Even though in the previous section we presented RandomRelax as almost a “strawman algorithm”, it is not true here. Since RandomRelax looks at a larger percentage of tuples in the database before returning the similar answers it is likely that it can obtain a larger number of relevant answers (i.e. with higher similarity). The 14 queries and the two sets of ranked answers were given to 8 graduate student[2] volunteers. To keep the feedback unbiased, information about the approach generating the answers was withheld from the users. Users were asked to re-order the answers according to their notion of relevance (similarity). Tuples that seemed completely irrelevant were to be given a rank of zero.

[2]Graduate students by virtue of their low salaries are all considered experts in used cars.

Results of user study: We used MRR (mean reciprocal rank) [Voo99], the metric for relevance estimation used in TREC QA evaluations, to compare the relevance of the answers provided by RandomRelax and GuidedRelax. In TREC QA evaluations, the reciprocal rank (RR) of a query, Q, is decided as the reciprocal of the position at which the single correct answer was found. Thus, if correct answer is at position 1: RR(Q)=1, if at position 2: RR(Q)=$\frac{1}{2}$ and so on. If no answer is correct then RR(Q)=0. The MRR over a set of questions is the average of the reciprocal rank of each question. While TREC QA evaluations assume unique answer for each query, we assume a unique answer for each of the top-10 answers of a query. Thus we re-define MRR for a query Q as

$$MRR(Q) = Avg\left(\frac{1}{|UserRank(t_i) - SystemRank(t_i)| + 1}\right) \tag{6.2}$$

where t_i is i^{th} answer to Q. Figure 13 shows the average MRR ascribed to both the query relaxation approaches.

The higher average MRR value given to GuidedRelax for most queries points to the fact that the ranked answers provided by GuidedRelax were considered more relevant by the users. Thus, even though it only looks at fewer tuples of the database, GuidedRelax is able to extract more relevant answers. Thus, the attribute ordering heuristic is able to closely approximate the importance users ascribe to the various attributes of the relation. The overall high relevance (average MRR=0.4) given to the answers shows that the value similarities learned have high accuracy and are found relevant by the users. Evaluating the user study results in conjunction with those checking efficiency (Section 6.3), we can claim that AIMQ is efficiently able to provide ranked answers to imprecise queries with high levels of user satisfaction.

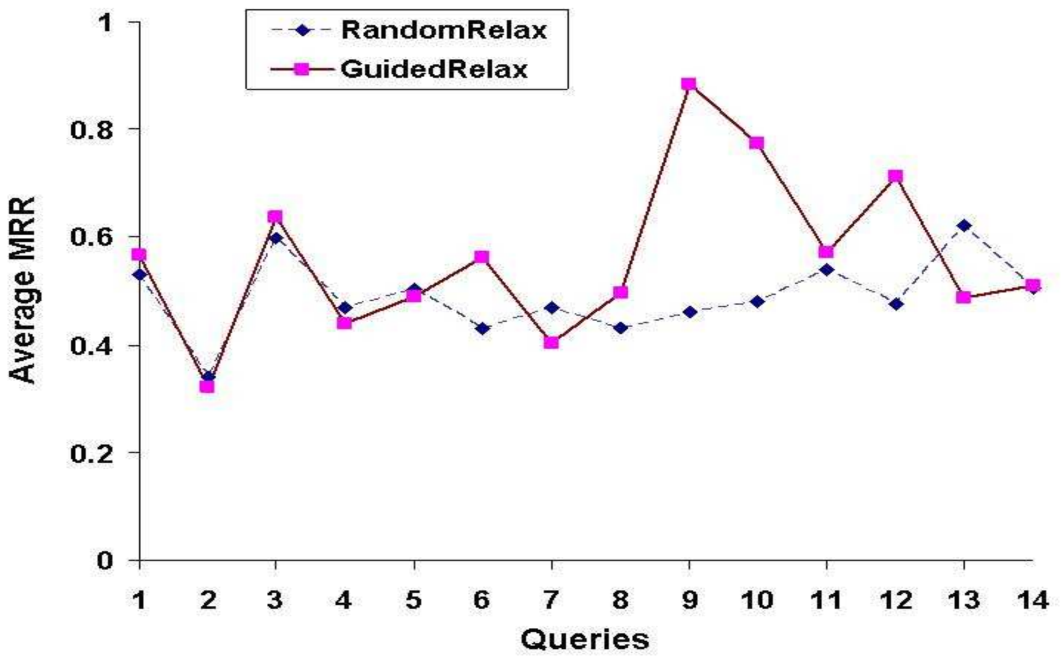


Figure 12. Average MRR for GuidedRelax and RandomRelax over CarDB

6.5. Summary and Discussion

We evaluated the efficiency and effectiveness of AIMQ in answering imprecise queries over CarDB, a database constructed from the online used car database *Yahoo Autos*. Both, the categorical value similarities estimation presented in Chapter 4 and the attribute importance learning and query relaxation algorithms presented in Chapter 5, require a sample of the database to learn the necessary statistics. As pointed out in Section 5.2.2, the probing phase required to extract samples may lead to inaccuracies in the estimated statistics.

To mitigate the concerns arising due to probing, we performed robustness analysis of the learning algorithms by using samples of CarDB. The closeness in statistics mined from the samples to those mined from the entire database proves that the AIMQ algorithms are resilient to the size of sample extracted during the probing phase. We acknowledge that such a conclusion is only possible if we assume normal or only a slightly skewed distribution of data in the database. But, if the database consists of a number of disjoint sets, then the statistics learnt from the samples may or may not accurately reflect the behavior over the entire database. However, we must point out that this drawback is expected for any learning algorithm that has to rely on a probed sample of the database. The solution in such a situation would be to use probing queries that span most subsets. However, that may lead to extracting the entire database. On the other hand, we could try to look at past usage of the database and learn the statistics for regions most often queried by users. Thus, database workloads when available can be effectively used to minimize the effects of sampling. Databases being autonomous may not provide their workloads, but AIMQ can maintain a log of queries issued by the users (both imprecise and precise) and use them to improve efficiency of the probing phase.

Results of the user study demonstrated high levels of user satisfaction for the imprecise query answers provided by our system. These results can also be seen as validating the value similarities estimated by the context sensitive semantic similarity estimation model developed in Chapter 4.

Why compare only with RandomRelax? In this chapter, we presented comparison results between our relaxation algorithm *GuidedRelax* and *RandomRelax* - an algorithm designed to mimic the random process by which users would relax queries. Specifically, given a tuple in the base set *RandomRelax* identifies a random relaxation order and uses it to relax queries. One critique that might arise regarding our evaluation is that *RandomRelax* is a strawman algorithm and so the comparison is not conclusive. We believe such an argument has no merit given the following two facts - (1) prior to GuidedRelax, there are is no domain-independent query relaxation algorithm and (2) in the absence of any domain knowledge, a user attempting to extract information from a database would be only a random search. Therefore, *RandomRelax* should be considered as the *benchmark* whose performance must be matched by any domain independent query relaxation algorithm. We look at several domain and/or user dependent relaxation algorithms in Section 9.2.

The most important component of any query relaxation algorithm is the heuristic used to determine the relaxation order of attributes. In AIMQ (for *GuidedRelax*) we use AFDs to derive such a heuristic. While domain-independent heuristics for estimating attribute importance and query relaxation are not available, several researchers have developed domain-independent heuristics for determining the importance of attribute values by looking at the underlying link structures [ABC$^+$02, HP02] or based on the frequency with which they occur in the database and queries [ACDG03]. Since, AIMQ assumes a single database relation, the link structure based approaches are not feasible. We discuss such approaches and highlight their limitations in Section 9.1.

The *QFIDF* measure introduced in [ACDG03] for measuring the *importance of an attribute value* is based on the much popular *TF-IDF* measure used in IR systems. In fact, *IDF* measure is in *QFIDF* mimicks IDF measure in IR by measuring how frequently the attribute value appears in all the tuples in the database. The *QF* measure computes the frequency of an attribute value in the entire workload - a log of past queries issued by users. Both *QF* and *IDF* are measuring popularity of the value in the workload and the database. However, popularity of a value as measured by both *QF* and *IDF* does not provide any information about how it interacts with values binding other attributes i.e. no dependence information between values can be obtained by looking their individual *QFIDF* values. Given the lack of dependence information, identifying relaxed queries by removing the least or most popular attribute value will not guarantee a non-empty resultset. The reason being that the bound attributes may often influence the choice of the binding value(s) for the free attribute(s). Thus, the relaxation heuristic based on *QFIDF* can give no more guarantees than *RandomRelax.* In fact, one can argue that the *QFIDF* based relaxation algorithm would be a type of random relaxation where the random number is replaced by the *QFIDF* value. Therefore, we used *RandomRelax* which is both domain and user independent to compare the performance of *GuidedRelax.*

CHAPTER 7

COMPARISON STUDY TO SHOWCASE DOMAIN-INDEPENDENCE AND CONSISTENCY

The robustness and efficiency of AIMQ system in answering imprecise queries has been clearly demonstrated by the evaluation results presented in the previous chapter, Chapter 6. Eventhough AIMQ did not make use of any domain specific information about used cars in answering imprecise queries over *CarDB*, to make a conclusive argument about the domain independence of AIMQ we must evaluate over additional domains. Therefore, in this chapter, we evaluate AIMQ using the *Census dataset* available from the *UCI Machine Learning Repository.* Furthermore, to further highlight the efficiency and robustness of AIMQ's algorithms, we compare the performance of AIMQ in terms of the relevance of answers suggested with that of an alternate imprecise query answering system that implements the *ROCK categorical value clustering* algorithm [GRS99].

7.1. Comparison Study: AIMQ versus ROCK

7.1.1. ROCK based Imprecise Query Answering System.

We set up another query answering system that uses the ROCK clustering algorithm to cluster all the tuples in the dataset and then uses these clusters to determine similar tuples. We chose ROCK to compare as it is also a domain independent solution like AIMQ and does not require users to provide distance metrics. ROCK[1] differs from AIMQ in the way it identifies tuples similar to a tuple in the base set. Specifically, given a tuple t belonging to the base set, ROCK first determines the cluster to which the tuple t belongs and then returns the most similar tuples from that cluster. Clustering using ROCK consists of extracting a small random sample from the database, applying the link based clustering algorithm on the sampled points and then assigning the remaining points from the dataset to the clusters. A detailed overview of the approach used by ROCK for clustering tuples containing categorical values is given in Appendix B.

7.1.2. Experimental Setup.

We used two real-life databases:- (1) the online used car database *Yahoo Autos*[2] and (2) the *Census Dataset* from UCI Machine Learning Repository[3], to compare the performance of AIMQ and ROCK.

Implemented Algorithms: We compare the performance of both the query relaxation algorithms, *GuidedRelax* and *RandomRelax*, described in the previous chapter with that of ROCK. The *RandomRelax* algorithm was designed to mimic the random process by which users would relax queries. The algorithm randomly identifies

[1]Henceforth, we use ROCK to refer to the query answering system using ROCK.
[2]Available at http://autos.yahoo.com.
[3]Available at http://www.ics.uci.edu/ mlearn/MLRepository.html.

Attribute	**Type**	**Distinct Values**
`Age`	`Numerical`	`NA`
`Workclass`	`Categorical`	`8`
`Demographic-weight`	`Numerical`	`NA`
`Education`	`Categorical`	`16`
`Marital-Status`	`Categorical`	`7`
`Occupation`	`Categorical`	`14`
`Relationship`	`Categorical`	`6`
`Race`	`Categorical`	`5`
`Sex`	`Categorical`	`2`
`Capital-gain`	`Numerical`	`NA`
`Capital-loss`	`Numerical`	`NA`
`Hours-per-week`	`Numerical`	`NA`
`Native-country`	`Categorical`	`41`

Table 5. Schema Description of CensusDB

a set of attributes to relax and creates queries. *GuidedRelax* makes use of the AFDs and approximate keys and decides a relaxation scheme as described in Algorithm 2. ROCK's computational complexity is $O(n^3)$, where n is the number of tuples in the dataset. In contrast, *AIMQ's* complexity is $\mathrm{O}(m \times k^2)$ where m is the number of categorical attributes, k is the average number of distinct values binding each categorical attribute and $m < k < n$. AIMQ and ROCK were both developed using Java. The evaluations were conducted on a Windows based system with 1.5GHz CPU and 768MB RAM.

The time required by AIMQ and ROCK to compute the necessary statistics is given in Table 6. The overall processing time required by AIMQ is significantly lesser than that for ROCK. We can clearly see that AIMQ is much more efficient than

	CarDB (25k)	**CensusDB (45k)**
AIMQ		
SuperTuple Generation	3 min	4 min
Similarity Estimation	15 min	20 min
ROCK		
Link Computation (2k)	20 min	35 min
Initial Clustering (2k)	45 min	86 min
Data Labeling	30 min	50 min

Table 6. Computation Time for AIMQ and ROCK over CarDB and CensusDB

ROCK. Note that the *Link Computation* and *Initial Cluster* identification phase of ROCK only uses very small sample (2000 tuples in our case) of the given datasets. Even though ROCK uses the same sized (2000 tuples) initial sample for both CarDB and CensusDB, the time required to compute links and derive clusters in CensusDB is almost twice that required in CarDB. The variation is due to the larger attribute space that ROCK has to search through in the CensusDB. Eventhough AIMQ also requires more computation time over CensusDB, the increase in time is negligible thereby demonstrating that AIMQ's learning algorithms are much less influenced by the number of attributes in the database.

7.2. Comparison Study using CarDB

To compare the performance of AIMQ with that of ROCK we began by setting up a small user study over the used car database CarDB. We used the database CarDB described in Section 6 as the first database over which to compare the performance of AIMQ and ROCK. As described earlier CarDB projects the relation *CarDB(Make,*

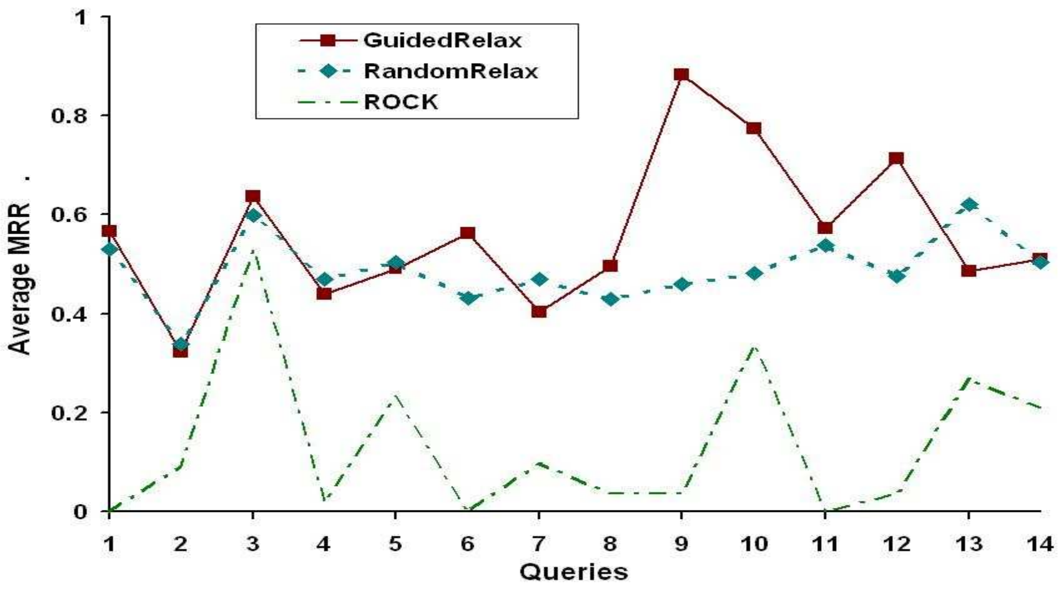


Figure 13. Average MRR of AIMQ and ROCK over CarDB

Model, Year, Price, Mileage, Location, Color) and is populated using $100,000$ tuples extracted from *Yahoo Autos*.

We randomly picked 14 tuples from the $100k$ tuples in CarDB to form the query set. Next, using both the *RandomRelax* and *GuidedRelax* methods, we identified 10 most similar tuples for each of these 14 queries. We also chose 10 answers using ROCK. We used the 25k sample of CarDB to learn the attribute importance weights used by *GuidedRelax*. The categorical value similarities were also estimated using the 25k sample dataset. Both *RandomRelax* and ROCK give equal importance to all the attributes and only differ in the similarity estimation model they use. The 14 queries and the three sets of ranked answers were given to 8 graduate student volunteers. To keep the feedback unbiased, information about the approach generating the answers was withheld from the users. Users were asked to re-order the answers according to their notion of relevance (similarity). Tuples that seemed completely irrelevant were to be given a rank of zero.

We again used MRR (mean reciprocal rank) [Voo99], the metric for relevance estimation used in TREC QA evaluations, to compare the relevance of the answers provided by AIMQ and ROCK. Figure 13 shows the average MRR ascribed to both the query relaxation approaches of ROCK. *GuidedRelax* has higher MRR than *RandomRelax* and ROCK. ROCK only ensures that the set of all "clusters" together optimize a criterion function but cannot guarantee that all tuples belonging to a cluster are equally similar. Moreover, ROCK uses equality as the measure to test similarity for categorical attributes. These limitations together lead to the poor overall performance of ROCK. Even though *GuidedRelax* looks at fewer tuples of the database, it is able to extract more relevant answers than *RandomRelax* and ROCK. Thus, the attribute ordering heuristic is able to closely approximate the importance users ascribe to the various attributes of the relation. Furthermore, the much higher relevance attributed to answers of *RandomRelax* than those of ROCK show that our similarity estimation model is superior to that employed by ROCK and is able to learn value similarities that are more acceptable to users.

7.3. Comparison Study using CesusDB

The results presented above do show that AIMQ is considerably better than ROCK - a contemporary domain independent categorical value similarity estimation

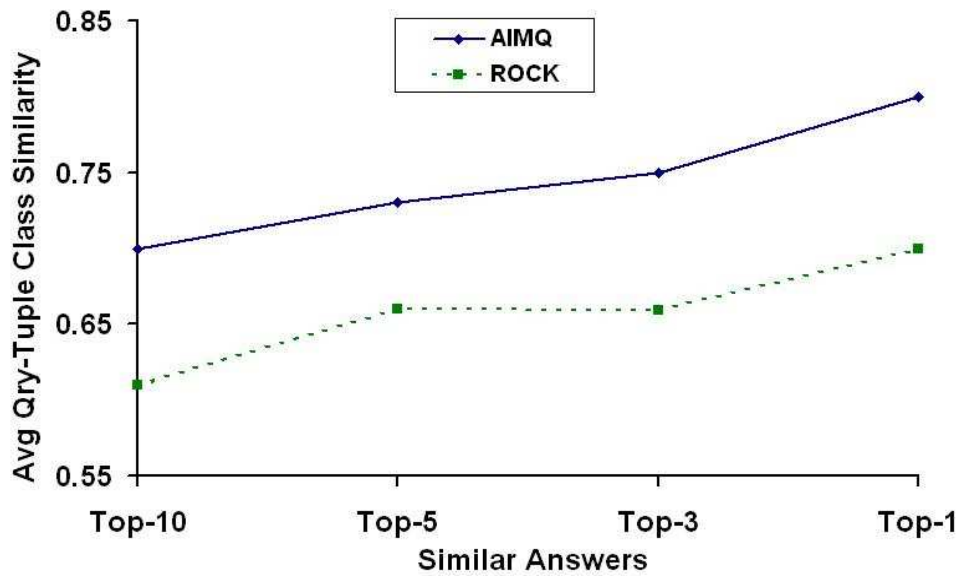


Figure 14. Classification Accuracy of AIMQ & ROCK over CensusDB

approach. However, one could argue that CarDB may be biased towards the heuristics used by AIMQ and hence ROCK performs badly over CarDB. Therefore, we also evaluated the performance of both the systems over the Census database, CensusDB.

The Census database we used projected the relation *CensusDB(Age, Work-class, Demographic-weight, Education, Marital-Status, Occupation, Relationship, Race,*

Value	**Similar Values**	**Similarity**
`Education=Bachelors`	`Some-college`	0.6
	`HS-Grad`	0.42
	`Masters`	0.27
`Workclass=Federal-Gov`	`State-Gov`	0.59
	`Local-Gov`	0.38
	`Private`	0.14

Table 7. CensusDB Value Similarities

Sex, Capital-gain, Capital-loss, Hours-per-week, Native-Country) and was populated with $45,000$ tuples provided by the *Census dataset*. Schema description of CensusDB with the count of number of distinct values binding the categorical values is given in Table 5. Each tuple was pre-classified into one of the two classes: (1) Income $> 50K$ and (2) Income $<= 50K$. Even though the tuples in the dataset were pre-classified, during the attribute importance estimation and value similarity learning phases we ignored the class labels. We used the classification to test the relevance of the similar answers returned by AIMQ and ROCK. Each tuple in the database contains information that can be used to decide whether the surveyed individual's yearly income is '$> 50k$' or '$<= 50k$'. An example user query over CensusDB could be

```
Q:-CensusDB(Education like Bachelors, Hours-per-week like 40)
```

The user issuing Q is interested in finding all individuals (tuples in CensusDB) who have *Education* similar to a *Bachelors* degree and work for *around* 40 hours a week. Since *Hours-per-week* is continuous valued we can use an L_p *metric* such as *Euclidean distance* to decide values close to 40. But, *Education* is a categorical attribute and hence determining values similar to *Bachelors* becomes a non-trivial task in the absence of a user-given distance metric. Moreover, other attributes like *Age, Occupation, WorkClass etc* also are important in determining similarity among individuals (tuples of CensusDB). Thus answering query Q would require learning both the importance to be ascribed to each attribute and the similarities between values binding the categorical attributes - two tasks that are efficiently and accurately accomplished by AIMQ.

To answer imprecise queries such as Q over CensusDB, we began by using a sample of $15k$ tuples of CensusDB to learn the attribute dependencies and categorical value similarities. AIMQ picked the approximate key *Age, Demographic-Weight, Hours-per-week* as the best key and used it to derive relaxation order shown in Figure 15.

```
Occupation → Capital-loss → Education → Relationship → Race →
Marital-Status → Sex → Capital-gain → Native-country →
Demographic-Weight → Hours-per-week → Workclass → Age
```

Figure 15. Attribute Relaxation Order in CensusDB

The ordering is intuitive since it is a known fact that on an average a more experienced person (higher age) would earn more. On the other hand, no particular occupation (e.g. *Sales, management*) will exclusively divide an income range and hence occupation alone cannot conclusively decide your yearly income. Therefore occupation must be have low importance as is reflected in the ordering. Table 7 shows a few similarities estimated by AIMQ between the values binding the categorical attributes.

We randomly selected 2000 tuples from CensusDB as the initial sample to give as input to ROCK. From the 2000 tuples, we were able to obtain 300 initial clusters. ROCK then assigned the remaining tuples to these 300 clusters.

We used the class information to check the relevance of the answers we provide. Specifically, tuples belonging to the same class are more similar than those in opposite classes. Therefore, we estimated the relevance of AIMQ's answers based on the number of answers having identical class as the query. We used 1000 tuples not appearing in the 15k sample as queries to test the system. The queries were equally distributed over the two classes. For each query, using *GuidedRelax* we identified the *first 10 tuples* that had similarity above 0.4. We also used ROCK to learn 10 answers to each of the test queries. Figure 14 compares the average classification accuracy of the *top-k* (where *k*={*10,5,3,1*}) answers given by both AIMQ and ROCK to each of the 1000 queries. We can see that the accuracy increases as we reduce the number of similar answers given to each query. Once again, AIMQ comprehensively outperforms ROCK in all the cases thereby proving that AIMQ is domain independent and that its learning algorithms are efficient and able to better model the value similarities.

7.4. Summary and Discussion

In this chapter, we compared the performance of AIMQ against a system that used the ROCK categorical value clustering algorithm to decide similarity between tuples. We used two databases *CarDB* and *CensusDB* to compare the performance of the two systems. The almost constant and considerably lesser time required by AIMQ in computing the necessary statistics over both domains demonstrates the efficiency, scalability and domain independence of AIMQ's algorithms.

The much higher relevance of answers given by AIMQ over CarDB and the higher classification accuracy of AIMQ's results over CensuDB show that AIMQ is able to return answers that are considerably more relevant to the users. This validates our claim that AIMQ is a domain independent solution and is applicable over a multitude of domains. The evaluation results presented in Chapter 6 and Chapter 7 provide substantial evidence that we have successfully achieved our goal of providing answers to imprecise queries without affecting any changes to the query answering model of the original database and more importantly, without requiring the users to provide any additional input other than the imprecise query itself. Eventhough, the high relevance of our suggested answers is heartening, it was never our intent to take the user out of the loop. We believe that adding some sort of user relevance feedback to the AIMQ approach can considerably improve the solution we present. However, asking users to grade the relevance of the answers they obtained would amount to burdening the users post the query execution phase. This would negate all the benefits of our approach. Hence, any feedback to be added must be obtained implicitly. A popular technique is use the database workloads to obtain feedback about the types of queries users issue over the database. With the assumption that the future queries will closely follow current trends, we can identify the popular queries and thereby the answers that users are interested in and provide these first when faced with a new query. Therefore, in the next chapter, Chapter 8, we present, AIMQ-Log, a system that extends AIMQ by adding implicit feedback to the imprecise query answering approach of AIMQ.

CHAPTER 8

AIMQ-Log: USING WORKLOAD TO CAPTURE USER INTEREST

The imprecise query answering approach, AIMQ, presented so far, was motivated by our desire to support imprecise queries over autonomous databases without changing the database or burdening the user with the need to provide much domain specific information apart from the query.

In this chapter we present a domain-independent approach, AIMQ-Log, that extends the AIMQ system to account for user interest. However, we are still interested in generating relevant results to an imprecise query without requiring the user to provide any domain specific information or any changes to the architecture of the underlying database.

Answering imprecise queries using AIMQ-Log involves mapping the given imprecise query to a set of precise queries obtained from a workload - log of past queries issued over the database. The set of precise queries relevant to the user given imprecise query are decided based on the similarity they show to the user query. The similarity between queries is estimated as the similarity among their answer tuples.

Under the relational data model, two queries are similar only if they contain same values for all their attributes or if their answer sets had common answer tuples. But two queries can be considered similar even if their tuples only match partially. We can consider the answers of a query as describing the context of the query and use the context-sensitive similarity estimation technique (Chapter 4) of AIMQ to measure the similarity of queries in the query log.

We implemented AIMQ-Log over BibFinder [NKH03], a publicly- available Web data mediator. Below, we describe the AIMQ-Log approach and provide results of a user study showing the high relevance of the precise queries we identify and thereby that of the answers suggested by AIMQ-Log.

8.1. Overview of AIMQ-Log

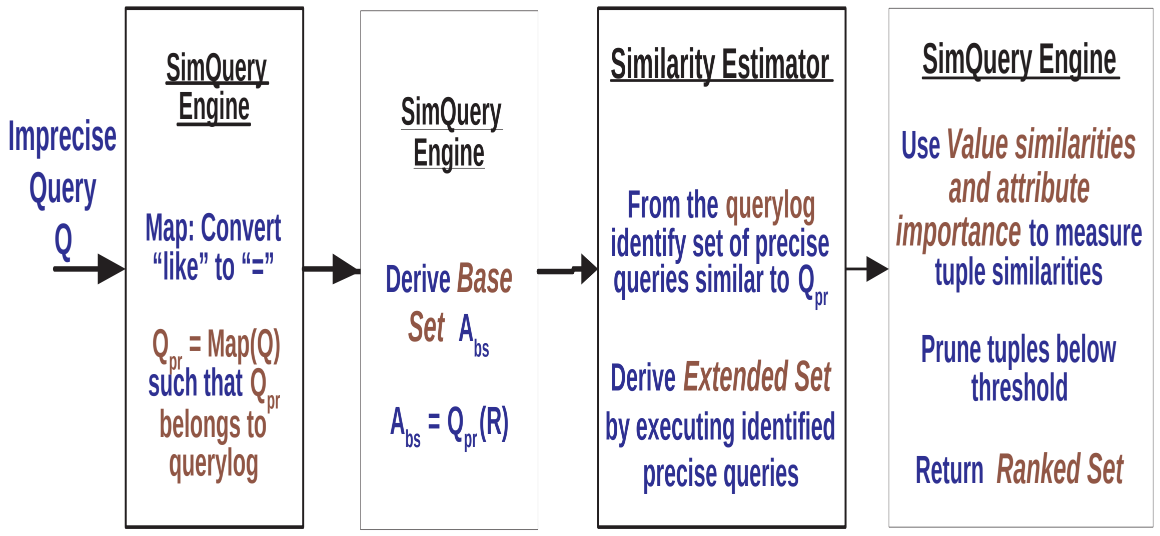

Imprecise Query Q
SimQuery Engine
Map: Convert "like" to "="
Q_pr = Map(Q) such that Q_pr belongs to querylog
SimQuery Engine
Derive Base Set A_bs
A_bs = Q_pr(R)
Similarity Estimator
From the querylog identify set of precise queries similar to Q_pr
Derive Extended Set by executing identified precise queries
SimQuery Engine
Use Value similarities and attribute importance to measure tuple similarities
Prune tuples below threshold
Return Ranked Set

Figure 16. FlowGraph of the AIMQ-Log approach

The AIMQ system's primary intent was minimizing the inputs a user has to provide before she can get answers for her imprecise query. However, in doing so, AIMQ fails to include users' interest while deciding the answers. A naive solution would be to ask user to provide feedback about the answers she receives. But doing so would negate the benefits of AIMQ. The ideal solution would be to obtain and use user feedback implicitly. Database *workloads* - log of past user queries, have been shown as being a good source for implicitly estimating the user interest [ACDG03]. Hence, we developed *AIMQ-Log* a system that differs from AIMQ in the way it determines the set of precise queries used to extract relevant answers from the database. Instead of the query relaxation approach followed by AIMQ, in AIMQ-Log we use the database workload to identify precise queries of interest. We only consider queries that occur frequently (i.e. more than a given threshold) in the workload for determining the relevant set. The more frequent a query, more likely it will be seen as relevant by a new user. The flowgraph of AIMQ-Log's query answering approach is given in Figure 16.

Given an imprecise query Q and the database workload Q_{log} containing past precise queries issued over R, AIMQ-Log begins by by identifying a precise query $\in Q_{log}$ that can be *mapped onto* Q. Query Q_{pr} is said to map into Q if the set of constrained attributes of Q_{pr} is a subset of the constrained attributes of the Q. Moreover both the precise and imprecise query constraints should have the same binding values. After determining Q_{pr}, we can then extract other queries from Q_{log} that are similar to the query Q_{pr}, thereby forming a set of precise queries whose

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 8. SuperTuple for query Author=Ullman

answers will be relevant to the imprecise query Q. That is,

$$Ans(Q) \approx Tuples(Q')$$

$$where \quad Q' \in Q_{log}, Sim(Q', Q_{pr}) > T_{sim}$$

As is the case with by AIMQ, the answer to Q will be an ordered union of the answers of the precise queries where the similarity of a tuple to the query Q is computed as explained in Chapter 4. We can obtain a sample of the database by by unifying the answersets of the queries we select from the query log. Then, as explained in Chapter 4 and Chapter 5 we can determine the attribute importance measures and value similarities required to measure the similarity of answer tuples to the imprecise query.

8.2. Generating Similarity Matrix

In this section, we explain how we compute the similarity between the queries. Suppose, *Author=Ullman* and *Author=Widom*are two queries on the relation *Publi-*

cations. 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. Thus, even though the queries (terms) are not the same, the fact that their contexts as defined by their resultsets are similar would make the queries similar. Hence, we can provide answers to the query *Author=Widom* to an user who issued a query *Author like Ullman*. 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 write papers on related topics. Thus, the answers to the query (and therefore the supertuple) can be seen as describing the context of the query. Hence, the similarity between two queries can be estimated using the context-sensitive similarity estimation approach described in Chapter 4.

Specifically, the similarity between two queries is measured as the similarity shown by their supertuples. As described in Chapter 4, we use the *Jaccard Similarity metric* [BYRN99] to estimate similarity between the supertuples. We computed the similarity between the queries using two similarity estimation metrics derived from Equation 4.2 based on how the importance weights were assigned. The derived metrics are

- *Doc-Doc similarity:* In this method, we gave equal importance to all attributes belonging to a supertuple. Thus, the supertuple then can be considered a document - a single bag representing all the values occurring in the supertuple.

The similarity between two queries Q_1 and Q_2 is then computed as

$$Sim_{dd}(Q_1, Q_2) = Sim_J(St_{Q1}, St_{Q2})$$

- *Weighted-Attribute similarity:* Here we use AIMQ's query-tuple similarity estimation function. The importance measures for each attribute were mined from a sample dataset obtained by unifying the answers for all popular queries in the query log. The similarity between two queries Q_1 and Q_2 is then determined as

$$Sim_{wa}(Q_1, Q_2) = VSim(St_{Q1}, St_{Q2})$$

Using Algorithm 3 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. A minimal similarity threshold T_{sim} is used to prune the number of queries found similar to a given query.

8.3. Evaluation

8.3.1. Experimental Setup. To evaluate the effectiveness of our approach in answering imprecise queries, we implemented AIMQ-Log over *BibFinder* [NKH03, Bib05]. *BibFinder* is a publicly-available Web data integration system that projects a unified schema over multiple bibliography databases. BibFinder provides a form-based interface and accepts queries over the relation

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

Algorithm 3 Creating Query Similarity Matrix

```
Require: Query log size - $n$, No of Attributes - $m$, Attribute Bags, Attribute Weights $W_{imp}$, Similarity Threshold $T_{sim}$
1: SimMatrix = null.
2: for $i = 1$ to n-1
3:    iBags = Get_Attribute_Bags(i).
4:    for $j = i + 1$ to n
5:       jBags = Get_Attribute_Bags(j).
6:       for $k = 1$ to m
7:          iDoc = Append(iDoc,iBags[k]).
8:          jDoc = Append(jDoc,jBags[k]).
9:          AtSim[k] = $\frac{|iBags[k] \cap jBags[k]|}{|iBags[k] \cup jBags[k]|}$
10:      $Sim_{dd} = \frac{|iDoc \cap jDoc|}{|iDoc \cup jDoc|}$
11:      if $(Sim_{dd} < T_{sim}) Sim_{dd} = 0$
12:         $Sim_{wa} = \sum_{k=1}^{m} AtSim[k] \times W_{imp}[k]$
13:      if $(Sim_{wa} < T_{sim}) Sim_{wa} = 0$
14:         SimMatrix[i][j] = [$Sim_{dd}$, $Sim_{wa}$]
15:      SimMatrix[j][i] = SimMatrix[i][j].
16: Return SimMatrix.
```

Several features of BibFinder validate the assumptions we made in this thesis. Queries over BibFinder are conjuncts of attribute-value pairs. Even though BibFinder integrates multiple Web data sources with varying query capabilities, it displays the behaviour similar to that of a Web database supporting boolean query answering model i.e. 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.

8.3.2. AIMQ-Log Architecture. The schematic diagram of AIMQ-Log as implemented over BibFinder is given in Figure 17. We add the following components

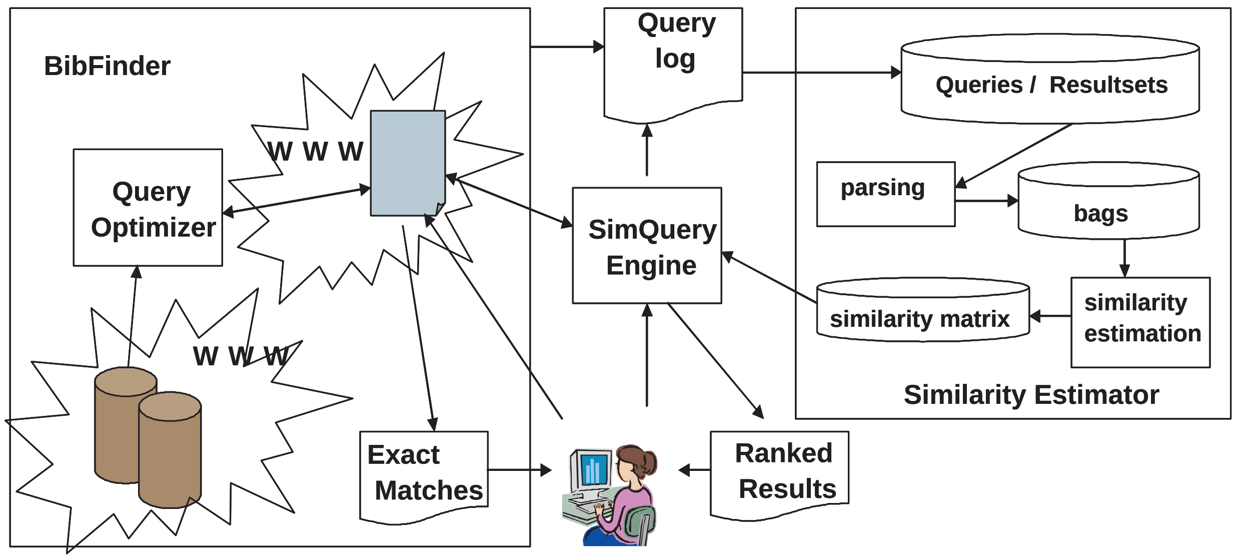


Figure 17. AIMQ-Log Implementation Over BibFinder

as a middle-ware between the user and BibFinder:

- A **SimQuery Engine**, that converts the given imprecise query into a set of precise queries. Given, an imprecise query Q, the SimQuery Engine begins by converting Q into a precise query Q_{pr}. Next it checks if Q_{pr} is present in the query log. If present, a list of queries that show similarity to Q_{pr} above the threshold T_{sim} is returned. If Q_{pr} is not present in the query log, then it will check if any subset (specialization) of Q_{pr} is present in the query log. The subset of Q_{pr} that is present in the query log is picked as the precise query that represents Q. SimQuery Engine then returns a ranked list of queries that show similarity above T_{sim} to the chosen subset of Q_{pr}. Results for the set of precise queries are then extracted from BibFinder and presented as a ranked list of tuples for the imprecise query Q.

- A **Similarity Estimator**, to calculate the similarity between each pair of queries in the query log. As shown in Figure 17, the Similarity Estimator begins by probing the database using the queries in the query log. The answers to each query are temporarily materialized for use in generating the query similarity matrix. Finally we estimate query similarities by computing the similarity between their supertuples.

We use the query log (workload) of BibFinder to identify the queries from which to learn the similarity matrix. We used 10000 queries from BibFinder's query log in our prototype system. We only picked queries that appeared more than 3 times in the query log. Table 9 lists the time taken and size of results produced at various stages of our similarity estimation 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.

Algorithm Step	**Time**	**Size**
`Probing`	`29 hours`	`35 Mb`
`Supertuple Generation`	`126 sec`	`21 Mb`
`Similarity Estimation`	`10 hours`	`6.0 Mb`

Table 9. Computation Time and Space Usage over BibFinder

We maintained a 10 second interval between each query issued over BibFinder. Hence the time required to probe BibFinder using queries from the query log is quite high in Table 9. Time to create the similarity matrix is high, as we must compare each query with every other query in the query log (see Algorithm 3). The complexity of the similarity estimation algorithm is $O(n^2)$, where n is the number of queries in the query log. We calculated both the *doc-doc* and *weighted-attribute* similarity between each pair of queries in the query log. The similarity threshold T_{sim} was set to 0.2 in our case since the resultsets were often very small and did not have very many features in common.

8.3.3. Evaluating the Relevance of Suggested Answers. To determine the correctness of the queries we suggest as being similar, we setup a user study. We asked our graduate student volunteers (8 of them) to evaluate the relevance of the queries we suggest. Each student was provided with a GUI 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. The user's were given 30 sample queries. For each imprecise query issued by the user, he/she had to determine how many among the top-10 similar queries they considered relevant. Also users were asked to report whether they found a query Q' relevant to Q based on:

- Q' having relevant terms: The values binding attributes of Q' are related to the those in Q. For example, the term *e-learning* is relevant to *Web-based learning* and hence queries containing these terms would be considered relevant to each other.

- Q' having relevant results: The results of Q' are relevant to Q, although no terms binding Q' are found relevant to Q. For example, the query *Author=Jaiwei Han* has results relevant 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. The same set of queries were used to decide accuracy of queries identified using both the doc-doc and the weighted-attribute similarity.

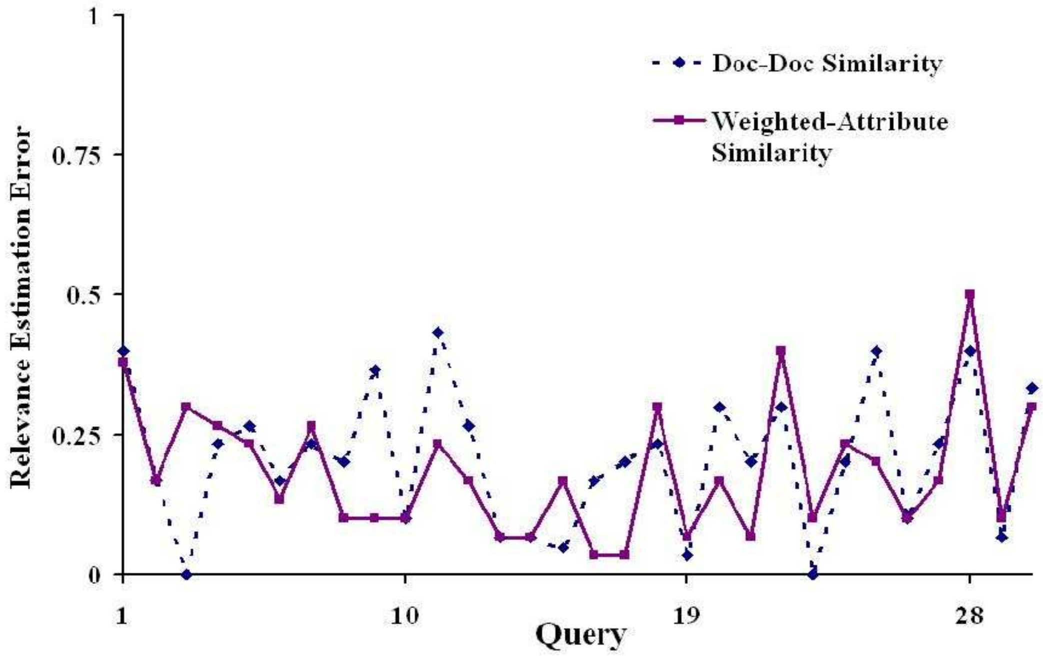


Figure 18. Error in Estimating Top-10 Similar Queries in BibFinder Querylog

Table 10 contains a sample set of queries recommended as being similar to three imprecise queries. Figure 18 illustrates the error in estimating the *top-10* relevant precise queries to a given imprecise query. The error is calculated as the number of

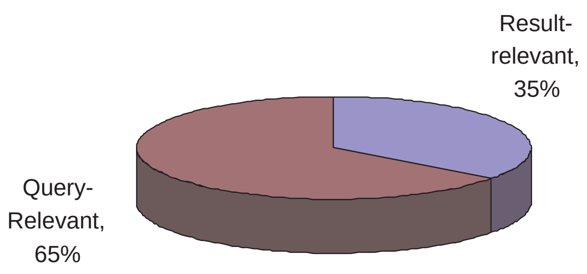


Figure 19. Relevance classification

queries among the *top-10* that were classified as not relevant by the user. The error is calculated as,

$$Error(Related(Q)) = 1 - Precison(Related(Q))$$
$$= \frac{10 - Relevant(Related(Q))}{10}$$

Both doc-doc and weighted-average show less than 25% average loss of precision with weighted-average showing better performance in general than doc-doc where all attributes were given equal importance. The high relevance of suggested answers (75% acceptable to users) demonstrates that the implicit user feedback mined from the workload is able to improve the ability of AIMQ framework to better model user's notion of relevance of answers.

Imprecise Query	**Title like web-based learning**
Related Queries	Title=E Learning
	Title=Web Technology
	Conference=WISE
Imprecise Query	**Title like Information Extraction**
Related Queries	Title=information filtering
	Title=Text Mining
	Title=Relevance Feedback
Imprecise Query	**Author like Abiteboul**
Related Queries	Author=vianu
	Author =Dan Suciu
	Author=Rakesh Agarwal

Table 10. Similar Queries from BibFinder's Querylog

Figure 19 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, but the query was not relevant. 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 weightage.

8.4. Summary and Discussion

We introduced a domain-independent approach, AIMQ-Log, that extends the AIMQ system presented earlier in the thesis for answering imprecise queries by adding implicit user feedback to the query answering process. The user feedback is added by picking the set of precise queries relevant to the given imprecise query from a set of frequent queries appearing in the database workload.

To evaluate the effectiveness of our approach, we performed experiments over a fielded autonomous Web database system, BibFinder. Results of a user study of our system show high levels of user satisfaction for the imprecise query answers provided by our system. The system was implemented without affecting the existing database thereby showing that it could be easily implemented over any existing databases.

A Hybrid Imprecise Query Answering System: The only constraint facing AIMQ-Log is the need to access the workload of a database. Since in this thesis we assume databases to be autonomous in nature, we cannot assume their workloads to be accessible to us. Therefore, AIMQ-Log cannot be seen as a solution that can be implemented directly over any existing system. Moreover, AIMQ-Log based system has the drawback that an imprecise query that cannot be mapped to the workload becomes unanswerable. On the other hand, AIMQ does not exploit the implicit feedback available from the workload. Therefore, we believe that any system for answering imprecise queries will be a *hybrid* of both AIMQ and AIMQ-Log, say $AIMQ_H$. Below we will briefly describe how an imprecise query will be answered by such a hybrid system.

On receiving an imprecise query Q, $AIMQ_H$ will first map Q to a precise query Q_p. If Q_p is one of the queries in the workload then based on the AIMQ-Log approach, we can identify a set of precise queries similar to Q_p and obtain relevant answers for query Q. If Q_p is not present in the workload, then using the AFD based query relaxation technique $AIMQ_H$ can derive a set of relaxed precise queries and for all the queries in the workload derive similar queries using the AIMQ-Log approach and extract relevant answers for Q_p. Note that this procedure allows us to avoid the need to execute the query Q_p over the database and performing multiple relaxations, one over each answer tuple of Q_p. Moreover, by picking similar queries from the workload ensures that every query will produce answers that are guaranteed to be similar to the query Q. In the unlikely event that none of the relaxations of Q_p are present in the workload, we can use the AIMQ approach to obtain relevant answers. Having the hybrid approach mitigates the problem of not being able to queries not present in the workload.

Eventhough the AIMQ-Log system and consequently $AIMQ_H$ only assumes the presence of a workload of precise queries, we can easily extend the system to make use of a log of imprecise queries once they become available. However, unlike the precise query log, the imprecise query log will need to maintain a list of precise queries issued to answer each imprecise query. Once such a log of imprecise queries becomes available, instead of looking for a precise query that maps to the given imprecise query and also appears in the precise query log, we can start by checking if the given query appears in the imprecise query log and quickly find the answers.

CHAPTER 9

RELATED WORK

The fundamental assumption of the relational models is that all data is represented as mathematical relations, i.e., a subset of the cartesian product of n sets. In this model, reasoning about the data is done in two-valued predicate logic, meaning there are two possible evaluations for each proposition: either *true or false.* Thus users of these systems are expected to formulate *precise queries* (queries evaluated using two-valued logic) that accurately captures their information need. But, often users may find it difficult to convert their information need (usually not clearly defined) into a precise query. Such users could be better served by providing a ranked set of answers from which they may be able to identify relevant answers. In contrast, IR systems tend to provide ranked sets of answers based on various notions of relevance. In this dissertation we looked at adapting techniques used in IR for supporting ranked retrieval of items that are relevant to the user query issued over an autonomous database using the relational model. We are not the first to attempt the task of combining IR and DB systems. A rich body of work relating to the integration of IR and DBMS systems exists although their solutions do not solve the problem of supporting imprecise queries over autonomous databases. Below we briefly describe several recent research efforts that have attempted to integrate IR and database techniques and/or tried to ease the difficulties faced by lay users when trying to extract information from a database.

Early approaches for retrieving answers to imprecise queries were based on theory of fuzzy sets. Fuzzy information systems [Mor90] store attributes with imprecise values, like height= “tall” and color=“blue or red”, allowing their retrieval with fuzzy query languages. The WHIRL [Coh98] system provides ranked answers by converting the attribute values in the database to vectors of text and ranking them using the vector space model. In [Mot98], Motro extends a conventional database system by adding a *similar-to* operator that uses distance metrics given by an expert to answer vague queries. Binderberger [OB03] investigates methods to extend database systems to support similarity search and query refinement over arbitrary abstract data types. In [GSVGM98], Goldman et al propose to provide ranked answers to queries over Web databases but require users to provide additional guidance in deciding the similarity. However, [OB03] requires changing the data models and operators of the underlying database while [GSVGM98] requires the database to be represented as a graph. In contrast, our solution provides ranked results without re-organizing the underlying database and thus is easier to implement over any database.

The problem of answering imprecise queries is related to three other problems. They are (1) *Empty answerset problem*- where the given query has no answers and needs to the relaxed. In [Mus04], Muslea focuses on solving the empty answerset problem by learning binding values and patterns likely to generate non-null resultsets. The relaxation is done by identifying a rule that best matches the given query but is a generalization of the query. However, the author considers all tuples to be equally relevant and therefore does not provide any criteria for ranking the results. Cooperative query answering approaches have also looked at solving this problem by identifying generalizations that will return a non-null result [Gas97]. More details about cooperative query answering techniques is given in Section 9.2. (2) *Structured query relaxation* - where a query is relaxed using only the syntactical information about the query. Such an approach is often used in XML query relaxation e.g. [SAYS02]. (3) *Keyword queries in databases* - Recent research efforts [ABC$^+$02, HP02] have looked at supporting keyword search style querying over databases. These approaches only return tuples containing at least one keyword appearing in the query. The results are then ranked using a notion of popularity captured by the *links*. A detailed discussion about these approaches is given below in Section 9.1.

The imprecise query answering problem differs from the first problem in that we are not interested in just returning some answers but those that are likely to be relevant to the user. It differs from the second and third problems as we consider the semantic relaxations rather than the purely syntactic ones.

9.1. Keyword Search in Databases

At first look, the keyword-based search popularized by Web search engines [Eng05b, Eng05c, Eng05a], where a user can specify a string of keywords and expect to retrieve relevant documents, possibly ranked by their relevance to the query may seem quite connected to our work. However, search engines only use the syntactic similarity between the query and the document to determine the relevance of the document. Hence only documents that exactly match the query is provided as an answer. Thus the queries accepted by search engines are precise queries. Therefore a search-engine user is still faced with the problem of having to formulate a query that precisely specifies her needs. To get a satisfactory answer, the user may have to iteratively refine her query. Clearly keyword-based search does not solve the problem we solve. In fact, our solution can be extended to enable search engines to accept imprecise queries.

Several recent research efforts [ABC$^+$02, HP02, GSVGM98, ACDG03] have looked at supporting keyword search style querying and/or providing ranked answers over databases. Both Banks [ABC$^+$02] and Discover [HP02] focus on using primary/foreign key relationships to determine proximity of the tuples across relations. In [ABC$^+$02], a database is viewed as a graph with objects/tuples as nodes and relationships as edges. Relationships are defined based on the properties of each application. For example an edge may denote a primary to foreign key relationship. The answers to keyword queries are provided by searching for Steiner trees [1] that contain all keywords. Heuristics are used to approximate the Steiner tree problem. A drawback of this approach is that a graph of the tuples must be created and main-

[1]The Steiner tree problem is a NP-complete combinatorial optimization problem in mathematics. It involves a graph G with weighted edges and some vertices designated as terminals. A Steiner tree is a subtree of G that connects the terminals with the minimum total weight (sum of weights of all edges in the tree). This problem is similar to minimum spanning tree where but there we are looking for a tree that connects all vertices of G. The Steiner tree problem has applications in circuit layout or network design. Although NP-complete, some restricted cases can be solved in polynomial time.

tained for the database. Furthermore, the important structural information provided by the database schema is ignored and the algorithms work on huge data graphs. In contrast, [HP02] uses the properties of the schema of the database. Its algorithms work on the schema graph, which is much smaller than the data graph, and does not need to keep any extra data representations. It exploits the properties of the database schema to produce the minimum number of SQL queries needed to answer to the keyword query. However this approach requires the underlying database to export schema information. Thus it is not applicable over autonomous databases that only provide a query interface (e.g. Web-enabled databases). In [GSVGM98], the user query specifies two sets of objects, the *Find set* and the *Near set*. These objects may be generated from two corresponding sets of keywords. The system ranks the objects in *Find set* according to their distance from the objects in the *Near*. An algorithm is presented that efficiently calculates these distances by building hub indices. While their approach is useful when the structure is unknown, they still rely on access to database internals which may not be possible over autonomous databases. Further they pre-compute the distances between nodes to determine the proximity and do not consider the possibility of updates to the underlying data changing the distances. Moreover given that most users will lack knowledge about the schema and organization of data it is highly impossible for a user to specify which objects are close to the objects of interest. In [ACDG03], the authors give an approach to provide $top-k$ answers to a relational query by determining the similarity between the query and each tuple. Authors develop two ranking functions: *IDF similarity* - based on cosine similarity with tf-idf weighting used in IR and *QF similarity* - based on occurrence frequencies of values in workload queries. Providing a ranked set of results would entail a system to look at each tuple in the database to determine its similarity to the query. In [ACDG03], authors avoid this problem by assuming availability of indexes on attributes that allow sorted access to the values. The authors then use a modified version of Fagin's Threshold Algorithm and its derivatives [FLN01, BGM02] to retrieve the top-k tuples without accessing all the tuples in the database.

9.2. Query Refinement

Research done in the context of cooperative query answering can be seen as being close to our work. Grice [Gri75] defines a cooperative answer as being correct, non-misleading and useful. Most research in cooperative answering tries to follow this definition. Initial work in cooperative answering focussed on natural language dialogue systems and employed natural language interfaces [Jos82, JWS81, PHW82]. Our work on the other hand can be seen as adding a similarity predicate to a relational query language without explicitly modifying the language. Cooperative answering techniques for databases have been considered in [CCL91, CCL92, Mot86]. Motro [Mot86] considers modifications to the relational model which would allow for cooperative behaviors in a relational database. In [Mot90], Motro describes a user interface for relational database systems that would allow users to interact with the system in more cooperative ways. The databases can then correct queries that have apparent mistakes or that return no answers.

Query generalization is also a form of cooperative answering, where the scope of the query is extended so that more information can be provided in the answers. Chu, Chen and Lee [CCL91, CCL92] explore methods to generate new queries related to the user's original query by generalizing and refining the user queries. The new queries can then be used to find answers which may be of interest to the user but not in the scope of the original query. The abstraction and refinement rely on the database having explicit hierarchies of the relations and terms in the domain. A generalized query is created by replacing relations/terms with corresponding values higher up in the hierarchy while replacing with terms lower in the hierarchy gives a refined query. In [Mot90], Motro proposes allowing the user to select directions of relaxation, thereby indicating which answers may be of interest to the user.

While the above approaches for cooperative answering focus on improving the interaction mechanisms between user and the system, we focus on allowing the user to obtain satisfactory answers without any repetitive refinement of the query. Further we do not require any domain-dependent information like term hierarchies to provide related answers.

9.3. Measuring Semantic Similarity

A contextual hypothesis of semantic similarity was first investigated in a domain of 65 synonymous nouns by Rubenstein and Goodenough [RG65], using rating and co-occurrence methods of semantic and contextual similarity. In a co-occurrence test of contextual similarity, one lists all words present in the context set of item

A and all words present in the context set of item B; then one computes the normalized coefficient representing the proportion of words common to both lists. The more words the lists have in common, the higher the correlation coefficient is and the more similar the two context sets are. Thus the authors concluded that *"similarity of context is reliable as a criterion for detecting pairs of words that are very similar"*. Miller and Charles [MC91] selected 30 of those pairs, and studied semantic similarity as a function of the contexts in which words are used. Their results further strengthen the context-sensitive view of semantic similarity.

In fact, Chales [Cha00] states that *"the results presented in [MC91] suggest that to know the meaning of a word is to know how to use it, and that encounters with the natural linguistic contexts of a word yield a contextual representation that summarizes information about the contexts in which the word may be used. The notion of a contextual representation of a word borrows from a common argument in linguistics and in psychology that most word meanings are derived from (linguistic) contexts."* On similar lines, Morris and Hirst [MH04] show that while most research on estimating semantic relations have been context-free i.e. the relations are considered out of any textual context and are then assumed to be relevant within textual contexts, empirical evaluations show that semantic similarity (and even meronymy, antonymy etc.) is often constructed in context, and cannot be determined purely from an apriori lexical resource such as WordNet or any other taxonomy.

Resnik [Res95] provides an alternate interpretation of context sensitive nature of word similarity. Specifically, he suggests that the behavior of one word can be approximated by smoothing its observed behavior together with the behavior of words to which it is similar. For example, a speech recognizer that has never seen the phrase *ate a peach* can still conclude that *John ate a peach* is a reasonable sequence of words in English if it has seen other sentences like *Mary ate a pear* and knows that *peach* and *pear* have similar behavior. Resnik [Res95] claims that as in information retrieval, the corpus based similarity estimation techniques can also use the "feature" representation of a word and compute the similarity between words by computing distance in a highly multi-dimensional space. However, he cautions that the difficulty with most distributional methods is that of interpreting the measured similarity. He argues that although word classes resulting from distributional clustering are often described as "semantic", they may often capture syntactic, pragmatic or stylistic factors as well.

Support for the contextual approach to measuring semantic similarity also comes from the computational model of meaning known as latent semantic analysis (LSA) [Lau98]. LSA is a theory for extracting and representing the contextualized meanings of words by statistical computations over a large corpora of text. The main idea is that the aggregate of all the word contexts in which a word can and cannot occur provides a set of constraints that determines the similarity of meaning of words. Cell entries in a matrix of rows and columns symbolize the frequencies with which a word appears in texts or contexts denoted by a column. LSA then uses singular value decomposition to infer semantic representations from the text.

9.4. Categorical Value Similarity Estimation

Clustering consists of partitioning a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often similarity or proximity for some defined distance measure. However, as we show below, compared to the similarity mining approach of AIMQ, these solutions are inefficient and/or require much input from the user. Recently the problem of clustering categorical data has received much attention [DMR98, GRS99, GGR99, GKR98]. In [GRS99], a hierarchical clustering algorithm, ROCK, that optimizes a criterion function defined in terms of number of "links" (neighbours) between tuples is given. In Chapter **??** we compared AIMQ with ROCK and presented results that show AIMQ is more efficient and provides answers with higher relevance. In [GKR98], STIRR, an iterative algorithm based on non-linear dynamical systems is used to isolate two groups of attribute values with large positive and small negative values that correspond intuitively to projections of clusters on the attribute. However, the authors of [GGR99] show that STIRR is unable to capture certain classes of clusters and therefore the corresponding value similarities. In [GGR99] authors assume attribute independence and monotonicity property to compute clusters using a Apriori style association mining algorithm. Unfortunately real-life data rarely satisfies the attribute independence assumption and hence this assumption affects the quality of the clusters given by [GGR99]. Das et al [DMR98], compute the similarity between two attribute values by generating probe queries binding features (other attributes) defining the values. However, [DMR98] requires users to choose the features of importance - a task that we believe is often difficult and/or undesirable for a lay user. In contrast to above approaches, AIMQ, without assuming independence between attributes or requiring users to provide the importance of an attribute, provides an efficient, domain and user independent solution for answering imprecise queries.

CHAPTER 10

CONCLUSION & FUTURE WORK

10.1. Conclusion

This dissertation has motivated the need for supporting imprecise queries over databases. The dissertation presents AIMQ, a domain independent approach for answering imprecise queries over autonomous databases.

Supporting imprecise queries over autonomous databases entails overcoming critical challenges like developing techniques for efficiently extracting relevant tuples and measuring the similarity of the answers to the query. Overcoming these challenges necessitated developing techniques for estimating the *importance to be ascribed to each attribute* and for *measuring the semantic similarity between values binding categorical values.* As part of AIMQ, we have developed techniques for:

- *Learning Attribute Importance:* We mine and use approximate functional dependencies between attributes from a small sample of the database to identify the importance to be ascribed to the attribute. This measure is then used to derive a heuristic to identify queries whose answers are likely to have high relevance to the given imprecise query.

- *Measuring Semantic Similarity:* We use an IR-style feature representation to develop a context sensitive semantic similarity estimation technique for values binding categorical attributes. We present a structure called *Supertuple* that represents the context in which a value appears in the database. The attribute importance estimated by AIMQ is used in weighing the similarity shown by different features describing a value.

As part of this dissertation, we have implemented the AIMQ system and evaluated its efficiency and effectiveness using three real-life databases,*BibFinder, Yahoo Autos and Census database*. The evaluation results demonstrate that:

- AIMQ is able to overcome inaccuracies that arise due to sampling and is able to efficiently learn the attribute importance and value similarity measures.
- AIMQ is able to provide answers to imprecise queries with high levels of user satisfaction as seen from the results of the user studies.
- The solution provided by AIMQ is truly domain-independent and can be applied to any autonomous database irrespective of the domain being represented.

To the best of our knowledge, AIMQ is the only domain independent system currently available for answering imprecise queries. It can be (and has been) implemented without affecting the internals of a database thereby showing that it could be easily implemented over any autonomous Web database.

10.2. Future Work

In this dissertation we have focussed only on answering imprecise queries over a single autonomous Web database. But more than one Web database may project a given relation. Even if all these systems were extended to support imprecise queries, it is not feasible for the user to query all of them to obtain most of the relevant answers. Hence a single point of contact where the user can issue the imprecise query and retrieve a single ranked set of answers from all the databases becomes essential.

Mediators in data integration systems do provide a single point of contact but for efficiently answering precise queries. The imprecise query answering approach we proposed aims to identify similar answers by issuing a set of precise queries that are similar to the query. Hence on first look, a mediator based system over the databases of interest may seem to solve our problem. In fact, in Chapter 8, we have implemented an extended version of AIMQ over the bibliography mediator BibFinder. For each precise query given to such a mediator, it would generate an execution plan that will have the highest net utility (e.g. high coverage and low cost). However, the best plan for answering the precise query developed may not be the best plan to execute to obtain answers relevant to the imprecise query. Therefore, a future extension of our work is to support imprecise queries over a mediation system. Supporting imprecise queries over multiple sources (mediator) will involve determining the number of relevant tuples each database is likely to contain for every possible imprecise query. Since learning and storing the statistics for all possible imprecise queries is infeasible, we will have to learn the same for classes of imprecise queries. The second step is to provide an efficient plan for searching the databases to efficiently identify the best answers for a given imprecise query. The solutions we develop will have to satisfy the following constraints:

- The databases will show varying degrees of overlap among each other. Duplicate elimination is considered a costly operation.
- We will return only the best k (top-k) relevant answers for any imprecise query. The relevance measures we use are system designed and not that given by a user.
- Only tuples having similarity scores above some pre-decided threshold δ is considered as relevant. Determining whether a tuple is relevant or not is a costly operation (in terms of time).

The cost of determining the k best answers for an imprecise query Q over R will be

$$Cost(Q(R)) = \sum_{i=1}^{n} Cost(IsRelevant(t_i, \delta) + Duplicate(t_i)) + \sum_{k=1}^{l} ExecutionCost(q_k(R))$$

where n is the number of distinct tuples extracted from which k best answers are obtained while l is the number of precise queries issued to obtain the k answers. Thus, we can reduce the cost of obtaining the top-k answers by reducing the number of irrelevant tuples and duplicates. Further reduction in cost can be obtained by reducing the number of precise queries issued to obtain the top-k answers. Intuitively, we must first call sources that return a large percentage of relevant answers. However, we should not invoke databases whose answers would overlap with the answers already extracted.

In [KNNV02, NKNV01, NNVK02] we mine and use coverage and overlap statistics of each source with respect to a precise query for deciding the best plan for extracting the answers. On similar lines, for every imprecise query and database pair we must store both the percentage of top-k answers returned as well as the total number of answers extracted. We must also estimate possible overlaps with other sources. Hence for every pair of query and database we will have to compute:

1. *B(Q,D)*: The best ranked tuple among the top-k returned by database D.
2. *W(Q,D)*: The worst ranked tuple among the top-k returned by database D.
3. *Percentage(k)*: The percentage of tuples returned by D that were in top-k.
4. $|Q(D)|$: Total number of answers for Q extracted from database D.

In [YPM03], two algorithms for merging results from ranked databases are provided. However, no overlap among databases is considered. Therefore extensions of the given algorithms to account for overlap among databases needs to be developed. The attribute importance estimation and semantic similarity estimation algorithms presented in this dissertation will be required even for answering imprecise queries over multiple sources.

REFERENCES

[ABC$^+$02] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. BANKS: Browsing and Keyword Searching in Relational Databases. *In proceedings of VLDB*, 2002.

[AC99] A. Abolunaga and S. Chaudhuri. Self-tunig Histograms: Building Histograms Without Looking at Data. *SIGMOD*, 1999.

[ACDG03] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated Ranking of Database Query Results. *CIDR*, 2003.

[ACPS96] S. Adali, K.S. Candan, Y. Papakonstantinou, and V.S. Subramanian. Query Caching and Optimization in Distributed Mediator Systems. *In proceedings of SIGMOD*, 1996.

[BGM02] N. Bruno, L. Gravano, and A. Marian. Evaluating Top-K Quries over Web-Accessible Databases. *In proceedings of ICDE*, 2002.

[Bib05] BibFinder: A Computer Science Bibliography Mediator. Available at: http://kilimanjaro.eas.asu.edu. 2005.

[BYRN99] R. Baeza-Yates and B. Ribiero-Neto. *Modern Information Retrieval.* Addison Wesley Longman Publishing, 1999.

[CCH92] J.P. Callan, W.B. Croft, and S.M. Harding. The INQUERY Retrieval System. *In Proceedings of the Database and Expert Systems Applications*, 1992.

[CCL91] W.W. Chu, Q. Chen, and R. Lee. Cooperative Query Answering via Type Abstraction Hierarchy. *Cooperative Knowledge Based Systems*, pages 271–290, 1991.

[CCL92] W.W. Chu, Q. Chen, and R. Lee. A Structured Approach for Cooperative Query Answering. *IEEE TKDE*, 1992.

[CD96] M.J. Carey and D.J. DeWitt. Of Objects and Databases: A Decade of Turmoil. *In proceedings of VLDB*, 1996.

[CGMH$^+$94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of Heterogeneous Information Sources. *In Proceedings of the 100th Anniversary Meeting, Information Processing Society of Japan, Tokyo, Japan*, pages 7–18, October 1994.

[CH] G. Cooper and E. Herskovits. A Bayesian method for Induction of Probabilistic Networks from Data. *Machine Learning, 9.*

[Cha00] W.G. Charles. Contextual Correlates of Meaning. *Applied Psycholinguistics*, 21:505–524, 2000.

[CHL$^+$04] K.C Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, Sep 2004.

[Coc77] W.G. Cochran. Sampling Techniques. *John Wiley & Sons, Third Edition*, 1977.

[Coh98] W. Cohen. Integration of Heterogeneous Databases without Common Domains Using Queries based on Textual Similarity. *In proceedings of SIGMOD*, pages 201–212, June 1998.

[Coo97] G. Cooper. A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. *Data Mining and Knowledge Discovery*, 2(1997).

[DMR98] G. Das, H. Mannila, and P. Ronkainen. Similarity of Attributes by External Probes. *In proceedings of KDD*, 1998.

[DR00] M. Dalkilic and E. Robertson. Information Dependencies. *In proceedings of PODS*, 2000.

[Eng05a] Altavista Search Engine. Available at http://www.altavista.com. 2005.

[Eng05b] Google Search Engine. Available at http://www.google.com. 2005.

[Eng05c] Yahoo Search Engine. Available at http://search.yahoo.com. 2005.

[FLN01] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *In proceedings of PODS*, 2001.

[Gas97] T. Gasterland. Cooperative Answering through Controlled Query Relaxation. *IEEE Expert*, 1997.

[GGR99] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS-Clustering Categorical Data Using Summaries. *In proceedings of KDD*, 1999.

[GKR98] D. Gibson, J. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. *In proceedings of VLDB*, 1998.

[Gri75] H. Grice. Logic and conversation. *Syntax and Semantics*, 1975.

[GRS99] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. *In proceedings of ICDE*, 1999.

[GSVGM98] R. Goldman, N .Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity Search in Databases. *In proceedings of VLDB*, 1998.

[HGKI02] T. Haveliwala, A. Gionis, D. Klein, and P Indyk. Evaluating Strategies for Similarity Search on the Web. *In proceedings of WWW, Hawai, USA*, May 2002.

[HK04] T. Hernandez and S. Kambhampati. Integration of Biological Sources: Current Systems and Challenges Ahead. *SIGMOD Record*, September 2004.

[HKPT98] Y. Huhtala, J. Krkkinen, P. Porkka, and H. Toivonen. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. *In proceedings of ICDE*, 1998.

[HKWY97] L.M. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. *In proceedings of VLDB*, 1997.

[HP02] V. Hristidis and Y. Papakonstantinou. Discover: Keyword Search in Relational Databases. *In proceedings of VLDB*, 2002.

[IAE03] I.F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting Top-k Join Queries in Relational Databases. *In proceedings of VLDB*, 2003.

[IMH$^+$04] I.F. Illyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic Discovery of Correlations and Soft Funcional Dependencies. *Sigmod*, 2004.

[JKW97] S. Jones, Karen, and P. Willet. *Readings in Information Retrieval.* Morgan Kaufmann, 1997.

[Jos82] A. Joshi. Mutual Beliefs in Question Answering Systems. *Mutual Knowledge edited by N.Smith*, 1982.

[JWS81] A. Joshi, B. Webber, and I. Sag. Elements of Discource Understanding. *Cambridge University Press*, 1981.

[Kim02] W. Kim. On Database Technology for US Homeland Security. *Journal of Object Technology*, 1(5):43–49, November-December 2002.

[Kin04] K. Kintigh. Enabling the study of long-term Human and Social Dynamics: A Cyberinfrastructure for Archeology. *NSF Human and Social Dynamics Program*, 2004.

[KLN$^+$04] S. Kambhampati, E. Lambrecht, U. Nambiar, Z. Nie, and G. Senthil. Optimizing Recursive Information Gathering Plans in EMERAC. *Journal of Intelligent Information Systems, Volume 22, Issue 2*, March 2004.

[KM95] J. Kivinen and H. Mannila. Approximate Dependency Inference from Relations. *Theoretical Computer Science*, 1995.

[KNNV02] S. Kambhampati, U. Nambiar, Z. Nie, and S. Vaddi. Havasu: A Multi-Objective, Adaptive Query Processing Framework for Web Data Integration. *ASU CSE TR-02-005*, April, 2002.

[KS05] Y. Kalfoglou and M. Schorlemmer. Ontology Mapping: The State of the Art. *Semantic Interoperability and Integration, Dagstuhl Seminar Proceedings*, 2005.

[Lau98] T.K. Lauder. Learning and Representing Verbal Meaning: The Latent Semantic Analysis theory. *Current Directions in Psychological Science*, 7(8):161–164, 1998.

[Lee87] T. Lee. An Iynformation-theoretic Analysis of relational databases-part I: Data Dependencies and Information Metric. *IEEE Transactions on Software Engineering SE-13*, October 1987.

[LKG99] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing Recursive Information Gathering Plans. *In proceedings of IJCAI*, 1999.

[Low03] The Lowell Database Research Self Assessment. June 2003.

[LRO96] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. *In proceedings of VLDB, Bombay, India*, 1996.

[MC91] G.A. Miller and W.G. Charles. Contextual Correlates of Semantic Similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.

[MH04] J. Morris and G. Hirst. Non-Classical Lexical Semantic Relations. *Workshop on Computational Lexical Semantics, Human Language Technology Conference, Boston*, May 2004.

[Mor90] J.M. Morrissey. Imprecise Information and Uncertainty in Information Systems. *ACM Transactions on Information Systems*, 8:159–180, April 1990.

[Mot86] A. Motro. Extending the Relational Model to Support Goal Queries. *In proceedings of Workshop on Expert Systems*, 1986.

[Mot90] A. Motro. Flex: A Tolerant and Cooperative User Interface to Database. *IEEE TKDE*, pages 231–245, 1990.

[Mot98] A. Motro. Vague: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, 1998.

[Mus04] I. Muslea. Machine Learning for Online Query Relaxation. *KDD*, 2004.

[NH97] N.F. Noy and C. Hafner. The State of the Art in Ontology Design: A Survey and Comparative Review. *AI Magazine*, 18(3):53–74, Fall 1997.

[Nie04] Z. Nie. Mining and Using Coverage and Overlap Statistics for Data Integration. *Ph.D Dissertation, Arizona State University*, 2004.

[NIS05] NIST. Trec data collections. available at http://trec.nist.gov/data.html. 2005.

[NK03] U. Nambiar and S. Kambhampati. Answering Imprecise Database Queries: A Novel Approach. *In proceedings of WIDM*, 2003.

[NK04a] U. Nambiar and S. Kambhampati. Mining Approximate Funcitonal Dependencies and Concept Similarities to Answer Imprecise Queries. *WebDB*, June 17-18, 2004.

[NK04b] U. Nambiar and S. Kambhampati. Providing Ranked Relevant Results for Web Database Queries. *In proceedings of WWW (Alternate Track Papers & Poster)*, May 17-22, 2004.

[NK05] U. Nambiar and S. Kambhampati. Answering Imprecise Queries over Web Databases. *VLDB Demonstration*, August, 2005.

[NKH03] Z. Nie, S. Kambhampati, and T. Hernandez. BibFinder/StatMiner: Effectively Mining and Using Coverage and Overlap Statistics in Data Integration. *VLDB Demonstration*, 2003.

[NKNV01] Z. Nie, S. Kambhampati, U. Nambiar, and S. Vaddi. Mining Source Coverage Statistics for Data Integration. *In proceedings of WIDM*, 2001.

[NNVK02] Z. Nie, U. Nambiar, S. Vaddi, and S. Kambhampati. Mining Coverage Statistics for Websource Selection in a Mediator. *In proceedings of CIKM, McLean, Virginia*, November 2002.

[OB03] Micheal Ortega-Binderberger. *Integrating Similarity Based Retrieval and Query Refinement in Databases.* PhD thesis, Ph.D Dissertation, UIUC, 2003.

[PHW82] M.E. Pollack, J. Hirschberg, and B. Webber. User Participation in the Reasonging Process of Expert Systems. *In proceedings of AAAI*, 1982.

[Por80] M.F. Porter. *An Algorithm for Suffix Stripping.* Program, 14(3):130-137, 1980.

[Res95] P. Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. *In proceedings of IJCAI*, 1995.

[RG65] H. Rubenstein and J.B. Goodenough. Contextual Correlates of Synonymy. *Computational Linguistics*, 8:627–633, 1965.

[Rij79] C.J. Van Rijsbergen. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

[RN03] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. *Prentice Hall*, Second Edition, 2003.

[SAYS02] S. Cho S. Amer-Yahia and D. Srivastava. Tree pattern relaxation. *EDBT*, 2002.

[SL90] A.P. Sheth and J.A. Larson. Federated Database Systems for managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SsBMU98] C. Silverstein, s. Brin, R. Motwani, and J. Ullman. Scalable Techniques for Mining Causal Structures. *VLDB*, 1998.

[TRV98] A. Tomasic, L. Raschid, and P. Valduriez. Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE TKDE*, 10(5), 1998.

[Voo99] E. Voorhees. The TREC-8 Question Answering Track Report. *TREC 8*, November 17-19, 1999.

[YPM03] C. Yu, G. Philip, and W. Meng. Distributed Top-N Query Processing with Possibly Uncooperative Local Systems. *In proceedings of VLDB*, 2003.

[ZZC04] B. He Z. Zhang and K.C.-C. Chang. Knocking the Door to the Deep

Web: Understanding Web Query Interfaces. *SIGMOD Demonstration*, June 2004.

APPENDIX A

OVERVIEW OF TANE - THE APPROXIMATE DEPENDENCY MINING ALGORITHM

TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies

The solution for supporting imprecise queries proposed in this dissertation requires computing the approximate functional dependencies between attributes of the database. These dependencies are used by AIMQ to learn the importance to be ascribed to each attribute, which in turn is used for effective query relaxation and query-tuple similarity estimation. While several researchers have proposed algorithms to mine the approximate functional depedencies, we found *TANE*, the algorithm proposed by Huhtala et al [HKPT98] as the most intuitive and efficient. Below we give a brief overview of the algorithm. For details about the various procedures of the algorithm, correctness of the proofs and performance results, please read [HKPT98].

A.1. Measuring Approximate Dependency

TANE defines the approximateness of a dependency $X \rightarrow A$ as the minimum number of tuples that need to be removed form the relation r for $X \rightarrow A$ to hold in r. The error $e(X \rightarrow A)$ is defined as

$$e(X \rightarrow A) = \frac{min\{|s| \mid s \subseteq r, X \rightarrow A \quad holds \quad in \quad r \backslash s\}}{|r|}$$

The measure e can be interpreted as the fraction of tuples with exceptions or errors affecting the dependency. Given an error threshold $\varepsilon, 0 \leq \varepsilon \leq 1$, $X \rightarrow A$ is an *approximate (functional) dependency* if and only if $e(X \rightarrow A)$ is at most ε.

TANE's approach for discovering dependencies is based on considering sets of tuples that agree on some sets of attributes. Determining whether a dependency holds or not is done by checking whether the tuples agree on the right-hand side of the dependency whenever they agree on the left-hand side. Formally, it can described using equivalence classes and partitions.

Partitions: Two tuples t and u are *equaivalent* with respect to a given set X of attributes if $t[A] = u[A]$ for all $A \in X$. Any such attribute set X can then partition the tuples into *equivalence classes*. If the equivalence class of $t \in r$ with respect to $X \subset R$ is $[t]_X$ then, the set of equivalence classes $\pi_X = \{[t]_X | t \in r\}$ is a *partition* of r under X. Thus, π_X is a collection of disjoint sets (equivalence classes) of tuples each of which has a unique value for the attribute set X and their union equals the relation r.

A simple test to determine if $X \to A$ holds is to check if $|\pi_X| = |\pi_{X \cup \{A\}}|$. Then we can compute the error $e(X \to A)$ as follows. Since any equivalence class of $c \subset \pi_X$ will be a union of one or more equivalence classes, $c'_1, c'_2 ...$ of $\pi_{X \cup \{A\}}$, tuples in all but one of the c'_is must be removed for $X \to A$ to hold. The minimum number of tuples to remove to make $X \to A$ hold in c is given by subtracting the size of the largest c'_is from size of c. Summing over all equivalence classes $c \in \pi_X$ gives the total number of tuples to remove. Thus

$$e(X \to A) = 1 - \sum_{c \in \pi_X} \frac{max\{|c'| | c' \in \pi_{X \cup \{A\}} \quad and \quad c' \subseteq c\}}{|r|}$$

A.2. Searching non-trivial dependencies:

To test the minimality of a potential dependency $X \backslash \{A\} \to A$, TANE must ensure that $Y \backslash \{A\} \to A$ holds for some proper subset Y of X. This information is stored by TANE in the set of right-hand candidates of Y, $C(Y)$. To find minimal dependencies, it suffices to test dependencies $X\ \{A\} \rightarrow A$, where $A \in X$ and $A \in C(X \backslash \{B\})$ for all $B \in X$. Here $C(X)$ is the collection of *initial rhs candidates* of the set $X \subseteq R$ and is defined as $C(X) = R \backslash \overline{C(X)}$ where $\overline{C(X)} = \{A \in X | X \backslash \{A\} \rightarrow A \quad holds\}$.

Eventhough the initial rhs candidates are sufficient to guarantee the minimality of discovered dependencies, TANE uses improved *rhs^{+} candidates* $C^{+}(X)$ to prune the search more effectively:

$$C^{+}(X) = \{A \in R | \forall B \in X, X \backslash \{A, B\} \rightarrow \{B\} \quad does \quad not \quad hold\}$$

TANE does not compute the partitions from scratch for each attribute set. Instead it computes a partition as a product of two previously computed partitions. The partitions $\pi_{\{A\}}$, for each $A \in R$ is computed directly from the database. However partitions π_X for $|X| \geq 2$ are computed as product of any two subsets of size $|X| - 1$. Once TANE has the partition π_X, it computes the error $e(X)$ as described above.

The worst case time complexity of TANE with respect to the number of attributes is exponential, but that is expected since the number of minimal dependencies can be exponential in the number of attributes. However, with respect to the number of tuples, the time complexity of TANE is linear (provided the set of dependencies do not change as the number of tuples increase). This linearity makes TANE especially suitable for relations with a large number of tuples but with small attributes sizes, e.g. most databases on the Web.

APPENDIX B

OVERVIEW OF THE ROCK CLUSTERING ALGORITHM

ROCK: A Robust Clustering Algorithm for Categorical Attribtues

In this dissertation we have compared the performance of AIMQ against that of ROCK, a domain-independent algorithm developed by *S. Guha, R. Rastogi and K. Shim* to cluster categorical attributes. Below we briefly describe the inner workings of the algorithm. For a more detailed description of the algorithm, please refer to [GRS99].

B.1. Links and Neighbours

The ROCK algorithm is based on the concept of clustering based on *links* between data points, instead of distances based on the L_p metric or the Jaccard coefficient. In contrast to clustering approaches that only use the similarity between the points while clustering, ROCK's link-based approach is claimed as being global in nature as it captures the global knowledge of neighbouring data points.

ROCK starts out by assuming that a pair of points are *neighbours* if their similarity exceeds a certain pre-defined threshold. The similarity between points can be based on L_p distances or Jaccard coefficient or any other non-metric function given by a domain expert. ROCK defines *link*(p_i, p_j) to be the number of common neighbours between p_i and p_j.

Authors argue that points belonging to a single cluster will in general have a large number of common neighbours and consequently more links. Thus during clustering, ROCK merges the clusters/points with the most number of links first. Specifically, ROCK aims to maximize the sum of $link(p_q, p_r)$ for all pairs of points p_q, p_r

belonging to a single cluster and at the same time, minimize the sum of $link(p_q, p_s)$ for p_q, p_s in different clusters. This results in a criterion function E_l:

$$E_l = \sum_{i=1}^{k} n_i \times \sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{1+2f(\theta)}}$$

where C_i denotes cluster i of size n_i. To prevent a clustering wherein all points are assigned to a single cluster, the criterion function E_l is defined as the weighted fraction of total number of links involving pairs of points in cluster C_i to the expected number of links in C_i. The weight is n_i, the number of points in C_i. Assuming $n_i^{f(\theta)}$ points in C_i, the expected number of links between points of C_i will be $n_i^{1+2f(\theta)}$. Dividing by the expected number of links in E_l prevents points with very few links between them from being put in the same cluster.

Handling of Categorical Data: ROCK considers data sets with categorical attributes by modeling each record as a transaction containing items. For every value v in the domain of a categorical attribute A, ROCK introduces an item $A.v$. A transaction T_i corresponding to a record in the database will contain $A.v$ if and only if the values of attribute A in the record is v. If A has missing values then the corresponding transaction will not contain any items for the attribute. Then similarity between records (points) is measured as the Jaccard similarity between the corresponding transactions.

B.2. The ROCK clustering Algorithm

The process of clustering data using ROCK involves three crucial steps: (1) ROCK begins by drawing a random sample from the database, (2) applies a hierarchical clustering algorithm that employs links on the sampled points and finally, (3) uses the clusters involving only the sampled points to assign the remaining data points to the appropriate clusters.

As described above, ROCK uses the criterion function to estimate the "goodness" of clusters and considers the best clustering between the points to be ones that result in highest values for the criteria function. On similar lines, authors define the goodness measure between pair of clusters C_i, C_j to be

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

where $link(C_i, C_j) = \sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r)$. The pair of clusters for which the above goodness measure is maximum is the best pair of clusters to be merged at any given step.

Given a set S of n sampled points that are randomly drawn from the original data set to be clustered and the number of desired clusters k, ROCK begins by computing the number of links between pairs of points. Initially each point is a separate cluster. For each cluster i, ROCK builds a local heap $q[i]$ that contains every cluster j for which $link[i, j]$ is non-zero. The clusters j in $q[i]$ are ordered in decreasing order of the goodness measure $g(i, j)$. In addition to the local heap, the algorithms also computes a global heap Q that contains all the clusters. Similar to the clusters in the local heaps, the clusters in Q are also ordered in decreasing order of their best goodness measure, $g(j, max(q[j]))$, where $max(q[j])$ denotes the best cluster in $q[j]$ to merge with cluster j.

ROCK iteratively clusters the data points by picking at each step,the cluster j

with maximum goodness in Q and the cluster in $q[j]$ that shows maximum goodness to j, as the best pairs of clusters to merge. The process continues till either k clusters only remain or when no two clusters have links between them.

In the final labeling phase, ROCK assigns the remaining data points to the clusters generated from the sample points as follows. First, a fraction of points from each cluster i, L_i is obtained. Then each point p in the original dataset is assigned to the cluster i in which p has the maximum number of neighbours. $(Ł_i| + 1)^{f(\theta)}$ is the expected number of neighbours for p in set L_i. Thus labeling each point p requires at most $\sum_{i=1}^{k} |L_i|$ operations.

Time complexity: ROCK's clustering algorithm, along with the computation of neighbour lists and links has a worst-case time complexity of $O(n^2+nm_mm_a+n^2logn)$ where m_m is the maximum number of neighbours, m_a is the average number of neighbours and n is the number of input data points. Note that ROCK only computes the clusters using a very small sample of the data set. In the worst case the link computation algorithm will have $O(n^3)$ complexity. However, the authors argue that on average the number of neighbours will be small causing the link matrix to be sparse. For such matrices they provide an algorithm whose complexity is $O(nm_mm_a)$ in general but becomes $O(n^2m_a)$ in the worst case.

The labeling phase of ROCK has the complexity $O(n_dkn_l)$ where n_d is the total number of points in the original dataset, k is the number of clusters derived from sample data points and n_l is the average number of neighbours in each cluster.

APPENDIX C

DESCRIPTION OF CARDB AND TEST QUERIES

We designed our first testbed, *CarDB*, to mimic the online used car database *Yahoo Autos*[1] CarDB was designed using a MySQL database and projects the relation

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

To populate CarDB, we probed the *Yahoo Autos* database by generating probe queries that bound the attributes *Make* and *Location* and extracted $100,000$ distinct tuples. A 10 second delay between each probe query was maintained. The values for Make were provided by Yahoo Autos. The potential binding values for Location consisted of all cities of US.

Attribute	**Type**	**Distinct Values (100k)**	**Distinct Values (25k)**
Make	Categorical	90	63
Model	Categorical	1152	747
Year	Categorical	55	45
Price	Numerical	NA	NA
Mileage	Numerical	NA	NA
Location	Categorical	1319	1082
Color	Categorical	2427	1085

Table 11. Schema Description of CarDB

Below is a listing of the approximate functional dependencies and approximate keys with their percentage support that we mined from the $25k$ sample of CarDB.

Approximate Functional Dependencies mined from CarDB:

Color $->$ Model : 0.12

Location $->$ Model : 0.13

Location $->$ Color : 0.20

[1]Available at http://autos.yahoo.com.

Price − > Model: 0.21

Model − > Color : 0.22

Price − > Location : 0.23

Price − > Color : 0.24

Color − > Year : 0.31 Location − > Year : 0.33

Price − > Make : 0.34

Color − > Make : 0.35

Location − > Make : 0.36

Model − > Year : 0.39

Price − > Year : 0.4

Model − > Make : 0.92

Make, Color − > Year : 0.41

Make, Location − > Model : 0.42

Year, Location − > Make : 0.43

Make, Location − > Year : 0.49

Location, Color − > Year : 0.54

Model, Color − > Year : 0.55

Location, Color − > Make : 0.6

Make, Price − > Model : 0.62

Make, Price − > Year : 0.65

Model, Location − > Year : 0.71

Model, Price − > Year : 0.82

Year, Price, Color $->$ Make : 0.81

Make, Price, Location $->$ Year : 0.91

Approximate Keys mined from CarDB:

Price : 0.2

Mileage : 0.8

Model, Color : 0.33

Location, Color : 0.43

Make, Price : 0.52

Price, Color : 0.61

Model, Location : 0.65

Model, Price : 0.73

Price, Location : 0.75

Year, Mileage : 0.91

Make, Mileage : 0.97

Mileage, Location : 0.97

Mileage, Color : 0.98

Price, Mileage : 0.98

Model, Mileage : 0.99

Make, Year, Location : 0.57

Make, Year, Price : 0.7

Year, Location, Color : 0.72

Make, Location, Color : 0.72

Year, Price, Color : 0.79

Model, Year, Location : 0.83

Make, Price, Color : 0.83

Year, Price, Location : 0.89

Make, Price, Location : 0.9

Model, Location, Color : 0.91

Model, Price, Color : 0.93

Model, Price, Location : 0.95

Price, Location, Color : 0.96

Make, Year, Location, Color : 0.88

Make	**Model**	**Year**	**Price**	**Mileage**	**Location**	**Color**
Ford	Contour	1996	3252	86120	Tucson	Red
Jeep	Cherokee	1999	10232	94264	Tucson	Gray
Cadillac	DeVille	1999	20646	47095	Port Richey	Silver
Mercury	Mountaineer	1999	8986	80801	Glendale Heights	White
BMW	7 SERIES	1995	13995	48543	Devon	Black
GMC	Jimmy	1989	1995	191687	Langhorne	Black
Nissan	Altima	1993	5248	126152	Warminster	Burgundy
Ford	Thunderbird	1995	4290	72790	Orland Park	Pewter
Mitsubishi	Galant	1993	2481	137958	Miami	Beige
Toyota	Tercel	1987	3491	130265	Miami	Blue
Nissan	300ZX	1986	7900	65611	Coconut Creek	Red
Ford	Bronco II	1986	4490	140479	Longwood	Red
Chevrolet	El Camino	1966	7500	100000	Sutter Creek	Red
Mercedes-Benz	SL Class	1988	17950	73906	Anaheim	Red
Volkswagen	Quantum	1984	1995	118579	San Leandro	Brown
Mercedes	c280	1994	6999	108455	Fremont	Pewter

Table 12. Queries over CarDB used to conduct user study

BIOGRAPHICAL SKETCH

Ullas Nambiar received his Bachelors Degree in Computer Science from M.S. University of Baroda, Gujarat in 1997. After a brief stint at hacking, he joined Arizona State University to pursue his Ph.D. in Computer Science.

His research interests are in enabling seamless access to heterogeneous information present in the autonomous data sources accessible on the Internet. This involves research in diverse areas such as data integration, imprecise query answering, source/query metadata mining, and developing domain-independent techniques for learning semantic data models.

He will be joining University of California, Davis as a Research Scholar from September 2005.