QUIC: Handling Query Imprecision & Data Incompleteness in Autonomous Databases

Subbarao Kambhampati, Garrett Wolf, Yi Chen, Hemal Khatri
Bhaumik Chokshi, Jianchun Fan, Ullas Nambiar
Department of Computer Science and Engineering,
Arizona State University, Tempe, AZ 85287, USA

ABSTRACT

As more and more information from autonomous databases becomes available to lay users, query processing over these databases must adapt to deal with the imprecise nature of user queries as well as incompleteness in the data due to missing attribute values (aka "null values"). In such scenarios, the query processor begins to acquire the role of a recommender system. Specifically, in addition to presenting answers which satisfy the user's query, the query processor is expected to provide highly relevant answers even though they do not exactly satisfy the query predicates. This broadened view of query processing poses several technical challenges. We propose a decision theoretic model for ranking answers in the in the order of their expected relevance to the user. This model combines a relevance function that reflects the relevance a user would associate with answer tuples and a density function which reflects the each tuple's distribution of missing data. Adoption of this model foregrounds three general challenges: (i) how to assess the relevance and density functions automatically (ii) how to support efficient query processing to retrieve relevant tuples and (iii) how to make users trust the recommended answers. We present a general framework for addressing these challenges, describe a preliminary implementation of the QUIC system and discuss the results of our preliminary empirical evaluation.

1. INTRODUCTION

The popularity of the World Wide Web has lead to the presence of multiple online databases that are often laxly curated and readily accessible to lay users. An important challenge facing the database community is to develop effective mediators that can handle the data incompleteness (caused by lax curation) and query imprecision (caused by the lay user population). This challenge brings to the realm of structured data many issues that traditionally were only handled in information retrieval. Dealing with this challenge is further complicated by the fact that the databases are autonomous and only support query-based access to the mediator.

Data Incompleteness: An increasing number of online databases are being populated with little or no curation. For example, databases like `autotrader.com` are populated using automated extraction techniques which crawl text classifieds and by car owners entering the data through forms. Similar techniques are also used in integration systems for scientific data such as *CBioC* (`cbioc.org`). Unfortunately, these methods are error prone (c.f. [17]) which leads to databases that are *incomplete* in that they contain tuples having many null values. In fact, on a random sample of 25,000 tuples from `autotrader.com`, we found that almost 34% were incomplete![1]

Query Imprecision: More and more lay users, be they causal web users, or domain experts that are nonetheless database novices, are accessing autonomous databases on the web. Often these users do not clearly define what they are looking for and many times the form-based query formulating tools do not support imprecise queries. Imprecision can involve posing queries that are either overly general [6] or overly specific [15] w.r.t. the information that is truly relevant. For example, users end up asking queries such as $Q : CarDB(Model{=}Civic)$ when they actually want to ask the query $Q' : CarDB(Model{\approx}Civic)$ (where "$\approx$" is to be interpreted as "like"). A car of model Accord that does not exactly satisfy the query can be relevant if the user was looking for a reliable Japanese car but over-specified the query. Further more, the query may be over-general in that among all the answer tuples, the ones with a lower mileage are preferred more by the user. A mediator system should be able to support such imprecision in the user's queries and return tuples which do not exactly satisfy the query constraints but nevertheless are likely to be relevant to the user.

Example: Consider a fragment of online Car database DB_f as shown in Table 1. Suppose a user poses an imprecise query $Q' : \sigma_{Model \approx Civic}$ on this table. Clearly tuples t_1 and t_6 are relevant as they are exact answers to the query. In addition, the tuples t_4 and t_7 might also be relevant if there are reasons to believe that the missing value might be a Civic. Finally, tuples t_2 and t_8 might be relevant if Civic is considered similar enough to Accord and/or Prelude.

Ideally, a query processor should be able to present such implicitly relevant results to the user, ranking them in terms of their expected relevance. In this paper, we present a general framework as well as a specific current implementation, called QUIC aimed at solving this problem. We start with the philosophy that query imprecision and data incompleteness are best modeled in terms of relevance and density functions. Informally, the relevance function assesses the rel-

[1]This type of incompleteness is expected to increase even more with services such as GoogleBase and Craigslist.com which provide users significant freedom in deciding which attributes to define and/or list.

Id	Make	Model	Year	Color	Body Style
1	Honda	Civic	2000	red	coupe
2	Honda	Accord	2004	blue	coupe
3	Toyota	Camry	2001	silver	sedan
4	Honda	null	2004	black	coupe
5	BMW	3-series	2001	blue	convt
6	Honda	Civic	2004	green	sedan
7	Honda	null	2000	white	sedan
8	Honda	Prelude	1999	blue	coupe

Table 1: Fragment of a Car Database

evance the user places on a tuple t with respect to a query Q. The density function attempts to capture the probability distribution that an incomplete tuple is in reality representing a complete tuple t in that all their non-null attribute values are the same.

Challenges: Given information about relevance and density functions, it is possible, in theory, to rank a set of possibly incomplete tuples in terms of their expected relevance given a specific query. Simple as it sounds, realizing such a query processing architecture presents several technical challenges, brought about mostly due to the autonomous nature of the databases, and the impatience of the user population:

Ranking Challenges: (i) We need a way to combine the density and relevance functions to provide a ranking of the tuples. (ii) Since the users are often impatient, we need methods for automatically and non-intrusively assessing the appropriate relevance function. (iii) Since the databases are autonomous, we need methods for automatically assessing the density function.

Optimization Challenges: Since often the query processor has only a form-based interface to access the databases, we need to be able to rewrite the queries appropriately in order to retrieve tuples that are likely to be relevant to the user. This necessitates novel query rewriting techniques.

Explaining/Justifying Answers: Since the query processor will be presenting tuples that do not correspond to exact answers to the user's query, it needs to provide explanations and justifications to gain the user's trust.

We have been addressing these challenges in the context of a preliminary prototype called QUIC. Our aim in the rest of this paper is to (i) describe the broad issues that arise in tackling these challenges and (ii) describe the approaches that have been taken in the current implementation of QUIC. The architecture of QUIC is shown in Figure 1. QUIC acts as a mediator between the user and autonomous web databases. Because of the autonomous nature of these databases, QUIC has virtually no control or prior knowledge over them. QUIC computes relevance and density functions from a probed sample of the autonomous databases. These functions are in turn used to reformulate the user query in order to retrieve tuples that are relevant even though they may not be exact answers. The retrieved tuples are ranked in terms of a ranking function that we call *expected relevance ranking*. The ranked results are returned with an explanation of the ranking scheme. The current implementation of QUIC uses approximate functional dependencies (AFDs, c.f. [10]) to mine and represent attribute dependencies. As we shall see, AFDs wind up being useful in multiple ways in QUIC.

Contributions over Prior Work: The problem we address brings together several core database challenges — including uncertainty, incompleteness and query imprecision, to the context of web with its autonomous databases and lay user population. The problems of data incompleteness and

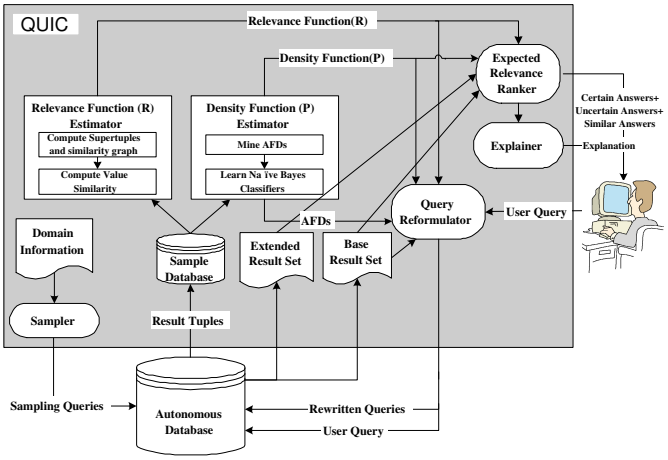

Figure 1: QUIC System Architecture.

query imprecision have, in the past, been tackled in isolation in traditional databases. However, the methods tended to involve direct modification of the databases, to rewrite null values (e.g. [11]) or to insert similarity measures between attribute values (e.g. [13]). The autonomous nature of the web sources, as well as the impatience of the lay users precludes straightforward adaptation of these methods. The complications include the need to automatically, indirectly and non-intrusively assess relevance and density functions, as well as the need to use these assessed functions in reformulating the user query so as to retrieve tuples that are expected to be of relevance to the user.

Assessment of relevance and density functions are special cases respectively of the general problems of "utility/preference elicitation"[4, 5] and "density estimation"[9, 16, 17] that have been studied extensively in the AI, statistics and pattern recognition communities. The challenge we face is adapting, from this welter of literature, the right methods that are sensitive to the autonomous nature of the databases as well as the possibly non-cooperative user population. Given that our treatment of incomplete tuples involves converting them into uncertain tuples, work on probabilistic databases[3, 18, 2] shares some of our goals. It however focuses on single centralized databases, assuming that the uncertainty in the base data is already quantified and can be recorded and handled within the databases themselves. In contrast, QUIC focuses on autonomous databases, and thus must assess the uncertainty in the base data automatically and handle the uncertainty by the mediator without modifying the underlying databases.

Work on QUIC is built on our previous work on the AIMQ system [15] that focuses exclusively on query imprecision, and our more recent work on QPIAD [12] which focuses exclusively on data incompleteness. QUIC extends these prior efforts both in terms of addressing query imprecision and data incompleteness together, as well as investigating a larger spectrum of methods for assessing relevance and density functions, and using them for query rewriting.

Organization: In the next section, we discuss how QUIC handles each of the challenges discussed above. In each sub-section we first discuss the broader challenges, and then present the current approaches taken in QUIC. This is followed in Section 3 by an empirical evaluation of the current implementation of QUIC. We conclude with a discussion of the directions we are currently pursuing.

2. CHALLENGES & CHOICES MADE

In this section, we define *Expected Relevance* as the basis for ranking results in the presence of query imprecision and data incompleteness. We use $\widehat{t}$ to denote a (possibly incomplete) tuple of a database D and t to denote a complete tuple. A complete tuple has a non-null value for each of the attributes, while an incomplete tuple has null values for some of its attributes. A tuple t is considered to belong to the set of completions of $\widehat{t}$ (denoted $C(\widehat{t})$), if t and $\widehat{t}$ agree on all the non-null attribute values.

2.1 Expected Relevance Ranking Model

The expected relevance ranking of a (possibly incomplete) tuple $\widehat{t}$ given a query Q by a user U over a database D, $\mathcal{ER}(\widehat{t}|Q,U,D)$ is specified in terms of two independent functions, the relevance function $\mathcal{R}$ and the density function $\mathcal{P}$ as defined below.

$\sigma_{Model \approx Civic}$	*Civic*	*Accord*	*Prelude*	*Corolla*
Relevance	1.0	0.78	0.59	0.48
Density	0.62	0.21	0.17	0.0

Table 2: Relevance for the query Q': $\sigma_{Model \approx Civic}$ and Density for tuple $\widehat{t_4}$.

Relevance Function $\mathcal{R}$: The relevance function, $\mathcal{R}(t|Q,U)$, is the relevance that user U associates with a complete tuple t as an answer to the query Q. Table 2 shows a possible relevance function for the query Q: $Model \approx Civic$. Here $\mathcal{R}(t|Q,U)$=1 for tuples having $Model$=$Civic$, 0.78 for tuples with $Model$=$Accord$, 0.59 for tuples with $Model$=$Prelude$, and so on. Moreover, it need not be the case that all tuples having $Model$=$Civic$ are of the same relevance to the user. Specifically, by considering the user's query to be under specified, we can give varying relevance scores to tuples with the same values for the constrained attributes (c.f [6]). For example, given two tuples each having $Model$=$Civic$, it could be the case that the car with lower $Mileage$ is more relevant to the user even though they may not have made it explicit.

Density Function $\mathcal{P}$: The density function, $\mathcal{P}(t|\widehat{t},D)$, is the probability that a (possibly incomplete) tuple $\widehat{t}$ from database D corresponds to a complete tuple $t \in C(\widehat{t})$ (where $C(\widehat{t})$ is the set of completions of $\widehat{t}$). Table 2 shows a possible density function for the incomplete tuple $\widehat{t_4}$, which estimates the probabilities of the possible values of the null over the domain of the $Model$ attribute (for example, based on the rest of the attribute values and correlations between attributes). In this case, the distribution of possible values is: 0.62 probability for *Civic*, 0.21 for *Accord*, 0.17 for *Prelude*, etc.

Expected Relevance $\mathcal{ER}$: Our ranking function, $\mathcal{ER}(\widehat{t}|Q,U,D)$ can now be defined as:

$$\mathcal{ER}(\widehat{t}|Q,U,D) = \sum_{t \in C(\widehat{t})} \mathcal{R}(t|Q,U)\mathcal{P}(t|\widehat{t},D)$$

It is easy to see that the ERR model is general enough to handle traditional database query processing where the relevance and density function are dirac delta functions ($\mathcal{R}(t|Q,U)=1$ if $t.A=v$ and 0 otherwise and $\mathcal{P}(t|\widehat{t},D)=1$ if $\widehat{t}=t$ and 0 otherwise) as well as query processing under data incompleteness and/or query imprecision.

Processing the query Q: $Model \approx Civic$ on the car database, we retrieve not only the tuples that exactly satisfy Q, namely t_1 and t_6, but also highly relevant tuples, including incomplete tuples $\widehat{t_4}$ and $\widehat{t_7}$, and similar tuples t_2 and t_8. These tuples are then ranked and presented in the decreasing order of their expected relevance based on the ERR model, where both the relevance and density functions are considered.

2.2 Estimating Relevance Function

Broad Challenges: Assessing the relevance function in general involves preference elicitation. For example, preferences could be used to determine relative importance of attributes as well as similarity between two attribute values. In the web scenarios, preferences could be obtained in several ways: (i)Preferences can be directly expressed by users as part of their queries. Several IR systems allow user specification of imprecise queries. The problem is that shifting the burden of preference expression to the users does not normally work well (ii) Preferences can be learned by monitoring user's clickstream or stare patterns. However, individually monitoring every user's preferences might be expensive. (iii) Preferences can be learned indirectly by analyzing users' past query patterns. Understanding the trade-offs offered by these approaches is an open problem.

Current Implementation in QUIC: QUIC currently focuses on imprecision due to overly-specific queries. Rather than assess user-specific relevance functions, QUIC attempts to assess relevance functions for the whole user population. Further more, QUIC assumes that relevance function is based on attribute value similarity and attribute importance. Currently, three alternative approaches are supported for computing the value similarities: content based filtering, collaborative filtering which uses the click patterns of the user population (where available), and co-occurrence statistics.

Content Based: In QUIC's content based relevance measurement, we learn a relevance function between two attribute values without any user interaction. QUIC issues a query binding each attribute value pair (AV) to a sample database, where the query result forms a supertuple for that AV pair (given a set S of tuples, a supertuple t_S is constructed as a tuple of bags, where each bag contains the possible values a particular attribute took in the tuples in S [15]). QUIC computes the similarity between supertuples (as measured by Jaccard coefficient with bag semantics) to estimate the similarity between AV pairs: $Sim_J\left(t_S^1.A_i, t_S^2.A_i\right) = \frac{|t_S^1.A_i \cap t_S^2.A_i|}{|t_S^1.A_i \cup t_S^2.A_i|}$, where t_S^1, t_S^2 are the supertuples, and $t_S^1.A_i$ ($t_S^2.A_i$) is the bag of values of attribute A_i in t_S^1 (t_S^2).

Co-click Based: In contrast to the content based relevance function assessment which does not involve actual user transactions, we also tested collaborative filtering based on co-click patterns. In this implementation, we obtained user clickstreams from the Yahoo! Autos (`autos.yahoo.com`) web pages by scraping the collaborative recommendations provided by the site. Specifically, each car's web page contained 1 to 3 recommendations for other cars which were viewed by people who also viewed the car on the given page. Using this information, we create an undirected weighted similarity graph for attributes like $Make$ and $Model$ (where multiple links between two nodes are reduced to a single link having a weight equal to the total number of links between the nodes).

Let $CSim(N_1,N_2)$ denote the co-click similarity between two nodes N_1 and N_2 in the similarity graph. If N_1 and N_2 are connected by an edge, we have $CSim(N_1,N_2) =$

$Sim_C(N_1, N_2) = \alpha^{(1-W(N_1,N_2)/(avg(agg(N_1),agg(N_2)))}$, where $agg(N_1)$ is the aggregate sum of the weights for all links between N_1 and all other nodes, $W(N_1, N_2)$ is the weight of the link between nodes N_1 and N_2, and α is a parameter between 0 and 1.

If N_1 and N_2 are connected through a path (rather than an edge) we compute $CSim(N_1, N_2)$ by assessing the transitive similarity by taking the product of all the Sim_C values along the shortest path between N_1 and N_2.

Co-occurrence Based: These approaches leverage the co-occurrence of words to measure the degree of similarity between two objects. In QUIC implementation, we use co-occurrence data gathered by Google Sets(http://labs.google.com/sets), a website that given an item (e.g. a car model), generates a set of frequently co-occurring items. We constructed an undirected graph for co-occurrences of each Make/Model value in our sample database, in a similar manner as generating co-click similarity graph. After constructing the graph, the similarity measurement between two nodes is computed in the same way as the co-click based approach.

Given any of the attribute value similarity measurements discussed above, we compute the relevance function of a tuple t to Q with constrained attributes A_1, ..., A_n as a sum of the weighted similarity measurements for each of the constrained attributes. Here the weights given to each attributeare made to sum to 1 and are computed using the mined AFD confidences as described in [15].

2.3 Estimating Density Function

Broad Challenges: Recall that given an incomplete tuple $\widehat{t}$ from the database, the function $\mathcal{P}(t|\widehat{t}, D)$ gives the probability that a tuple t that is a completion of $\widehat{t}$ belongs to the database. The function $\mathcal{P}$ is easy to specify if we have access to the *generative model* of the database. For a relational database with n attributes, the generative model is simply the *joint probability distribution* $P(A_1, A_2, \cdots, A_n)$. Given the joint, the individual $\mathcal{P}(t|\widehat{t}, D)$ functions can be computed through conditioning and marginalization over the joint. So, we can either assess the joint and use it to compute $\mathcal{P}$ functions or assess $\mathcal{P}$ functions directly. The relative advantages of these approaches depend on the tradeoffs between accuracy and learning costs.

Current Implementation in QUIC: Whether we learn the joint distribution or the $\mathcal{P}$ function directly, one way of going about is to learn their Bayes Net representations from a sample of the database[16, 9]. The cost of learning the distribution depends on the topology of the network. It can be made simple by making strong assumptions on the interactions between the attribute values. Given an incomplete tuple $\widehat{t}$ with a null value on attribute A_i, we need to estimate $\mathcal{P}(t|\widehat{t}, D)$, the probability of a completion t from the completion set $C(\widehat{t})$. In QUIC, we assume that tuples with more than a single predicted null value are not likely to be ranked high given their low relevance to the user hence we only predict at most one null per tuple. In addition, we assume independence between attributes, and assess the density function by learning an AFD-enhanced Naïve Bayes Classifier(NBC) from a sample database [12]. First the system mines the inherent correlations among database attributes represented as AFDs[10]. Then it builds Naive Bayes Classifiers based on the features selected by AFDs to compute probability distribution over the possible values of the missing attribute value for a given tuple. Specifically, given an AFD $A_j \rightsquigarrow A_i$, where $1 \leq j \leq m$, and $t.A_i = v_i$, we have $\mathcal{P}(t|\widehat{t}, D) = \mathcal{P}(\widehat{t}.A_i{=}v_i) \times \prod_{j=1}^{m} P(\widehat{t}.A_j|\widehat{t}.A_i = v_i)$.

We have also investigated other single attribute density estimation techniques, most notably, learning general Bayes Nets[12]. However, our experiments showed that the improved accuracy of the learned Bayes Nets was not enough to offset the significantly increased learning cost involved in searching over the possible Bayes Net topologies (the advantage of Naive Bayes Net is that the topology is fixed).

2.4 Relevance and Correlation-based Query Rewriting

Broad Challenges: The ERR model we discussed in the previous sections can be used to rank the answer tuples in terms of their expected relevance, once potentially relevant tuples are retrieved from the data sources. However, to support practical query processing based on ERR over autonomous databases, it is neither efficient nor feasible to retrieve all tuples and then rank them. Rather, we need to be pro-active in retrieving tuples that are likely to be highly relevant answers to the user's query (besides the ones that exactly satisfy the query). Since most autonomous databases only allow query-based access to their data, we need to develop novel query rewriting techniques to support this. These techniques would need to exploit the relevance and density functions to rewrite the user query appropriately. For example, in a used car data source scenario, given a user query to retrieve *Honda* cars, we could (i) retrieve relevant incomplete tuples by rewriting the query as selection query on *Accord*, *Civic* etc. (to the extent that the assessed density function says that Accords and Civics are correlated with the make Honda. (ii) retrieve relevant complete tuples by rewriting the query as retrieving *Toyota* cars (to the extent that the assessed relevance function says Hondas are similar to Toyotas).

Current Implementation in QUIC: To retrieve only possible similar tuples and incomplete tuples with high expected relevance, QUIC rewrites (expands) an input user query into a set of queries using the AFDs. Let us continue the running example $Q' : \sigma_{Model \approx Civic}$ on Table 1. First, certain answers t_1 and t_6 are retrieved as base set. Given an AFD $\{Make, BodyStyle\} \rightsquigarrow Model$, we know that car $Model$ is closely related to its $Make$ and $BodyStyle$. If a tuple $\hat{t}$ has the same $Make$ and $BodyStyle$ as a tuple t in the base set, then $\hat{t}$ is likely to be a relevant $Model$ to the user. Based on this intuition, the query reformulator generates $Q_1 : \sigma_{Make=Honda \wedge BodyStyle=sedan}$ and $Q_2 : \sigma_{Make=Honda \wedge BodyStyle=coupe}$. The ranking of Q_1 is the same as the ranking of a tuple $Make = Honda \wedge BodyStyle = sedan \wedge Model = null$ with respect to Q'. Similarly for Q_2. Issuing the queries Q_1 and Q_2 will retrieve the extended result set, namely the tuples t_2, $\widehat{t_4}$, $\widehat{t_7}$, and t_8. Among them, t_2 and t_8 are similar tuples whose $Model$ is relevant (but not equal) to $Civic$, and $\widehat{t_4}$ and $\widehat{t_7}$ are incomplete tuples, whose possible values of $Model$ could be $Civic$ or relevant to $Civic$. Note that the ranking of $\widehat{t_7}$ is the same as the ranking of Q_1, the ranking of $\widehat{t_4}$ is the same as Q_2, while the ranking of t_2 and t_8 need to be computed upon retrieval. Finally, the tuples from the extended result set are returned to the user in ranked order.

2.5 Explaining Answers to Users

Broad Challenges: Given that our query processor is a quasi-recommender system, it is important that its ranked results are trusted by the user. While rank explanation is critical enough in traditional databases (c.f. [8]), it is even

Query Results for query *Make* like honda and *Model* like civic and *Year* like 2001

Make	Model	Year	Price	Mileage	Location	Color	Relevant	Explanation
?	civic	2001	13490	58977	orange park	silver	☐	This car is **100%** likely to have **make=honda** given that its **model=civic**
honda	civic	2003	17490	16667	orlando	gray	☐	Cars having **year=2003** are **80%** similar to cars having **year=2001**
honda	accord	2001	15994	48123	atlanta	silver	☐	Cars having **model=accord** are **78%** similar to cars having **model=civic** given that **78%** of users who looked at **civic** also looked at **accord**
honda	?	2001	14995	32533	pasadena	black	☐	This car is **73%** likely to have **model=civic** given that its **make=honda, year=2001, and color=black**
honda	?	2001	15990	43137	glendale heights	silver	☐	This car is **32%** likely to have **model=accord** given that its **make=honda, year=2001, and color=silver** and **78%** of users who looked at **civic** also looked at **accord**

Next Query | Reset **Overall, were the explanations useful?** ○

Figure 2: Snapshot of Ranking Function Evaluation User Study.

more critical in the presence of query imprecision and data incompleteness. As a motivating example, consider a scenario where the user gives a selection query $Q_{Model \approx Civic}$, and one of the tuples returned by the system has a null value on model. Without an accompanying explanation, the user would be hard pressed to understand why this is a relevant answer for her query.

Current Implementation in QUIC: QUIC attempts to explain the relevance of its answers by providing *snippets* of its reasoning involving the assessed relevance and density functions. In the example above, the first level explanation may involve stating that the null value is 62% likely to be "Accord" (by the density function) which is 78% similar to "Civic" (by the relevance function). A further "drill down" is used to provide additional details on the explanation. For example, the density assessment could be justified in terms of the AFDs, and the relevance assessment can be justified in terms of co-click patterns.Figure 2 shows a snapshot of the explanations provided during our user study.

3. EMPIRICAL EVALUATION

We have implemented a prototype of QUIC, a screen shot is shown in Figure 2. Evaluating the effectiveness of QUIC is challenging due to the absence of a benchmark which defines relevance judgements for database queries. Therefore we have conducted small scale user studies for empirical evaluation. We performed two user studies with 10 people with none of them being familiar with this work. We use an online used car database *cars(make,model,year,price,mileage,location,color)* extracted from Yahoo! Autos(available at http://autos.yahoo.com).

Content	Co-click.	Co-occur.
Accord	Accord	Passport
Ram	Prelude	Odyssey
TL	Corolla	S2000
CR-V	Accent	Prelude
...	...	...

Table 3: Ranked similarity lists from the user study for Q: $\sigma_{Model \approx Civic}$.

Similarity Metric User Study: The first user study evaluates the effectiveness of content, co-click, co-occurrence based similarity metrics. Each user was shown 30 sets of lists. Each set contained a value for either a make or model of a car. In addition, each set had 3 ordered lists of similar makes/models, as shown in Table 3, which were produced using the content, co-click, and co-occurrence based approaches (the lists were randomly shuffled and the users were not given the methods used to generate the lists). Users were then asked to order the lists in terms of how good the ranked lists were to their own similarity relevance model. Table 4 shows the percentage of the make/model lists produced by each approach that users found to be most consistent with their own relevance model. As we can see, co-click filtering is the most effective approach for similarity assessment. Therefore, when the click streams are available,[2] we utilize the co-click based approach and choose content based similarity measurement otherwise.

Attribute	Co-click	Co-occur.	Content
Make	63.3%	29.3%	7.3%
Model	74.7%	11.3%	14.0%

Table 4: Evaluation of Relevance Assessment.

Ranking Order User Study: The second user study was used to evaluate the ranked results produced by QUIC. 14 multi-attribute selection queries on the car database are tested. For each query, the user was shown 10 tuples consisting of similar and incomplete tuples ranked based on Expected Relevance,[3] along with the explanations. Each tuple was then judged by the user as either relevant or not relevant.

To evaluate our ranking based on the $\mathcal{ER}$, we used normalized R-Metric [1] which is defined as: $\sum_i \frac{r_i}{2^{(\frac{i-1}{9})}}$ where i is the i^{th} ranked tuple, r_i is 1 if the user found the tuple relevant and 0 otherwise. The R-Metric computes the sum of the scores for the tuples which the user found to be relevant. As we can see, the formula gives more weight to the higher ranked tuples than the lower ranked ones.

Figure 3 shows the R-Metric scores for 4 types of queries where each type has the same set of constrained attributes. It shows that the $\mathcal{ER}$ ranking is effective in recommending highly relevant tuples, with an average R_{Metric} score above 0.75 for all 14 queries. In particular, for queries containing constrained attribute $Make$, the R_{Metric} scores are higher than the queries with constrained attributes $Model$ or $Year$. This is because the AFD for $Make$ has a higher confidence and corresponds with users' domain knowledge, making the incomplete tuples returned to be convincingly relevant. These experiments show that our ERR ranking model is effective in recommending relevant answers to the users. Furthermore, 41% of the time, the users found that the explanations offered by the system were useful.

Query Rewriting Evaluation: To show the effectiveness of our query rewriting techniques in retrieving highly rel-

[2]In the current implementation, click streams were only available for the attributes $Make$ and $Model$ hence content based similarity was used for the remaining attributes.

[3]We do not show tuples that exactly satisfy the user query because all such tuples are considered as equally relevant to the user.

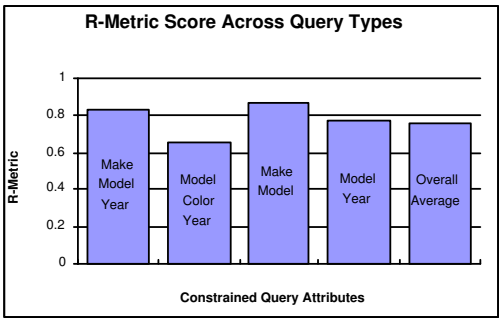

Figure 3: R-Metric Ranking Scores Across Queries

evant results, we compare the $\mathcal{ER}$ rank of each individual tuple in the extended query result set with the $\mathcal{ER}$ rank of the query that was used to retrieve this tuple, and count the number of rank inversions. For example, suppose rewritten query Q_1 is ranked higher than Q_2, we would expect the tuples retrieved by Q_1 to have a higher rank than those retrieved by Q_2. We count the number of tuples retrieved by Q_2 which actually have a higher rank than a tuple retrieved by Q_1. Then, we compute the number of inversions by dividing this number by the total number of tuples retrieved by query Q_2. Table 5 shows the percentage of inversions over 4 different query types with an average of 3.4% inversion. Thus, we can conclude that our query rewriting techniques indeed retrieve highly relevant tuples first by issuing the queries in order of their ERR rank.

Query Attributes	% of Inversions
Make, Model, Year	3.4%
Model, Color, Year	3.9%
Make, Model	2.9%
Model, Year	4.6%
Overall Average	**3.4%**

Table 5: Percentage of Rank Inversions from the query rankings to the individual tuple rankings.

In addition to the evaluation presented here, the empirical studies in [12, 15] can be seen as ablation studies on QUIC. In particular, the effectiveness of the AFD and NBC based density assessment is evaluated in [12] and the effectiveness of content-based relevance assessment is evaluated in [15].

4. CONCLUSION & PROSPECTUS

In this paper we motivated the problem of handling data incompleteness and query imprecision in autonomous databases. We argued that effectively tackling this problem requires solutions to density and relevance estimation, query rewriting and result explanation. We showed that solving these problems requires tools from decision theory, utility elicitation, statistical learning, as well as core database techniques for handling uncertainty and incompleteness. We then summarized our current progress in designing a system, QUIC, for handling these challenges. We ended by describing promising preliminary empirical studies on QUIC.

At the time of this writing, we are focusing on three important outstanding challenges: (i) handling attribute correlations in assessing relevance and density functions (ii) supporting more expressive queries, including joins and (iii) focusing on data-integration issues that arise in the context of multiple autonomous databases. The first issue arises when the database tuples contain multiple nulls and/or the user query is imprecise across multiple constrained attributes. In both cases, the density and relevance estimation techniques need to consider the dependencies between the attributes. A naive method would be to consider correlated attributes as single super-attributes. We are investigating assessment methods for relevance and density functions in structured representations such as Bayes Nets, conditional random fields and CP-nets [9, 16, 5, 17]. Supporting join queries brings forth several of the core issues in probabilistic databases (c.f. [3]), including the fact that the uncertainty can no longer be represented in terms of the so-called x-tuples, and requires c-tables [11]. We are investigating the adoption of lineage-based uncertainty handling (c.f . [3]). Finally, the presence of multiple autonomous databases poses the challenge of selecting sources that can provide tuples with the highest expected relevance with fewest resources. We are considering adapting our work on source selection (e.g. [14]) as well as collection selection (e.g. [7]).

Acknowledgements: We thank Aravind Kalavagattu, Zaiqing Nie, Anhai Doan, Alon Halevy, John Tangney and the CIDR reviewers for useful comments on earlier drafts.

5. REFERENCES

[1] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. *CIDR*, 2003.

[2] P. Andritsos, A. Fuxman and R.J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. ICDE 2006.

[3] O. Benjelloun, A.D. Sarma, A. Halevy, and J. Widom. ULDBs: Databases with Uncertainty and Lineage. VLDB 2006.

[4] J. Blythe. Visual exploration and incremental utility elicitation. *AAAI*, 2002.

[5] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research(JAIR)*, 2003.

[6] S. Chaudhuri, G. Das, V. Hristidis, G. Weikum: Probabilistic Ranking of Database Query Results. VLDB 2004.

[7] B. Chokshi, J. Dyer, T. Hernandez and S. Kambhampati. Relevance and Overlap Aware Text Collection Selection. ASU CSE TR 06-019. 2006.

[8] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD Conference*, 2006.

[9] T. Hastie, R. Tibshirani and J. Friedman. Elements of Statistical Learning. Springer Verlag (2001).

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

[11] T. Imielinski, and W. Lipski Jr. Incomplete Information in Relational Databases. In *J. ACM 31(4): 761-791*, 1984.

[12] H. Khatri, J. Fan, Y. Chen, and S. Kambhampati. QPIAD: Query processing over incomplete autonomous databases. In *Proc. ICDE*, 2007.

[13] A. Motro. Vague: A user interface to relational databases that permits vague queries. ACM TOIS 6(3), 1998.

[14] Z. Nie, S. Kambhampati: A Frequency-based Approach for Mining Coverage Statistics in Data Integration. ICDE 2004.

[15] U. Nambiar and S. Kambhampati. Answering imprecise queries over autonomous web databases. In *ICDE*, 2006.

[16] R.E. Neapolitan. *Learning Bayesian Networks.* Prentice Hall. 2004.

[17] R. Gupta and S. Sarawagi. Creating Probabilistic Databases from Information Extraction Models. VLDB 2006.

[18] D. Suciu and N. Dalvi. Foundations of Probabilistic Answers to Queries. In *Proc. SIGMOD*. 2005.