Query Processing over Incomplete Autonomous Databases

Garrett Wolf Hemal Khatri Bhaumik Chokshi Yi Chen Subbarao Kambhampati
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287, USA
{garrett.wolf, hemal.khatri, bmchoksh, yi, rao}@asu.edu

ABSTRACT

Incompleteness due to missing attribute values (aka "null values") is very common in autonomous web databases, on which user accesses are usually supported through mediators. Traditional query processing techniques that focus on the strict soundness of answer tuples often ignore tuples with critical missing attributes, even if they wind up being relevant to a user query. Ideally we would like the mediator to retrieve such possible answers and gauge their relevance by accessing their likelihood of being pertinent answers to the query. The autonomous nature of web databases poses several challenges in realizing this objective. Such challenges include the restricted access privileges imposed on the data, the limited support for query patterns, and the bounded pool of database and network resources in the web environment. We introduce a novel query rewriting and optimization framework QPIAD that tackles these challenges. Our technique involves reformulating the user query based on mined correlations among the database attributes. The reformulated queries are aimed at retrieving the relevant possible answers in addition to the certain answers. QPIAD is able to gauge the relevance of such queries allowing tradeoffs in reducing the costs of database query processing and answer transmission. To support this framework, we develop methods for mining *attribute correlations* (in terms of Approximate Functional Dependencies), *value distributions* (in the form of Naïve Bayes Classifiers), and *selectivity estimates*. We present empirical studies to demonstrate that our approach is able to effectively retrieve relevant possible answers with high precision, high recall, and manageable cost.

1. INTRODUCTION

Data integration in autonomous web database scenarios has drawn much attention in recent years, as more and more data becomes accessible via web servers which are supported by back-end databases. A mediator provides a unified query interface as a global schema of the underlying databases. Queries on the global schema are then rewritten as queries over autonomous databases through their web interfaces. Current mediator systems [21, 16] only return to the user *certain answers* that exactly satisfy all the user query predicates. For example, in a used car trading application, if a user is interested in cars made by *Honda*, all the returned answers will have the value "Honda" for attribute *Make*. Thus, an *Accord* which has a *missing* value for *Make* will not be returned by such systems. Unfortunately, such an approach is both inflexible and inadequate for querying autonomous web databases which are inherently incomplete. As an example, Table 1 shows statistics on the percentage of incomplete tuples from two autonomous web databases. The statistics were computed from a randomly probed sample. The table also gives statistics on the percentage of missing values for the *Body Style* and *Engine* attributes.

Website	# of Attributes	Total Tuples	Incomplete Tuples %	Body Style %	Engine %
www.AutoTrader.com	13	25127	33.67%	3.6%	8.1%
www.CarsDirect.com	14	32564	98.74%	55.7%	55.8%

Table 1: Statistics on missing values in web databases

Such incompleteness in autonomous databases should not be surprising as it can arise for a variety of reasons, including:

Incomplete Entry: Web databases are often populated by lay individuals without any central curation. For example, web sites such as *Cars.com*[1] and *Yahoo! Autos*,[2] obtain information from individual car owners who may not fully specify complete information about their cars, thus leaving such databases scattered with missing values (aka "null" values). Consider a car owner who leaves the *Make* attribute blank, assuming that it is obvious as the *Model* of the car she is selling is *Accord*. This type of incompleteness is expected to increase even more with services such as GoogleBase which provide users significant freedom in deciding which attributes to define and/or list [23].

Inaccurate Extraction: Many web databases are being populated using automated information extraction techniques. As a result of the inherent imperfection of these extractions, many web databases may contain missing values. Examples of this include imperfections in web page segmentation (as described in [11]) or imperfections in scanning and converting handwritten forms (as described in [2]).

Heterogenous Schemas: Global schemas provided by mediator systems may often contain attributes that do not appear in all of the local schemas. For example, a global schema for the used car trading domain has an attribute called *Body Style*, which is supported by *Cars.com*, but not by *Yahoo! Autos*. Given a query on

[1]http://www.cars.com
[2]http://autos.yahoo.com

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '07, September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

the global schema for cars having *Body Style* equal to *Coupe*, mediators which only return the certain answers are not able to make use of information from the *Yahoo! Autos* database thereby failing to return a possibly large portion of the relevant tuples.[3]

Although there has been work on handling incompleteness in databases (see Section 2), much of it has been focused on single databases on which the query processor has complete control. The approaches developed–such as the "imputation methods" that attempt to modify the database directly by replacing null values with likely values–are not applicable for autonomous databases where the mediator often has restricted access to the data sources. Consequently, when faced incomplete databases, current mediators only provide the certain answers thereby sacrificing recall. This is particularly problematic when the data sources have a significant fraction of incomplete tuples, and/or the user requires high recall (consider, for example, a law-enforcement scenario, where a potentially relevant criminal is not identified due to fortuitous missing information).

A naïve approach for improving the recall in these systems would be to return, in addition to all the certain answers, all the tuples with missing values on the constrained attribute(s) as *possible* answers to the query. For example, given a selection query for cars made by "Honda", a mediator could return not only those tuples whose *Make* values are "Honda" but also the ones whose *Make* values are missing(null). This approach, referred to as ALLRETURNED, has an obvious drawback, in that many of the tuples with missing values on constrained attributes are *irrelevant* to the query. Intuitively, not every tuple that has a missing value for *Make* corresponds to a car made by *Honda*! Thus, while improving recall, the ALLRETURNED approach can lead to drastically lower precision.

In an attempt to improve precision, a more plausible solution could start by first retrieving all the tuples with *null* values on the constrained attributes, predicting their missing values, and then deciding the set of relevant query answers to show to the user. This approach, that we will call ALLRANKED, has better precision than ALLRETURNED. However, most of the web-accessible database interfaces we've found, such as *Yahoo Autos*, *Cars.com*, *Realtor.com*, etc, *do not allow the mediator to directly retrieve tuples with null values on specific attributes*. In other words, we cannot issue queries like "list all the cars that have a missing value for *Body Style* attribute". Even if the sources do support binding of *null* values, retrieving and additionally ranking all the tuples with missing values involves high processing and transmission costs.

Our Approach: In this paper, we present QPIAD,[4] a system for mediating over incomplete autonomous databases. To make the retrieval of possible answers feasible, QPIAD bypasses the null value binding restriction by generating *rewritten* queries according to a set of mined attribute correlation rules. These rewritten queries are designed such that there are no query predicates on attributes for which we would like to retrieve missing values. Thus, QPIAD is able to retrieve possible answers without binding null values or modifying underlying autonomous databases. To achieve high precision and recall, QPIAD learns Approximate Functional Dependencies (AFDs) for attribute correlations, Naïve Bayesian Classifiers (NBC) for value distributions, and query selectivity estimates from a database sample obtained off-line. These data source statistics are then used to provide a ranking scheme to gauge the relevance of a possible answer to the original user query. Furthermore, instead of ranking possible answers directly, QPIAD orders the rewritten queries in the order of the number of relevant answers they are expected to bring as determined by the attribute value distributions and selectivity estimations. The rewritten queries are issued in this order, and the returned tuples are ranked in accordance with the query that retrieved them. By ordering the rewritten queries rather than ranking the entire set of possible answers, QPIAD is able to optimize both precision and recall while maintaining efficiency. We extend this general approach to handle selection, aggregation and join queries, as well as supporting multiple correlated sources.

Contributions: QPIAD's query rewriting and ranking strategies allow it to efficiently retrieve relevant possible answers from autonomous databases given mediator's query-only capabilities and limited query access patterns to these databases. To the best of our knowledge, the QPIAD framework is the first that retrieves relevant possible answers with missing values on constrained attributes without modifying underlying databases. Consequently, it is suitable for querying incomplete autonomous databases. The idea of using learned attribute correlations, value distributions, and query selectivity to rewrite and rank queries, which consider the natural tension between precision and recall, is also a novel contribution of our work. In addition, our framework can leverage attribute correlations among data sources in order to retrieve relevant possible answers from data sources not supporting the query attribute (e.g. local schemas which do not support the entire set of global schema attributes). Our experimental evaluation over selection, aggregation, an join queries shows that QPIAD retrieves most relevant possible answers while maintaining low query processing costs.

Assumptions: In this paper we assume that tuples containing more than one null over the set of query constrained attributes are less relevant to the user. Therefore, such tuples are not ranked but simply output after the ranked tuples containing zero or a single null over the set of query constrained attributes.

Organization: The rest of the paper is organized as follows. In the next section we discuss related work on handling incomplete data. Next, we cover some preliminaries and an overview of our framework in Section 3. Section 4 proposes online query rewriting and ranking techniques to retrieve relevant possible answers from incomplete autonomous databases in the context of selection, aggregation, and join queries, as well as retrieving possible answers from data sources which do not support the query attribute in their local schemas. Section 5 provides the details of learning attribute correlations, value distributions, and query selectivity used in our query rewriting phase. A comprehensive empirical evaluation of our approach is presented in Section 6. We conclude the paper in Section 7.

2. RELATED WORK

Querying incomplete databases: Traditionally, incompleteness in databases has been handled by one of two broad approaches. The first–which we call *possible world approaches* [15, 22, 2]–tracks the completions of all incomplete tuples. All feasible completions are considered equally likely and the aim of query processing is to return certain vs. possible answers without making any distinctions among the possible answers. To help track all possible worlds, *null* values are typically represented using one of three different meth-

[3]Moreover, an attribute may not appear in a schema intensionally as the database manager may suppress the values of certain attributes. For example, the travel reservation website *Priceline.com* suppresses the airline/hotel name when booking tickets and hotel.

[4]QPIAD is an acronym for Query Processing over Incomplete Autonomous Databases. A 3-page poster description of QPIAD first appeared in [19].

ods, each of increasing generality: (i) Codd Tables where all the *null* values are treated equally; (ii) V-tables which allow many different *null* values marked by variables; and (iii) Conditional tables which are V-tables with additional attributes for conditions.

The second type of approaches for handling incomplete databases–which we call *probabilistic approaches* ([6, 3, 35])–attempt to quantify the distribution over the completions of an incomplete tuple, and use this information to distinguish between the likelihood of various possible answers. Our work falls in this second category. The critical novelty of our work is that our approach learns the distribution automatically, and also avoids modifying the original database in any way. It is therefore suitable for querying incomplete autonomous databases, where a mediator is not able to store the estimation of missing values in sources. [6] handles incompleteness for aggregate queries in the context of OLAP databases, by relaxing the original queries using the hierarchical OLAP structure. Whereas our work learns attribute correlations, value distributions and query selectivity estimates to generate and rank rewritten queries.

Querying inconsistent databases: Work on handling inconsistent databases also has some connections. While most approaches for handling inconsistent databases are more similar to the "possible worlds approaches" used for handling incompleteness (e.g. [4]), some recent work (e.g. [1]) uses probabilistic approaches for handling inconsistent data.

Probabilistic Databases: Incomplete databases are similar to probabilistic databases (c.f. [33, 7, 31, 34]) once the probabilities for missing values are assessed. [33] gives an overview of querying probabilistic databases where each tuple is associated with an additional attribute describing the probability of its existence. Some recent work on the TRIO [31, 34] system deals with handling uncertainty over probabilistic relational databases. In such systems, the notion of incompleteness is closely related to uncertainty: an incomplete tuple can be seen as a disjunction of its possible completions. However, we go a step further and view the incomplete tuple as a *probability distribution* over its completions. The distribution can be interpreted as giving a quantitative estimate of the probability that the incomplete tuple corresponds to a specific completion in the real world. Furthermore we address the problem of retrieving incomplete tuples from autonomous databases where the mediator does not have capabilities to modify the underlining databases.

Query Relaxation: Reformulating queries using database constraints for query optimization in distributed mediator systems has been described in [28]. There has been work on query relaxation over databases [25] which focuses on how to relax input query constraints such that the data which *partially* satisfies the query constraints is also returned. Our work is similar to such efforts in the spirit of retrieving relevant data even when it does not exactly satisfy user queries. However, we focus on retrieving data that has missing values in the query constrained attributes, yet is likely to be relevant to the original user query. Furthermore, our query relaxation for retrieving possible answers is value directed.

Ranked Joins: The part of our query processing framework which handles join queries over autonomous sources is similar to work on *ranked joins*[14]. However, we use predicted values learnt from the data itself to perform joins without modifying the underlying databases.

Learning missing values: There has been a large body of work on missing values imputation [8, 29, 30, 35, 3]. Common imputation approaches include substituting missing data values by the mean, the most common value, default value of the attribute in question, or using k-Nearest Neighbor [3], association rules [35], etc. Another approach used to estimate missing values is *parameter estimation*. Maximum likelihood procedures that use variants of the Expectation-Maximization algorithm [8, 29] can be used to estimate the parameters of a model defined for the complete data. In this paper, we are interested not in the standard imputation problem but a variant that can be used in the context of query rewriting. In this context, it is important to have schema level dependencies between attributes as well as distribution information over missing values. We use AFDs for the former, and an AFD-enhanced Naïve Bayes Classifiers for the later. We experimented with other methods including association rules and bayes network learning - but found them to be either significantly less accurate or significantly costlier to compute.

3. PRELIMINARIES AND ARCHITECTURE OF QPIAD

We will start with formal definitions of certain answers and possible answers with respect to selection queries.

DEFINITION 1 (COMPLETE/INCOMPLETE TUPLES). *Let $R(A_1, A_2, \cdots, A_n)$ be a database relation. A tuple $t \in R$ is said to be complete if it has non-null values for each of the attributes A_i; otherwise it is considered incomplete. A complete tuple t is considered to belong to the set of completions of an incomplete tuple $\widehat{t}$ (denoted $\mathcal{C}(\widehat{t})$), if t and $\widehat{t}$ agree on all the non-null attribute values.*

Now consider a selection query Q: $\sigma_{A_m=v_m}$ over relation $R(A_1, \cdots, A_n)$ where $(1 \leq m \leq n)$.

DEFINITION 2 (CERTAIN/POSSIBLE ANSWERS). *A tuple t_i is said to be a certain answer for the query Q: $\sigma_{A_m=v_m}$ if $t_i.A_m{=}v_m$. t_i is said to be an possible answer for Q if $t_i.A_m{=}null$, where $t_i.A_m$ is the value of attribute A_m in t_i.*

Notice an incomplete tuple is a certain answer to a query, if its null values are not on the attributes constrained in the query.

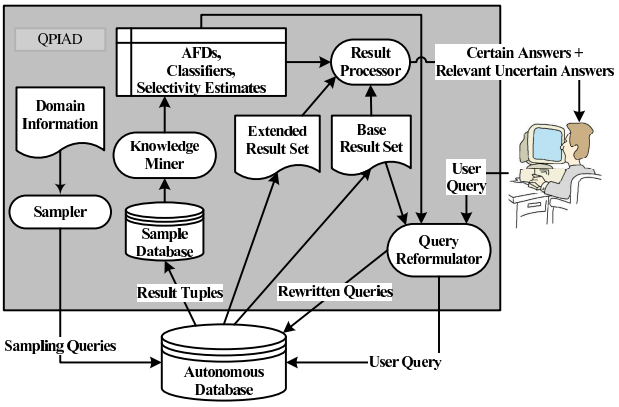


Figure 1: QPIAD System Architecture.

There are several key functionalities that QPIAD needs to perform in order to retrieve and rank possible answers to a user query: (i) *learning attribute correlations* to generate rewritten queries, (ii) *assessing the value probability distributions* of incomplete tuples

ID	Make	Model	Year	Body Style
1	Audi	A4	2001	Convt
2	BMW	Z4	2002	Convt
3	Porsche	Boxster	2005	Convt
4	BMW	Z4	2003	null
5	Honda	Civic	2004	null
6	Toyota	Camry	2002	Sedan

Table 2: Fragment of a Car Database

to provide a ranking scheme for possible answers, (iii) *estimating query selectivity* to estimate the recall and determine how many rewritten queries to issue, and based on the above (iv) *ordering rewritten queries* to retrieve possible tuples that have a high degree of relevance to the query.

The system architecture of the QPIAD system is presented in Figure 1. A user accesses autonomous databases by issuing a query to the mediator. The query reformulator first directs the query to the autonomous databases and retrieves the set of all certain answers (called the *base result set*). In order to retrieve highly relevant possible answers in ranked order, the mediator dynamically generates rewritten queries based on the original query, the base result set, and attribute correlations in terms of Approximate Functional Dependencies(AFDs) learned from a database sample. The goal of these new queries is to return an *extended result set*, which consists of highly relevant possible answers to the original query. Since these rewritten queries are not all equally good in terms of retrieving relevant possible answers, they are ordered before being posed to the databases. The ordering of the rewritten queries is based on their expected *F-Measure* which considers the estimated selectivity and the value distributions for the missing attributes.

QPIAD mines attribute correlations, value distributions, and query selectivity using a small portion of data sampled from the autonomous database using random probing queries. The knowledge mining module learns AFDs and AFD-enhanced Naïve Bayesian Classifiers (where the AFDs play a feature selection role for the classification task) from the samples. Then the knowledge mining module estimates the selectivity of rewritten queries. Armed with the AFDs, the corresponding classifiers, and the selectivity estimates, the query reformulator is able to retrieve the relevant possible answers from autonomous databases by rewriting the original user query and then ordering the set of rewritten queries such that the possible answers are retrieved in the order of their ranking in precision.

4. RETRIEVING RELEVANT POSSIBLE ANSWERS

In this section, we describe the QPIAD query rewriting approach for effectively and efficiently retrieving relevant possible answers from incomplete autonomous databases. We support queries involving selections, aggregations and joins. This query rewriting framework can also retrieve relevant answers from data sources not supporting the entire set of query constrained attributes.

4.1 Handling Selection Queries

To efficiently retrieve possible answers in their order of precision, QPIAD follows a two-step approach. First, the original query is sent to the database to retrieve the certain answers which are then returned to the user. Next, a group of rewritten queries are intelligently generated, ordered, and sent to the database. This process is done such that the query patterns are likely to be supported by the web databases, and only the most relevant possible answers are retrieved by the mediator in the first place.

Generating Rewritten Queries: The goal of the query rewriting is to generate a set of rewritten queries to retrieve relevant possible answers. Let's consider the same user query Q asking for all convertible cars. We use the fragment of the Car database shown in Table 2 to explain our approach. First, we issue the query Q to the autonomous database to retrieve all the certain answers which correspond to tuples t_1, t_2 and t_3 from Table 2. These certain answers form the *base result set* of Q. Consider the first tuple t_1=$\langle Audi, A4, 2001, Convt\rangle$ in the base result set. If there is a tuple t_i in the database with the same value for $Model$ as t_1 but missing value for $Body\ Style$, then $t_i.Body\ Style$ is likely to be $Convt$. We *capture this intuition by mining attribute correlations* from the data itself.

One obvious type of attribute correlation is "*functional dependencies*". For example, the functional dependency $Model \rightarrow Make$ often holds in automobile data records. There are two problems in adopting the method directly based on functional dependencies: (i) often there are not enough functional dependencies in the data, and (ii) autonomous databases are unlikely to advertise the functional dependencies. The answer to both these problems involves *learning* approximate functional dependencies from a (probed) sample of the database.

DEFINITION 3 (APPROXIMATE FUNCTIONAL DEPENDENCY). *$X \rightsquigarrow A$ over relation R is an* approximate functional dependency(AFD) *if it holds on all but a small fraction of the tuples. The set of attributes X is called the* determining set *of A denoted by dtrSet(A).*

For example, an AFD $Model \rightsquigarrow Body\ Style$ may be mined, which indicates that the value of a car's $Model$ attribute *sometimes* (but not always) determines the value of its $Body\ Style$ attribute. According to this AFD and tuple t_1, we issue a rewritten query $Q'_1 : \sigma_{Model=A4}$ with constraints on the *determining set* of the attribute $Body\ Style$, to retrieve tuples that have the same $Model$ as t_1 and therefore are likely to be $Convt$ in $Body\ Style$. Similarly, we issue queries $Q'_2 : \sigma_{Model=Z4}$ and $Q'_3 : \sigma_{Model=Boxster}$ to retrieve other relevant possible answers.

Ordering Rewritten Queries: In the query rewriting step of QPIAD, we generate new queries according to the distinct value combinations among the base set's determining attributes for each of the constrained attributes. In the example above, we used the three certain answers to the user query Q to generate three new queries: $Q'_1 : \sigma_{Model=A4}$, $Q'_2 : \sigma_{Model=Z4}$ and $Q'_3 : \sigma_{Model=Boxster}$. Although each of these three queries retrieve possible answers that are likely to be more relevant to Q than a random tuple with missing value for $Body\ Style$, they may not all be equally good in terms of retrieving relevant possible answers.

Thus, an important issue in query rewriting is the order in which to pose the rewritten queries to the database. This ordering depends on two orthogonal measures: the *expected precision* of the query– which is equal to the probability that the tuples returned by it are answers to the original query, and the *selectivity* of the query–which is equal to the number of tuples that the query is likely to bring in. As we shall show in Section 5, both the precision and selectivity can be estimated by mining a probed sample of the database.

For example, based on the value distributions in the sample database, we may find that a $Z4$ model car is more likely to be a *Convertible* than a car whose model is $A4$. As we discuss

in Section 5.2, we build AFD-enhanced classifiers which give the probability values $P(Body\ Style{=}Convt|Model{=}A4)$, $P(Body\ Style{=}Convt|Model{=}Z4)$ and $P(Body\ Style{=}Convt|Model{=}Boxster)$. Similarly, the selectivity of these queries can be different. For example, we may find that the number of tuples having $Model{=}A4$ is much larger than that of $Model{=}Z4$.

Given that we can estimate precision and selectivity of the queries, the only remaining issue is how to use them to order the queries. If we are allowed to send as many rewritten queries as we would like, then ranking of the queries can be done just in terms of the expected precision of the query. However, things become more complex if there are limits on the number of queries we can pose to the autonomous source. Such limits may be imposed by the network/processing resources of the autonomous data source or possibly the time that a user is willing to wait for answers.

Given the maximum number of queries that we can issue to a database, we have to find a reasonable tradeoff between the precision and selectivity of the queries issued. Clearly, all else being equal, we will prefer high precision queries to low precision ones and high selectivity queries to low selectivity ones. The tricky issue is how to order a query with high selectivity and low precision in comparison to another with low selectivity and high precision. Since the tension here is similar to the precision vs. recall tension in IR, we decided to use the well known *F-Measure* metric for query ordering. In the IR literature, *F-Measure* is defined as the weighted harmonic mean of the precision (P) and recall (R) measures: $\frac{(1+\alpha)*P*R}{\alpha*P+R}$. We use the query precision for P. We estimate the recall measure R of the query by first computing query throughput, i.e., expected number of relevant answers returned by the query (which is given by the product of the precision and selectivity measures), and then normalizing it with respect to the expected cumulative throughput of all the rewritten queries. Notice that the *F-Measure* based ordering reduces to precision-based ordering when $\alpha = 0$.

In summary, we use the F-measure ordering to select k top queries, where k is the number of rewritten queries we are allowed to issue to the database. Once the k queries are chosen, they are posed in the order of their expected precision. This way the relevant possible answers retrieved by these rewritten queries need not be ranked again, as their rank – the probability that their null value corresponds to the selected attribute– is the same as the precision of the retrieving query.

Note that the parameter α in the *F-measure*, as well as the parameter k (corresponding to the number of queries to be issued to the sources), can be chosen according to source query restrictions, source response times, network/database resource limitations, and user preferences. The unique feature of QPIAD is its flexibility to generate rewritten queries accordingly to satisfy the diverse requirements. It allows the tradeoff between precision and recall to be tuned by adjusting the α parameter in its *F-Measure* based ordering. When α is set to be 0, the rewritten queries are ordered solely in terms of precision. When α is set to be 1, the precision and recall are equally weighted. The limitations on the database and network resources are taken into account by varying k–the number of rewritten queries posed to the database.

4.2 Query Rewriting Algorithm

In this section, we describe the algorithmic details of the QPIAD approach. Let $R(A_1, A_2, \cdots, A_n)$ be a database relation. Suppose $dtrSet(A_m)$ is the determining set of attribute A_m $(1 \leq m \leq n)$, according to the highest confidence AFD (to be discussed in Section 5.3). QPIAD processes a given selection query $Q{:}\ \sigma_{A_m=v_m}$ according to the following two steps.

1. Send Q to the database and retrieve the base result set $RS(Q)$ as the certain answers of Q. Return $RS(Q)$ to the user.
2. Generate a set of new queries Q', order them, and send the most relevant ones to the database to retrieve the extended result set $\widehat{RS}(Q)$ as relevant possible answers of Q. This step contains the following tasks.
   (a) *Generate rewritten queries.* Let $\pi_{dtrSet(A_m)}(RS(Q))$ be the projection of $RS(Q)$ onto $dtrSet(A_m)$. For each distinct tuple t_i in $\pi_{dtrSet(A_m)}(RS(Q))$, create a selection query Q'_i in the following way. For each attribute A_x in $dtrSet(A_m)$, create a selection predicate $A_x{=}t_i.v_x$. The selection predicates of Q'_i consist of the conjunction of all these predicates.
   (b) *Select rewritten queries.* For each rewritten query Q'_i, compute the estimated precision and estimated recall using its estimated selectivity derived from the sample. Then order all Q'_is in order of their *F-Measure* scores and choose the top-K to issue to the database.
   (c) *Re-order selected top-K rewritten queries.* Re-order the selected top-K set of rewritten queries according to their estimated precision which is simply the conditional probability of $P_{Q'_i}{=}P(A_m{=}v_m|t_i)$. [5]
   (d) *Retrieve extended result set.* Given the top-K queries $\{Q'_1, Q'_2, \cdots, Q'_K\}$ issue them in the according to their estimated precision-base orderings. Their result sets $RS(Q'_1), RS(Q'_2), \cdots, RS(Q'_K)$ compose the extended result set $\widehat{RS}(Q)$.[6]
   (e) *Post-filtering.* If the database does not allow binding of null values, remove from $\widehat{RS}(Q)$ the tuples with $A_m \neq null$. Return the remaining tuples in $\widehat{RS}(Q)$ as the relevant possible answers of Q.

Multi-attribute Selection Queries: Although we described the above algorithm in the context of single attribute selection queries, it can also be used for rewriting multi-attribute selection queries by making a simple modification to Step 2(a). Consider a multi-attribute selection query $Q : \sigma_{A_1=v_1 \wedge A_2=v_2 \wedge \cdots \wedge A_c=v_c}$. To generate the set of rewritten queries Q', the modification requires Step 2(a) to run c times, once for each constrained attribute $A_i, 1 \leq i \leq c$. In each iteration, the tuples from $\pi_{dtrSet(A_i)}(RS(Q))$ are used to generate a set of rewritten queries by replacing the attribute A_i with selection predicates of the form $A_x{=}t_i.v_x$ for each attribute $A_x \in dtrSet(A_i)$. For each attribute $A_x \in dtrSet(A_m)$ which is not constrained in the original query, we add the constraints on A_x to the rewritten query. As we have discussed in Section 1, we only rank the tuples that contain zero or one *null* in the query constrained attributes. If the user would like to retrieve tuples with more than one null, we output them at the end without ranking.

For example, consider the multi-attribute selection query $Q{:}\ \sigma_{Model=Accord \wedge Price\ between\ 15000\ and\ 20000}$ and the mined AFDs $\{Make, Body\ Style\} \rightsquigarrow Model$ and $\{Year, Model\} \rightsquigarrow Price$. The algorithm first generates a set of rewritten queries by replacing the attribute constraint $Model{=}Accord$ with selection predicates for each attribute in the determining set of $Model$ using the attribute values from the tuples in the base set $\pi_{dtrSet(Model)}(RS(Q))$. After the first iteration, the algorithm may have generated the following queries:

[5] By re-ordering the top-K queries in order of their precision we ensure that each of the returned tuples will have the same rank as the query that retrieved it.

[6] All results returned for a single query are ranked equally.

Q'_1: $\sigma_{Make=Honda \wedge Body\ Style=Sedan \wedge Price\ between\ 15000\ and\ 20000}$,
Q'_2: $\sigma_{Make=Honda \wedge Body\ Style=Coupe \wedge Price\ between\ 15000\ and\ 20000}$

Similarly, the algorithm generates additional rewritten queries by replacing $Price$ with value combination of its determining set from the base set while keeping the original query constraint $Model$=$Accord$. After this second iteration, the following rewritten queries may have been generated:

$$Q'_3: \sigma_{Model=Accord \wedge Year=2002},$$
$$Q'_4: \sigma_{Model=Accord \wedge Year=2001},$$
$$Q'_5: \sigma_{Model=Accord \wedge Year=2003}$$

After generating a set of rewritten queries for each constrained attribute, the sets are combined and the queries are ordered just as they were in Step 2(b). The remainder of the algorithm requires no modification to support multi-attribute selection queries.

Base Set vs. Sample: When generating rewritten queries, one may consider simply rewriting the original query using the sample as opposed to first retrieving the base set and then rewriting. However, since the sample may not contain all answers to the original query, such an approach may not be able to generate all rewritten queries. By utilizing the base set, QPIAD obtains the entire set of determining set values that the source can offer, and therefore achieves a better recall.

4.3 Retrieving Relevant Answers from Data Sources Not Supporting the Query Attributes

In information integration, the global schema exported by a mediator often contains attributes that are not supported in some of the individual data sources. We adapt the query rewriting techniques discussed above to retrieve relevant possible answers from a data source not supporting the constrained attribute in the query. For example, consider a global schema $GS_{UsedCars}$ supported by the mediator over the sources *Yahoo! Autos* and *Cars.com* as shown in Figure 2, where *Yahoo! Autos* doesn't support queries on $Body\ Style$ attribute. Now consider a query Q: $\sigma_{Body\ Style=Convt}$ on the global schema. The mediator that only returns certain answers won't be able to query the *Yahoo! Autos* database to retrieve cars with $Body\ Style\ Convt$. None of the relevant cars from *Yahoo! Autos* can be shown to the user.

Mediator	$GS(Make, Model, Year, Price, Mileage, Location, Body\ Style)$
Cars.com	$LS(Make, Model, Year, Price, Mileage, Location, Body\ Style)$
Yahoo! Autos	$LS(Make, Model, Year, Price, Mileage, Location)$

Figure 2: Global schema and local schema of data sources

In order to retrieve relevant possible answers from *Yahoo! Autos*, we apply the attribute correlation, value distribution, and selectivity estimates learned on the *Cars.com* database to the *Yahoo! Autos* database. For example, suppose that we have mined an AFD $Model \rightsquigarrow Body\ Style$ from the *Cars.com* database. To retrieve relevant possible answers from the *Yahoo! Autos* database, the mediator issues rewritten queries to *Yahoo! Autos* using the base set and AFDs from the *Cars.com* database.

The algorithm that retrieves relevant tuples from a source S_k not supporting the query attribute is similar to the QPIAD Algorithm presented in Section 4.2, except that the base result set is retrieved from the *correlated source* S_c in Step 1.

DEFINITION 4 (CORRELATED SOURCE). *For any autonomous data source S_k not supporting a query attribute A_i, we define a* correlated source S_c *as any data source that satisfies the following: (i) S_c supports attribute A_i in its local schema, (ii) S_c has an AFD where A_i is on the right hand side, (iii) S_k supports the determining set of attributes in the AFD for A_i mined from S_c.*

From all the sources correlated with a given source S_k, we use the one for which the AFD for A_i has the highest confidence. Then using the AFDs, value distributions, and selectivity estimates learned from S_c, ordered rewritten queries are generated and issued in Step 2 to retrieve relevant possible answers for the user query from source S_k.

4.4 Handling Aggregate Queries

As the percentage of incomplete tuples increases, aggregates such as *Sum* and *Count* need to take the incomplete tuples into account to get accurate results. To support aggregate queries, we first retrieve the base set by issuing the user's query to the incomplete database. Besides computing the aggregate over the base set (certain answers), we also use the base set to generate rewritten queries according to the QPIAD algorithm in Section 4.2. For example, consider the aggregate query Q: $\sigma_{Body\ Style=Convt \wedge Count(*)}$ over the Car database fragment in Table 2. First, we would retrieve the certain answers t_1, t_2, and t_3 for which we would compute their certain aggregate value $Count(*) = 3$. As mentioned previously, our first choice could be to simply return this certain answer to the user effectively ignoring any incomplete tuples. However, there is a better choice, and that is to generate rewritten queries according to the algorithm in Section 4.2 in an attempt to retrieve relevant tuples whose $BodyStyle$ attribute is *null*.

When generating these rewritten queries, tuple t_2 from the base set would be used to form the rewritten query Q'_2: $\sigma_{Model=Z4}$ based on the AFD $Model \rightsquigarrow Body\ Style$. Before issuing the query to the database we must first consider how to combine the certain and possible aggregate values. When combining the certain and possible aggregate values, we have two options (i) combine a fraction of the rewritten query's aggregate result with the certain aggregate where the fraction is equal to the query's precision or (ii) combine the entire rewritten query's aggregate result with the certain aggregate but do so only for those queries in which the most likely value is equal to the value of the constrained query attribute.

Using the first approach, we would find $P(Body\ Style{=}Convt|Model{=}Z4)$, issue the rewritten query, and take this probability, namely the query's precision, and multiply it with the aggregate result which is returned. This fraction of the aggregate is then added to the certain aggregate. For example, assuming the probability above, the final resulting aggregate over the incomplete data source would be calculated as

$$Count_{Total}(*) = Count_{Certain}(*) + (Count_{Possible}(*))(P(Convt|Z4))$$

and the process repeated for tuples t_1 and t_3 whose corresponding rewritten queries are Q'_1: $\sigma_{Model=A4}$ and Q'_3: $\sigma_{Model=Boxster}$ respectively.

Using the second approach, we would find the probability distribution over all $Body\ Style$ values given that the $Model$ is known. Since the original query Q was on $Body\ Style{=}Convt$ we check the $Body\ Style$ distribution to find the value with the highest probability. If the value with the highest probability happens to be $Convt$ then the entire aggregate from the rewritten query combined with the certain aggregate, if the highest probability is not for the value $Convt$ then the rewritten query's aggregate is discarded. Therefore, when considering the rewritten query Q'_2: $\sigma_{Model=Z4}$ from above, the final resulting aggregate over the incomplete data source would be

$Count_{Total}(*)$=$Count_{Certain}(*)$+$Count_{Possible}(*)$=3+1=4 assuming that $Convt$ is the maximum predicted probability given that $Model$=$Z4$. In Section 6, we present the results of our empirical evaluation on aggregate query processing in the context of QPIAD. The results show an improvement in the aggregate value accuracy when incomplete tuples are included in the calculations.

4.5 Handling Join Queries

To support joins over incomplete autonomous data sources, the results are retrieved independently from each source and then joined by the mediator. When retrieving possible answers, the challenge comes in deciding which rewritten queries to issue to each of the sources and in what order.

We must consider both the precision and estimated selectivity when ordering the rewritten queries. Furthermore, we need to ensure that the results of each of these queries agree on their join attribute values. Given that the mediator provides the global schema, a join query posed to the mediator must be broken down as a pair of queries, one over each autonomous relation. In generating the rewritten queries, we know the precision and selectivity estimates for each of the pieces, thus our goal is to combine each pair of queries and compute a combined estimate of precision and selectivity. It is important to consider these estimates in terms of the query pair as a whole rather than simply considering the estimates of the pair's component queries alone. For example, when performing a join on the results of two rewritten queries, it could be the case that the top ranked rewritten query from each relation does not have join attribute values in common. Therefore despite their high ranks at each of their local relations, the query pair could return little or no answers. As a result, when retrieving both certain and possible answers to a query, the mediator needs to order and issue the rewritten queries intelligently so as to maximize the precision/recall of the joined results.

In processing such join queries over relations $R1$ and $R2$, we must consider the orderings of each pair of queries from the sets $Q1 \cup Q1'$ and $Q2 \cup Q2'$ where $Q1$ and $Q2$ are the complete queries derived from the user's original join query over the global schema and $Q1'$ and $Q2'$ are the sets of rewritten queries generated from the bases sets retrieved from $R1$ and $R2$ respectively. Given that the queries must return tuples whose join attribute values are the same in order for a tuple to be returned to the user, we now consider adjusting the α parameter in our *F-Measure* calculation so as to give more weight to recall without sacrificing too much precision. The details of the approach taken by QPIAD are as follows:[7]

1. Send complete queries $Q1$ and $Q2$ to the databases $R1$ and $R2$ to retrieve the base result sets $RS(Q1)$ and $RS(Q2)$ respectively.
2. For each base set, generate a list of rewritten queries $Q1'$ and $Q2'$ using the QPIAD rewriting algorithm described in Section 4.2.
3. Compute the set of all query pairs QP by taking the Cartesian product of each query from the sets $Q1 \cup Q1'$ and $Q2 \cup Q2'$. For each pair, calculate the new estimated precision, selectivity, and *F-Measure* values.
   (a) For each rewritten query in $Q1'$ and $Q2'$, use the NBC classifiers to determine the join attribute value distributions $JD1$ and $JD2$ given the determining set attribute values from the base sets $RS1$ and $RS2$ respectively as discussed in Section 5.2.
   (b) For each join attribute value v_{j1} and v_{j2} in $JD1$ and $JD2$ respectively, compute its estimated selectivity as the product of the rewritten query's precision, selectivity, and the value probability distribution for either v_{j1} or v_{j2}.
   (c) For each query pair $qp \in QP$ compute the estimated selectivity of the query pair to be

$$EstSel(qp) = \sum_{\substack{v_{j1} \in JD1 \\ v_{j2} \in JD2}} EstSel(qp_1, v_{j1}) * EstSel(qp_2, v_{j2})$$

4. For each query pair, compute its *F-Measure* score using the new precision, estimated selectivity, and recall values. Next, select the top-K query pairs from the ordered set of all query pairs according to the algorithm described in Section 4.2.
5. For each selected query pair qp, if the component queries qp_1 and qp_2 have not previously been issued as part of another query pair, issue them to the relations $R1$ and $R2$ respectively to retrieve the extended result sets $\widehat{RS1}$ and $\widehat{RS2}$.
6. For each tuple $\widehat{t_{i1}}$ in $\widehat{RS1}$ and $\widehat{t_{i2}}$ in $\widehat{RS2}$ where $\widehat{t_{i1}}.v_{j1} = \widehat{t_{i2}}.v_{j2}$ create a possible joined tuple. In the case where either $\widehat{t_{i1}}.v_{j1}$ or $\widehat{t_{i2}}.v_{j2}$ is null, predict the missing value using the NBC classifiers and create the possible join tuple. Finally, return the possible joined tuple to the user.

5. LEARNING STATISTICS TO SUPPORT RANKING AND REWRITING

As we have discussed, to retrieve possible answers in the order of their relevance, QPIAD requires three types of information: (i) attribute correlations in order to generate rewritten queries (ii) value distributions in order to estimate the precision of the rewritten queries, and (iii) selectivity estimates which combine with the value distributions to order the rewritten queries. In this section, we present how each of these are learned. Our solution consists of three stages. First, the system mines the inherent correlations among database attributes represented as AFDs. Then it builds Naïve Bayes Classifiers based on the features selected by AFDs to compute probability distribution over the possible values of the missing attribute for a given tuple. Finally, it uses the data sampled from the original database to produce estimates of each query's selectivity. We exploit AFDs for feature selection in our classifier as it has been shown that appropriate feature selection before classification can improve learning accuracy[5]. For a more in depth evaluation of our feature selection techniques we refer the reader to [18].

5.1 Learning Attribute Correlations by Approximate Functional Dependencies(AFDs)

In this section, we describe the method for mining AFDs from a (probed) sample of database. Recall that an AFD ϕ is a functional dependency that holds on all but a small fraction of tuples. According to [20], we define the *confidence* of an AFD ϕ on a relation R as: $conf(\phi) = 1 - g_3(\phi)$, where g_3 is the ratio of the minimum number of tuples that need to be removed from R to make ϕ a functional dependency on R. Similarly, we define *approximate key(AKey)* which is a key of all but a small fraction of tuples in R. We use TANE[13] algorithm to discover AFDs and AKeys whose confidence is above a threshold α, which is set to 0.3 in QPIAD to ensure that we do not miss any significant AFDs.

Pruning Noisy AFDs: In most cases, AFDs with high confidence are desirable for learning probability distributions for missing values. However, not all high confidence AFDs are useful for feature selection, such as high confidence AKeys whose values are distinct.

[7]The selectivity estimation steps are only performed for the rewritten queries because the true selectivity of the complete queries is already known once the base set is retrieved.

For example, consider a relation *car(VIN, Model, Make)*. After mining, we find that *VIN* is an AKey (in fact, a key) which determines all other attributes. Given a tuple t with null value on *Model*, its *VIN* is not helpful in estimating the missing *Model* value, since there are no other tuples sharing t's *VIN* value. Therefore AFDs with a superset of AKey attributes in the determining set are not useful features for classification and should be removed. For example, suppose we have an AFD $\{A_1, A_2\} \rightsquigarrow A_3$ with confidence 0.97, and an AKey $\{A_1\}$ with confidence 0.95. Since most of $\{A_1, A_2\}$ value pairs would be distinct, this AFD won't be useful in predicting the values for A_3 and needs to be pruned. An AFD will be pruned if its confidence difference to the corresponding AKey is below a threshold δ(currently set at 0.3 based on experimentation).

5.2 Learning Value Distributions using Classifiers

Given a tuple with a null value, we now need to estimate the probability of each possible value of this null. We reduce this problem to a classification problem using mined AFDs as selected features. A classifier is a function f that maps a given attribute vector $\vec{x}$ to a confidence that the vector belongs to a class. The input of our classifier is a random sample S of an autonomous database R with attributes $A_1, A_2, \cdots, A_n$ and the mined AFDs. For a given attribute A_m, $(1 \leq m \leq n)$, we compute the probabilities for all possible class values of A_m, given all possible values of its determining set $dtrSet(A_m)$ in the corresponding AFDs.

We construct a Naïve-Bayes Classifier(NBC) A_m. Let a value v_i in the domain of A_m represent a possible class for A_m. Let $\vec{x}$ denote the values of $dtrSet(A_m)$ in a tuple with null on A_m. We use Bayes theorem to estimate the probabilities: $P(A_m{=}v_i|\vec{x}) = \frac{P(\vec{x}|A_m=v_i)P(A_m=v_i)}{P(\vec{x})}$ for all values v_i in the domain. To improve computation efficiency, NBC assumes that for a given class, the features $X_1, \cdots, X_n$ are conditionally independent, and therefore we have: $P(\vec{x}|A_m{=}v_i) = \prod_i P(x_i|A_m{=}v_i)$. Despite this strong simplification, NBC has been shown to be surprisingly effective[9]. In the actual implementation, we adopt the standard practice of using NBC with a variant of Laplacian smoothing called m-estimates[24] to improve the accuracy.

5.3 Combining AFDs and Classifiers

So far we glossed over the fact that there may be more than one AFD associated with an attribute. In other words, one attribute may have multiple determining set with different confidence levels. For example, we have the AFD $Model \rightsquigarrow Make$ with confidence 0.99. We also see that certain types of cars are made in certain countries, so we might have an AFD $Country \rightsquigarrow Make$ with some confidence value. As we use AFDs as a feature selection step for NBC, we experimented with several alternative approaches for combining AFDs and classifiers to learn the probability distribution of possible values for null. One method is to use the determining set of the AFD with the *highest confidence* which we call the *Best-AFD* method. However, our experiments showed that this approach can degrade the classification accuracy if its confidence is too low. Therefore we ignore AFDs with confidence below a threshold (which is currently set to be 0.5 based on experimentation), and instead use all other attributes to learn the probability distribution using NBC. We call this approach *Hybrid One-AFD*. We could also use an *Ensemble of classifiers* corresponding to the set of AFDs for each attribute, and then combine the probability distribution of each classifier by a weighted average. At the other extreme, we could ignore feature selection based on AFD completely but use all the attributes to learn probability distribution using NBC. Our experiments described in Section 6 show that Hybrid One-AFD approach has the best classification accuracy among these choices.

5.4 Learning Selectivity Estimates

As discussed in Section 4, the F-measure ranking requires an estimate of the selectivity of a rewritten query. This is computed as $SmplSel(Q) * SmplRatio(R) * PerInc(R)$, where $SmplSel(Q)$ is the selectivity of the rewritten query Q when it is issued to the sample. *SmplRatio(R)* is the ratio of the original database size over the size of the sample. We send queries to both the original database and its sample off-line, and use the cardinalities of the result sets to estimate the ratio. *PerInc(R)* is the percentage of tuples that are incomplete in the database. It can be estimated as the percentage of incomplete tuples that we encountered while creating the sample database.

6. EMPIRICAL EVALUATION

In this section, we describe the implementation and an empirical evaluation of our system QPIAD for query processing over incomplete autonomous databases.

6.1 Implementation and User Interface

The QPIAD system is implemented in Java and has a web-form based interface through which the users issue their queries. A live demo of the QPIAD prototype is available at `http://styx.dchp.asu.edu:8080/QPIADWeb`. Given a user query, the system returns each relevant possible answer to the user along with a *confidence* measure equal to the answer's assessed degree of relevance. Although the confidence estimate could be biased due to the imperfections of the learning methods, its inclusion can provide useful guidance to the users, over and above the ranking.

In addition, QPIAD can optionally "explain" its relevance assessment by providing snippets of its reasoning as support. In particular, it justifies the confidence associated with an answer by listing the AFD that was used in making the density assessment. In the case of our running example, the possible answer t_4 for the query Q' will be justified by showing the learned AFD $Model \rightsquigarrow Body\ Style$.

6.2 Experimental Settings

To evaluate the QPIAD system, we performed evaluations over three data sets. The first dataset, *Cars(year, make, model, price, mileage, body style, certified)*, is built by extracting around 55,000 tuples from *Cars.com*. Databases like this one are inherently incomplete as described in Table 1. The second dataset, *Census(age, workshop, education, marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country)*, is the *United States Census* database made up of 45,000 tuples which we obtained from the UCI data repository.[8] The third dataset, *Complaints(model, year, crash, fail date, fire, general component, detailed component, country, ownership, car type, market)*, is a *Consumer Complaints* database which contains roughly 200,000 tuples collected from the NHSTA Office of Defect Investigations repository[9] and is used in conjunction with the *Cars* database for evaluating join queries.

In order to perform an *oracular study* based on a *ground truth*, we consider only the complete tuples and then artificially introduce

[8] http://www.ics.uci.edu/mlearn/MLRepository.html

[9] http://www-odi.nhtsa.dot.gov/cars/problems/complain/

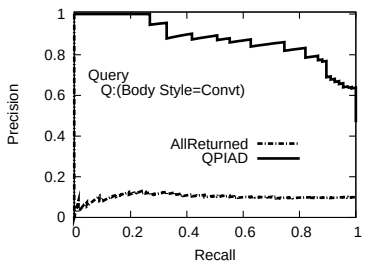


Figure 3: Comparison of ALLRETURNEDand QPIADfor Query $Q(Cars)$: $\sigma_{BodyStyle=Convt}$.

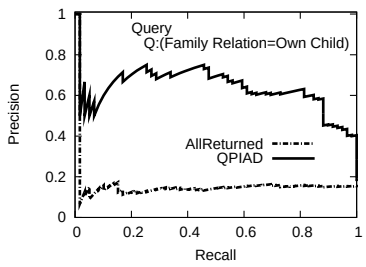


Figure 4: Comparison of ALLRETURNED and QPIAD for Query $Q(Census)$: $\sigma_{FamilyRelation=OwnChild}$.

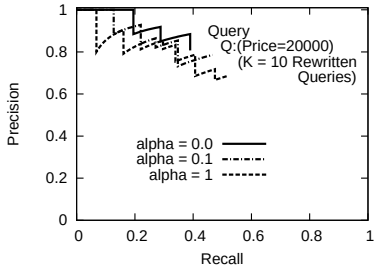


Figure 5: Effect of α on precision and recall in QPIAD.

the data incompleteness. Specifically, for each database, we start with only complete tuples and take this to be the oracular database. Using a copy of the oracular data, we construct our test database by randomly selecting 10% of the tuples each of which are then made incomplete by setting one attribute to *null* at random. Next, we partitioned each database into two parts: a training set (i.e. the sample from which AFDs and classifiers are learned) and a test set. To simulate the relatively small percentage of the training data available to the mediators, we experimented with training sets of different sizes, ranging from 3% to 15% of the entire database, as will be discussed in Section 6.5.

To compare the effectiveness of retrieving relevant possible answers, we consider two salient dimensions of the QPIAD approach, namely *Ranking* and *Rewriting*, which we evaluate in terms of *Quality* and *Efficiency* respectively. For the experiments, we randomly formulate single attribute and multi attribute selection queries and retrieve possible answers from the test databases.

We compare QPIAD with the ALLRETURNED and ALLRANKED approaches. Recall that ALLRETURNED approach presents all tuples containing missing values on the query constrained attribute without ranking them. The ALLRANKED approach begins by retrieving all the certain and possible answers, as in ALLRETURNED, then it ranks possible answers according to the classification techniques described in Section 5. In fact, neither approach is feasible as web databases are unlikely to support binding of null values in queries. In contrast, the QPIAD approach uses query rewriting techniques to retrieve only relevant possible answers in a ranked order and fits for web applications. Even when bindings of null values are allowed, we show in this section that the QPIAD approach provides better quality and efficiency.

In the rest of the evaluation, we focus on comparing the effectiveness of retrieving relevant possible answers. In other words, all the experiments presented in this section, except for those on aggregate queries, ignore the "certain" answers as all the approaches are expected to perform equally well over such tuples.

6.3 Evaluation of Quality

To evaluate the effectiveness of QPIAD ranking, we compare it against the ALLRETURNED approach which simply returns to the user all tuples with missing values on the query attributes. Figures 3 and 4 show the precision and recall curves of a query on the *Cars* and *Census* databases respectively. It shows that the QPIAD approach has significantly higher precision when compared to ALLRETURNED.

To reflect the "density" of the relevant answers along the time line, we also plot the precision of each method at the time when first $K(K{=}1, 2, \cdots, 100)$ answers are retrieved as shown in Figures 6 and 7. Again QPIAD is much better than ALLRETURNED in retrieving relevant possible answers in the first K results, which is critical in web scenarios.

Effect of Alpha Value on F-Measure: To show the effect of α on precision and recall of the query, we present the result for a query $Price = 20000$ with different α values in Figure 5. The number of rewritten queries was restricted to 10. we can see that as the value of α is increased from 0, QPIAD gracefully trades precision for recall.

6.4 Evaluation of Efficiency

To evaluate the effectiveness of QPIAD's rewriting, we compare it against the ALLRANKED approach which simply retrieves all the tuples having missing values on the query constrained attributes and then ranks all such tuples according to their relevance to the query. As we mentioned earlier, we do not expect ALLRANKED approach to be feasible at all for many real world autonomous sources as they do not allow direct retieval of tuples with null values on specific attributes. Nevertheless, these experiments are conducted to show that QPIAD outperforms ALLRANKED even when null value selections are allowed. Figure 8 shows the number of tuples that are retrieved by the ALLRANKED and QPIAD approaches respectively in order to obtain a desired level of recall. As we can see, the number of tuples retrieved by the ALLRANKED approach is simply the total number of tuples with missing values on the query attribute, hence it is independent of the desired level of recall. On the other hand, the QPIAD approach is able to achieve similar levels of recall while only retrieving a small fraction of the tuples retrieved by ALLRANKED. The reason for this is that many of the tuples retrieved by ALLRANKED, while having missing values on the query attributes, are not very likely to be the value the user is interested in. QPIAD avoids retrieving irrelevant tuples and is therefore very efficient.

6.5 Evaluation of Learning Methods

Accuracy of Classifiers: Since we use AFDs as a basis for feature selection when building our classifiers, we perform a baseline study on their accuracy. For each tuple in the test set, we compute the probability distribution of possible values of a null, choose the one with the maximum probability and compare it against the actual value. The classification accuracy is defined as the proportion of the

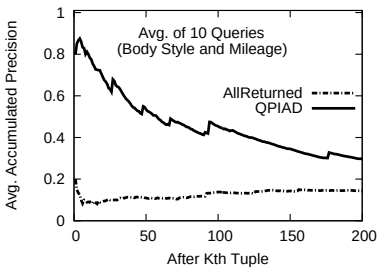


Figure 6: Avg. Accumulated Precision After Retrieving the Kth Tuple Over 10 Queries (Body Style and Mileage).

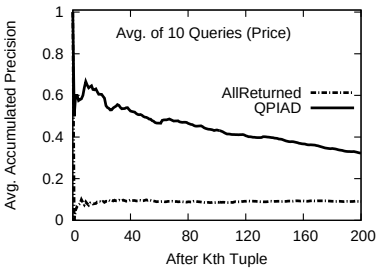


Figure 7: Avg. Accumulated Precision After Retrieving the Kth Tuple Over 10 Queries (Price).

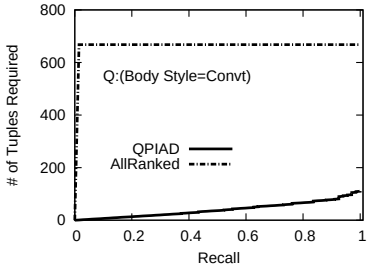


Figure 8: Number of Tuples Required to Achieve a Given Level of Recall for Query $Q(Cars)$: $\sigma_{BodyStyle=Convt}$.

tuples in the test set that have their null values predicted correctly.

Table 3 shows the average prediction accuracy of various AFD-enhanced classifiers introduced in Section 5.3. In this experiment, we use a training set whose size is 10% of the database. The classification accuracy is measured over 5 runs using different training set and test set for each run. Considering the large domain sizes of attributes in *Cars* database (varying from $2(Certified)$ to $416(Model)$), the classification accuracy obtained is quite reasonable, since a random guess would give much lower prediction accuracy. We can also see in Table 3 that the Hybrid One-AFD approach performs the best and therefore is used in our query rewriting implementation.[10]

Database	Best AFD	All Attributes	Hybrid One-AFD
Cars	68.82	66.86	68.82
Census	72	70.51	72

Table 3: Null value prediction accuracy across different AFD-enhanced classifiers

While classifier accuracy is not the main focus of our work, we did do some comparison studies to ensure that our classifier was competitive. Specifically, we compared AFD-enhanced NBC classifier with two other approaches — one based on association rules[35] and the other based on learning Bayesian networks from the data [12]. For Bayes network learning, we experimented with the WEKA Data Mining Software[11]. We found that although the AFD-enhanced classifiers were significantly cheaper to learn than Bayes networks, their accuracy was competitive. To compare our approach against association-rule based classifiers, we used the algorithm proposed in [35]. Our experiments showed that association rules perform poorly as they focus only on attribute-value level correlations and thus fail to learn from small samples. In contrast AFD-enhanced NBC classifiers can synergistically exploit schema-level and value-level correlations. Details of these evaluations are available [18].

Robustness w.r.t. Confidence Threshold on Precision: QPIAD presents ranked relevant possible answers to users along with a confidence so that the users can use their own discretion to filter off answers with low confidence. We conducted experiments to evaluate how pruning answers based on a confidence threshold affects the precision of the results returned. Figure 9 shows the average precision obtained over 40 test queries on Cars database by pruning answers based on different confidence thresholds. It shows that the high confidence answers returned by QPIAD are most likely to be relevant answers.

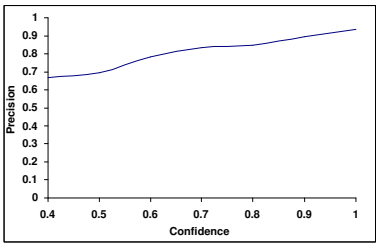


Figure 9: Average Precision for various confidence thresholds(Cars).

Robustness w.r.t. Sample Size: The performance of QPIAD approach, in terms of precision and recall, relies on the quality of the AFDs, Naïve Bayesian Classifiers and selectivity estimates learned by the knowledge mining module. In data integration scenario, the availability of the sample training data from the autonomous data sources is restrictive. Here we present the robustness of the QPIAD approach in the face of limited size of sample data. Figure 10 shows the accumulated precision of a selection query on the Car database, using various sizes of sample data as training set. We see that the quality of the rewritten queries all fluctuate in a relatively narrow range and there is no significant drop of precision with the sharp decrease of sample size from 15% to 3%. We obtained a similar result for the Census database [18].

6.6 Evaluation of Extensions

Effectiveness of using Correlation Between Data Sources: We consider a mediator performing data integration over three data sources *Cars.com* (www.cars.com), *Yahoo! Autos* (autos.yahoo.com) and *CarsDirect* (www.carsdirect.com). The global schema supported by the mediator and the individual local schemas

[10] In Table 3 the Best-AFD and Hybrid One-AFD approaches are equal because there were high confidence AFDs for all attributes in the experimental set. When this is not the case, the Hybrid One-AFD approach performs better than the Best-AFD approach.

[11] Available from http://www.cs.waikato.ac.nz/ml/weka/

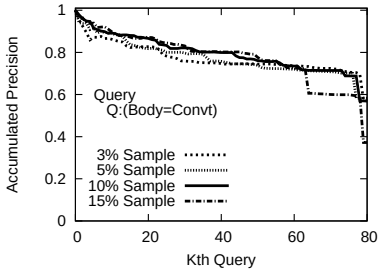
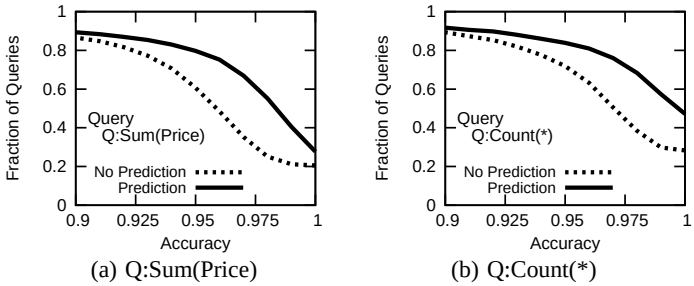


Figure 10: Accumulated precision curve with different sample sizes on Cars database.



(a) Q:Sum(Price) (b) Q:Count(*)

Figure 12: Accuracy of aggregate queries with and without missing value prediction.

are shown in Figure 2. The schema of *CarsDirect* and *Yahoo! Autos* do not support $Body\ Style$ attribute while *Cars.com* does support queries on the $Body\ Style$. We use the AFDs and NBC classifiers learned from *Cars.com* to retrieve cars from *Yahoo! Autos* and *CarsDirect* as possible relevant possible answers for queries on $Body\ Style$, as discussed in Section 4.3.

To evaluate the precision, we check the actual $Body\ Style$ of the retrieved car tuples to determine whether the tuple was indeed relevant to the original query. The average precision for the first K tuples retrieved from *Yahoo! Autos* and *CarsDirect* over the 5 test queries is quite high as shown in Figure 11. This shows that using the AFDs and value distributions learned from correlated sources, QPIAD can retrieve relevant answers from data sources not supporting query attribute.

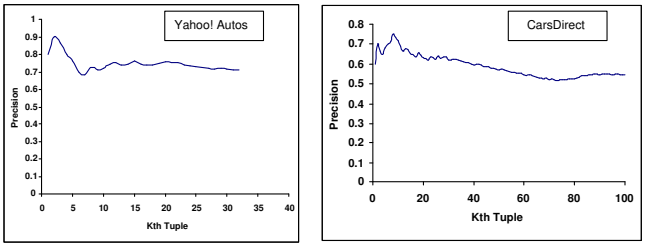


Figure 11: Precision curves for first K tuples retrieved using correlated source *Cars.com*.

Evaluation of Aggregate Queries: To evaluate our approach in terms of supporting aggregate queries, we measured the accuracy of aggregation queries in QPIAD where missing values in the incomplete tuples are predicted and used to compute the final aggregate result. We compare the accuracy of our query rewriting and missing value prediction with the aggregate results from the complete oracular database and the aggregate results from the incomplete database where incomplete tuples are not considered. Next we will outline the details of our experiments.

We performed the experiments over an *Cars* database consisting of 8 attributes. First, we created all distinct subsets of attributes where the size of the subsets ranged from 1 to 7 (e.g. $\{make\}$, $\{make, model\}$, $\{make, model, year\}$, ..., $\{model\}$, $\{model, year\}$, ..., *etc.*). Next, we issued a query to the sample database and selected the distinct combinations of values for each of these subsets.

Using the distinct value combinations for each of these subsets, we created queries by binding the values to the corresponding attribute in the subsets. We then issued each query to the complete database to find its true aggregate value. We also issued the same query to the incomplete database and computed the aggregate value without considering incomplete tuples. Finally, we issued the query to the incomplete database only this time we predicted the missing values and included the incomplete tuples as part of the aggregate result. When predicting the missing values, we only included incomplete tuples whose missing attribute value is mostly likely to be the value constrained in the query, namely *method 2* in Section 4.4.

In Figure 12, we show the percentage of queries which achieve different levels of accuracy with and without missing value prediction. The results are significant, for example, Figure 12(a) shows that when missing value prediction is used to computed the aggregate result, roughly 10% more queries achieve 100% accuracy than if the aggregate had only taken the certain tuples into account, thus ignoring all incomplete ones.

Evaluation of Join Queries: To evaluate our approach in the context of join queries we performed a set of experiments on the *Cars* and *Complaints* databases. In the experiments, we join the *Cars* and *Complaints* relations on the $Model$ attribute. The experimental results shown in Figure 13 involve join queries where the attributes from both the relations are constrained. We evaluate the performance of our join algorithm in terms of precision and recall with respect to a complete oracular database.

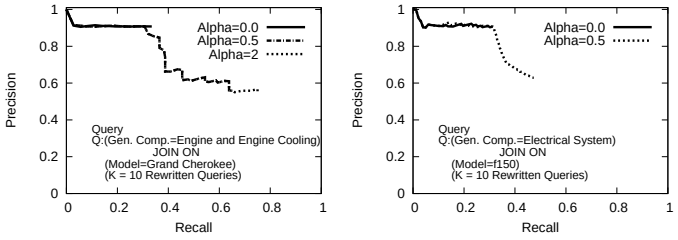


Figure 13: Precision-Recall Curves for Queries on Cars Join Complaints

We present the results for a join query $Model = Grand\ Cherokee \wedge General\ Component = Engine\ and\ Engine\ Cooling$. We set α to 0, 0.5 and 2 to measure the effect of giving different preferences to precision and recall. In addition, we restricted the number of rewritten queries which could be sent to the database to 10 queries. Figure 13(a) shows the precision-recall curve for this query. We can see that for $\alpha = 0$ high precision is maintained but recall stops at 0.34. For $\alpha = 0.5$ the precision is the same as when $\alpha = 0$ up until recall reaches 0.31. At this point,

the precision decreases although, a higher recall, namely 0.66, is achieved. The precision when $\alpha = 2$ is similar to the case where $\alpha = 0.5$ but achieves 0.74 recall with only a small loss in precision near the tail of the curve. When looking at the top 10 rewritten queries for each of these α values we found that when $\alpha = 0$, too much weight is given to precision and thus incomplete tuples are never retrieved from the *Cars* database. This is due to our ability to predict missing values which happens to be better on the *Complaints* database and hence the top 10 rewritten queries tend to include the complete query from the *Cars* database paired with an incomplete query from the *Complaints* database. However, when $\alpha = 0.5$ or $\alpha = 2$ incomplete tuples are retrieved from both the databases because in this approach the ranking mechanism tries to combine both precision and recall. Similar results for the query Q :$_{Model=f150 \wedge General\ Component=Electrical\ System}$ are shown in Figure 13(b).

7. CONCLUSION

Incompleteness is inevitable in autonomous web databases. Retrieving highly relevant possible answers from such databases is challenging due to the restricted access privileges of mediator, limited query patterns supported by autonomous databases, and sensitivity of database and network workload in web environment. We developed a novel query rewriting technique that tackles these challenges. Our approach involves rewriting the user query based on the knowledge of database attribute correlations. The rewritten queries are then ranked by leveraging attribute value distributions according to their likelihood of retrieving relevant possible answers before they are posed to the databases. We discussed rewriting techniques for handling queries containing selection, joins and aggregations. To support such query rewriting techniques, we developed methods to mine attribute correlations in the form of AFDs and the value distributions of AFD-enhanced classifiers, as well as query selectivity from a small sample of the database itself. Our comprehensive experiments demonstrated the effectiveness of our query processing and knowledge mining techniques.

As we mentioned, part of the motivation for handling incompleteness in autonomous databases is the increasing presence of databases on the web. In this context, a related issue is handling query imprecision–most users of online databases tend to pose imprecise queries which admit answers with varying degrees of relevance (c.f. [26]). In our ongoing work, we are investigating the issues of simultaneously handling data incompleteness and query imprecision [17].

8. REFERENCES

[1] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. *ICDE*, 2006.

[2] Lyublena Antova, Christoph Koch, Dan Olteanu 10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information In *Proc. ICDE* 2007.

[3] G. E. A. P. A. Batista and M. C. Monard. A study of k-nearest neighbour as an imputation method. In *HIS*, 2002.

[4] Bertossi, L. Consistent Query Answering in Databases. ACM Sigmod Record, June 2006, 35(2):68-76.

[5] A. Blum and P. Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 1997.

[6] D. Burdick, P. Deshpande, T. Jayram, R. Ramakrishnan, S. Vaithyanathan. OLAP over uncertain and imprecise data. In *VLDB*, 2005.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. *SIGMOD*, 2003.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via em algorithm. In *JRSS*, pages 1–38, 1977.

[9] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[10] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD Conference*, pages 155–166, 2005.

[11] R. Gupta and S. Sarawagi. Creating Probabilistic Databases from Information Extraction Models. VLDB 2006: 965-976

[12] D. Heckerman. A tutorial on learning with bayesian networks, 1995.

[13] Y. Huhtala, J. Krkkinen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE Conference*, pages 392–401, 1998.

[14] I. Ihab, A. Walid, E. Ahmed Supporting Top-k Join Quereis in Relational Databases. In *VLDB*, 2003.

[15] T. Imieliski and J. Witold Lipski. Incomplete information in relational databases. *Journal of ACM*, 31(4):761–791, 1984.

[16] Z. G. Ives, A. Y. Halevy, and D. S. Weld. Adapting to source properties in processing data integration queries. *SIGMOD*, 2004.

[17] S. Kambhampati, G. Wolf, Y. Chen, H. Khatri, B. Chokshi, J .Fan, U. Nambiar QUIC: A System for Handling Imprecision & Incompleteness in Autonomous Databases. In *CIDR*, 2007 (Demo).

[18] H. Khatri. *Query Processing over Incomplete Autonomous Web Databases, MS Thesis, Dept. of CSE, Arizona State University,* rakaposhi.eas.asu.edu/hemal-thesis.pdf. 2006.

[19] H. Khatri, J. Fan, Y. Chen, S. Kambhampati QPIAD: Query Processing over Incomplete Autonomous Databases. In *ICDE*, 2007 (Poster).

[20] J. Kivinen and H. Mannila. Approximate dependency inference from relations. In *ICDT Conference*, 1992.

[21] D. Lembo, M. Lenzerini, and R. Rosati. Source inconsistency and incompleteness in data integration. *KRDB*, 2002.

[22] W. Lipski. On semantic issues connected with incomplete information databases. *ACM TODS*, 4(3):262–296, 1979.

[23] J. Madhavan, A. Halevy, S. Cohen, X. Luna Dong, S. Jeffery, D. Ko, C. Yu. Structured Data Meets the Web: A Few Observations. IEEE Data Eng. Bull. 29(4): 19-26 (2006)

[24] T. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[25] I. Muslea and T. J. Lee. Online query relaxation via bayesian causal structures discovery. In *AAAI*, pages 831–836, 2005.

[26] U. Nambiar, S. Kambhampati. Answering Imprecise Queries over Autonomous Web Databases. In *ICDE*, 2006.

[27] A. Ola and G. Özsoyoglu. Incomplete relational database models based on intervals. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):293–308, 1993.

[28] L. Popa and V. Tannen. An equational chase for path-conjunctive queries, constraints, and views. In *ICDT*, 1999.

[29] M. Ramoni and P. Sebastiani. Robust learning with missing data. *Mach. Learn.*, 45(2):147–170, 2001.

[30] D. B. R. Roderick J. A. Little. *Statistical Analysis with Missing Data,Second edition*. Wiley, 2002.

[31] A. D. Sarma, O. Benjelloun, A. Y. Halevy, and J. Widom. Working models for uncertain data. In *ICDE*, 2006.

[32] L. Si and J. Callan. Relevant Document Distribution Estimation Method for Resource Selection. In *ACM SIGIR*, 2003.

[33] D. Suciu and N. Dalvi. Tutorial: Foundations of probabilistic answers to queries. In *SIGMOD Conference*, 2005.

[34] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[35] C.-H. Wu, C.-H. Wun, and H.-J. Chou. Using association rules for completing missing data. In *HIS Conference*, 2004.