SMART-INT: A SYSTEM FOR ANSWERING QUERIES OVER WEB DATABASES USING ATTRIBUTE DEPENDENCIES

by

Anupam Khulbe

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2009

SMART-INT: A SYSTEM FOR ANSWERING QUERIES OVER WEB DATABASES USING ATTRIBUTE DEPENDENCIES

by

Anupam Khulbe

has been approved

August 2009

ABSTRACT

Many web databases can be seen as providing partial and overlapping information about entities in the world. To answer queries effectively, it is required to integrate the information about the individual entities that are fragmented over multiple tables. At first blush this is just the inverse of a traditional database normalization problem - the universal relation is to be reconstructed from the given tables (sources). However, the tables maybe missing Primary Key - Foreign Key relations, which leads to technical challenges. It is clear that reconstruction and retrieval of relevant entities will have to involve joining the tables. While tables do share attributes, direct joins over these shared attributes can result in reconstruction of many spurious entities thus seriously compromising precision. This work is aimed at addressing the problem of data integration in such scenarios. To make-up for the missing primary key-foreign key relations, this approach mines and uses attribute dependencies. Such dependencies can both be intra-table and inter-table. Given a query, these dependencies are used to piece together a tree of relevant tables and schemes for joining them. For experimental evaluation, a master table was fragmented horizontally and vertically to generate a set of overlapping tables. A set of randomly generated queries were targeted at both the master tables and the fragmented tables. The result from the master tables acted as the ground truth, against which the result from this approach was evaluated, in terms of precision and recall. Approaches like answering from a single table, and direct joins were employed as benchmarks. The result tuples produced by this approach are demonstrated to be able to strike a favorable balance between precision and recall.

ACKNOWLEDGMENTS

I am thankful to Dr. Subbarao Kambhampati for his guidance and immense patience as an advisor. The in-depth technical discussions with him were always engaging, and his enthusiam was a great source of motivation. I am thankful to Dr. Pat Langley for his guidance and the freedom he provided in terms of direction of this work. His suggestions and perspective were instrumental in shaping the idea. I am thankful to Dr. Jieping Ye for being in the thesis committee.

It has been a great experience to know Ravi Gummadi as a colleague as well as a friend. I am thankful to him for the countless hours we spent on technical discussions and working together.

I am thankful to Aravind Kalvagattu for his suggestions and feedback on the machine learning aspect of this work. Furthermore, I appreciate his guidance towards research in general and sharing his own research experiences.

I am grateful to Raju Balakrishnan for the contructive criticism provided during the research meetings and in-person discussions.

Last, but not the least, I thank Garrett Wolf for his suggestions and comments during the initial phases of this work.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

With the advent of web-oriented computing, databases have now assumed a huge web-presence. There is a plethora of autonomous databases, like Google Base and Amazon SimpleDB, readily accessible on the web. Users all over the world update these data-sources in a highly decentralized and independent manner. As a downside to this relaxed administration scheme, these databases present some challenges which render the traditional query answering approaches unsuitable. In particular, these unsupervised updates to the databases effectively lead to fragmentation of data across the web, where primary-key foreign-key relationships are typically missing across these fragmented chunks, which makes SQL join-based approaches suffer from severe performance penalties. While working in collaboration with Ravi Gummadi, my colleague at DBYochan lab, I developed a novel solution, SmartInt, for answering queries over web-databases which are missing primary-key foreign-key relationships.

Given the web-scale of these databases, an effective source selection strategy is needed to have a scalable query-answering performance. I was mainly interested in creating an effective source selection strategy in the context of this novel problem. My primary contribution towards this work was formulating a source selection problem in this context and devising a solution mechanism for it. One of the novel features of this source selection approach is selection of a tree of tables, which seemlessly handles the order in which constraints get evaluated and attributes get integrated.

Apart from devising the solution framework along with Gummadi, and developing a solution for source selection, I have contributed towards developing the learning module for the SmartInt system. A detailed description of individual contributions is available later in chapter 9.

2 MOTIVATION

With the advent of web, data available online is rapidly increasing, and an increasing portion of that data corresponds to large number of web databases populated by web users. Web databases can be viewed as providing partial but overlapping information about entities in the world. Conceptually, each entity can be seen as being fully described by a universal relation comprising of all its attributes. Individual sources can be seen as exporting parts of this universal relation. This picture looks very similar to the traditional database set-up where a universal relation is normalized into smaller relations for effective storage. The materialized web data sources can thus be viewed as an *ad hoc* normalization of the universal relation, and the integration of sources involves reconstructing the universal relation and running queries on it. There are however two important complications:

- The Primary Key-Foreign Key relationships that are crucial for reconstructing the universal relation are often missing from the tables. This is in part because partial information about the entities are independently entered into different tables, and synthetic keys (such as vehicle ids, model ids, employee ids) are simply ignored.[1]
- Most users accessing these tables are lay users and are not often aware of all the attributes of the universal relation. Thus their queries may be "imprecise" [15] in that they may miss requesting some of the relevant attributes about the entities under consideration.

Thus a core part of the source integration on the web can be cast as the problem of reconstructing the universal relation in the absence of primary key-foreign key relations, and in the presence of lay users (In practice, this reconstruction problem is buried under

[1]In some cases, such as public data sources about people, the tables may even be explicitly forced to avoid keeping such key information.

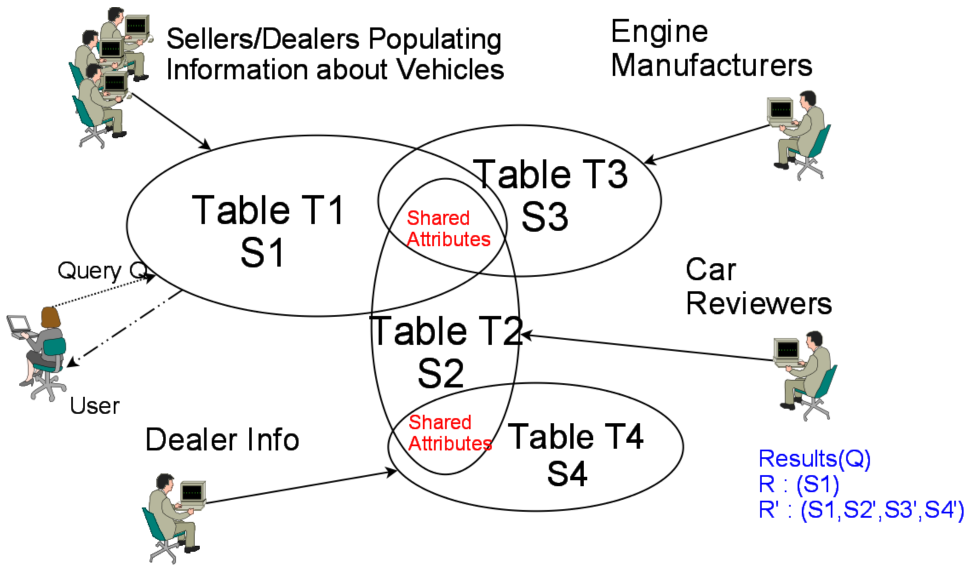

Fig. 1. Overlapping Tables in the Database

the more immediate problem of schema heterogeneity, as in addition to the loss of PK-FK information, different tables tend to rename their columns. Thus, a more accurate generative model for web data sources is that they are the result of an *ad hoc* normalization followed by the attribute name change. Many effective solutions, such as SIMIFLOOD [13] do exist for computing attribute mappings to handle the name change problem. Rather than revisit that problem, in this paper we will simply assume that attribute name change problem has been addressed by one of those methods. This allows us to focus on the central problem of reconstruction of universal relation in the absence of primary key-foreign key relationships.)

Traditional data integration techniques (such as GAV/LAV) that rely on manually constructed schema mappings are not practical both because of the difficulty of finding experts able to construct mappings, and because of the difficulty of constructing sound and complete mappings in the absence of PK-FK relationships (see Section 4).

Our aim is to provide a fully automated solution to this problem. Let us illustrate the challenges involved in this enterprise through a motivating example. Consider a set of tables populated in a *Vehicle* domain (Figure 1). In the example, there are multiple tables with different schemas containing information about the vehicle domain. The universal schema of entity 'Vehicle' is described as follows:

Vehicle (VIN, vehicle-type, location, year, door-count, model, make, review, airbags, brakes, year, condition, price, color, engine, cylinders, capacity, power, dealer, dealer-address)

Let us assume that the database has the following tables: *Table I with Schema S1* - populated by normal web users who sell and buy cars, *Table II with Schema S2* - populated by crawling reviews of different vehicles from websites, *Table III with Schema S3* - populated by engine manufacturers/vendors with specific details about vehicle engines and *Table IV with Schema S4*. The following shows the schema for these tables and the corresponding schema mappings among them.

S1 (make, model_name, year, condition, color, mileage, price, location, phone)

S2 (model, year, vehicle-type, body-style, door-count, airbags, brakes, review, dealer)

S3 (engine, mdl, cylinders, capacity, power)

S4 (dealer, dealer-address, car-models)

The following attribute mappings are present among the schemas:

(S1: model_name = S2: model = S3: mdl, S2: dealer = S4: dealer) The italicized attribute *MID (Model ID)* refers to a *synthetic primary key* which would have been present if the users had complete knowledge about the entity which they are populating. If it is

present, entity completion becomes trivial because you can simply use that attribute to join the tables, but there can be a variety of reasons why that attribute is not available:

- In autonomous databases, users populating the data are not aware of all the attributes and may end up missing the 'key' information.
- Since each table is autonomously populated, though each table has a key, it might not be a shared attribute which makes join infeasible.
- The same problem can arise when primary key is intentionally masked, since it describes sensitive information about the entity(Social Security Number etc).

TABLE I
SCHEMA 1 - CARS(S_1)

MID	Make	Model_name	Price	Other Attrbs
HACC96	Honda	Accord	19000	...
HACV08	Honda	Civic	12000	...
TYCRY08	Toyota	Camry	14500	...
TYCRA09	Toyota	Corolla	14500	...

TABLE II
SCHEMA 2 - REVIEWS (S_2)

Model	Review	Vehicle-type	Dealer	Other Attrb
Corolla	Excellent	Midsize	Frank	...
Accord	Good	Fullsize	Frank	...
Highlander	Average	SUV	John	...
Camry	Excellent	Fullsize	Steven	...
Civic	Very Good	Midsize	Frank	...

Suppose the user is interested in the following query: **Give me 'Make', 'Model' of all vehicles whose price is less than $15000 and which have 4-cylinder engine.** The above

TABLE III
SCHEMA 3 - ENGINE (S_3)

MID	Mdl	Engine	Cylinders	Other Attrb
HACC96	Accord	K24A4	6	...
TYCRA08	Corolla	F23A1	4	...
TYCRA09	Corolla	155 hp	4	...
TYCRY09	Camry	2AZ-FE I4	6	...
HACV08	Civic	F23A1	4	...
HACV07	Civic	J27B1	4	...

TABLE IV
SCHEMA 4 - DEALER INFO (S_4)

Dealer	Address	Other Attrb
Frank	1011 E Lemon St, Scottsdale, AZ	...
Steven	601 Apache Blvd, Glendale, AZ	...
John	900 10th Street, Tucson, AZ	...

query would translate to the following SQL notation. We use this example throughout the paper to illustrate working of different modules of the system:

```
SELECT make,model WHERE price < $15000 AND cylinders = '4'.
```

The query here is "partial" in that it does not specify the exact tables over which the query is to be run. Part of the challenge is to fill-in that information. Let us examine two obvious approaches to answer the query on this database:

- **Answering from a single table**: The first approach is to answer the query from one table which conforms to the most number of constraints mentioned in the query and provides maximum number of attributes. In the given query since 'make', 'model' and 'price' map onto Table I, we can directly query that table by ignoring constraint on the 'cylinders'. The resulting tuples are shown in Table V. The second tuple related to 'Camry' has 6 cylinders and is shown as an answer. Hence ignoring con-

TABLE V
RESULTS OF QUERY Q FROM TABLE T_1

Make	Model	Price
Honda	Civic	12000
Toyota	Camry	14500
Toyota	Corolla	14500

TABLE VI
RESULTS OF QUERY Q USING DIRECT-JOIN ($T1 \bowtie T3$)

Make	Model	Price	Cylinder	Engine	Other attrbs
Honda	Civic	12000	4	F23A1	...
Honda	Civic	12000	4	J27B1	...
Toyota	Corolla	14500	4	F23A1	...
Toyota	Corolla	14500	4	155 hp	...

straints would lead to erroneous tuples in the final result set which do not conform to the constraints.

- **Direct Join**: The second and a seemingly more reasonable approach is joining the tables using whatever shared attribute(s). The result of doing a direct join based on the shared attribute('model') is shown in Table VI. If we look at the results, we can see that even though there is only one 'Civic' in Table I, we have two Civics in the final results. The same happens for 'Corolla' as well. These errors are due to absence of Primary Key - Foreign Key relationship between these two tables. This leads to spurios results.

Apart from the limitations discussed above, these approaches fail to get other attributes which describe the entity. In such a scenario, providing the complete information about the entity to users requires:

TABLE VII
RESULTS OF QUERY Q USING ATTRIBUTE DEPENDENCIES

Make	Model	Price	Cylinders	Review	Dealer	Address
Honda	Civic	12000	4	Very Good	Frank	1011 E St
Toyota	Corolla	14500	4	Excellent	Frank	1011 E St

- **Linking attributes and propagating constraints spanning across multiple tables, and retrieving precise results.**
- **Increasing the completeness of the individual results by retrieving additional relevant attributes and their associated values from other overlapping tables not specified in the query(thereby reconstructing the universal relation from different local schemas).**

Addressing these two needs poses serious challenges. In the absence of information on how the tables overlap, it is not possible to link the attributes across tables. We can find the mappings between attributes using algorithms like Similarity Flooding [13]. However, these alone would not be enough. Since attribute mappings between tables still leaves the problem of absence of key information open, the usual process of getting results through direct join would result in very low precision. Moreover discovering additional attributes related to those mentioned in the query requires knowledge of attribute dependencies, which are not apparent.

3 SmartInt SYSTEM

In the previous chapter we discussed in detail about the problem of missing key information in Web databases and why other approaches fail. This chapter describes about the SmartInt system and how different components work at higher level. It also enlists the contributions of this thesis.

3.1 System Description

Our approach for addressing these challenges involves starting with a base table containing a subset of query-relevant attributes, and attempting to "complete" the tuples by predicting the values of the remaining relevant attributes. The prediction/completion of the tuples is made possible by approximate functional dependencies, which are automatically mined from samples of individual tables. The selection of base table itself is influenced by the confidences of the available AFDs. Intuitively, the base table should contain important attributes for whose values cannot be predicted accurately, but which can help in predicting other attributes. Our base table selection step formalizes this intuition in terms of the confidence of the available AFDs. As a simple illustration of the idea, suppose the following simple AFDs are mined from our tables (note that the actual mined AFDs can have multiple attributes on the left hand side): (1) $S_2 : \{model\} \rightarrow vehicle\_type$, (2) $S_2 : \{model\} \rightarrow review$, (3) $S_3 : \{model\} \rightarrow cylinders$. Rule 3 provides us information about the number of cylinders which helps in conforming the results to the '4 cylinder' constraint. Rules 1 & 2 provide information on vehicle type and review for a given model, and hence provide more information in response to the query. They allow us to expand partial information about the car model into more complete information about vehicle type, review and cylinders. The results using attribute dependencies are shown in Table VII and conform to the constraints and are more informative compared to other approaches.

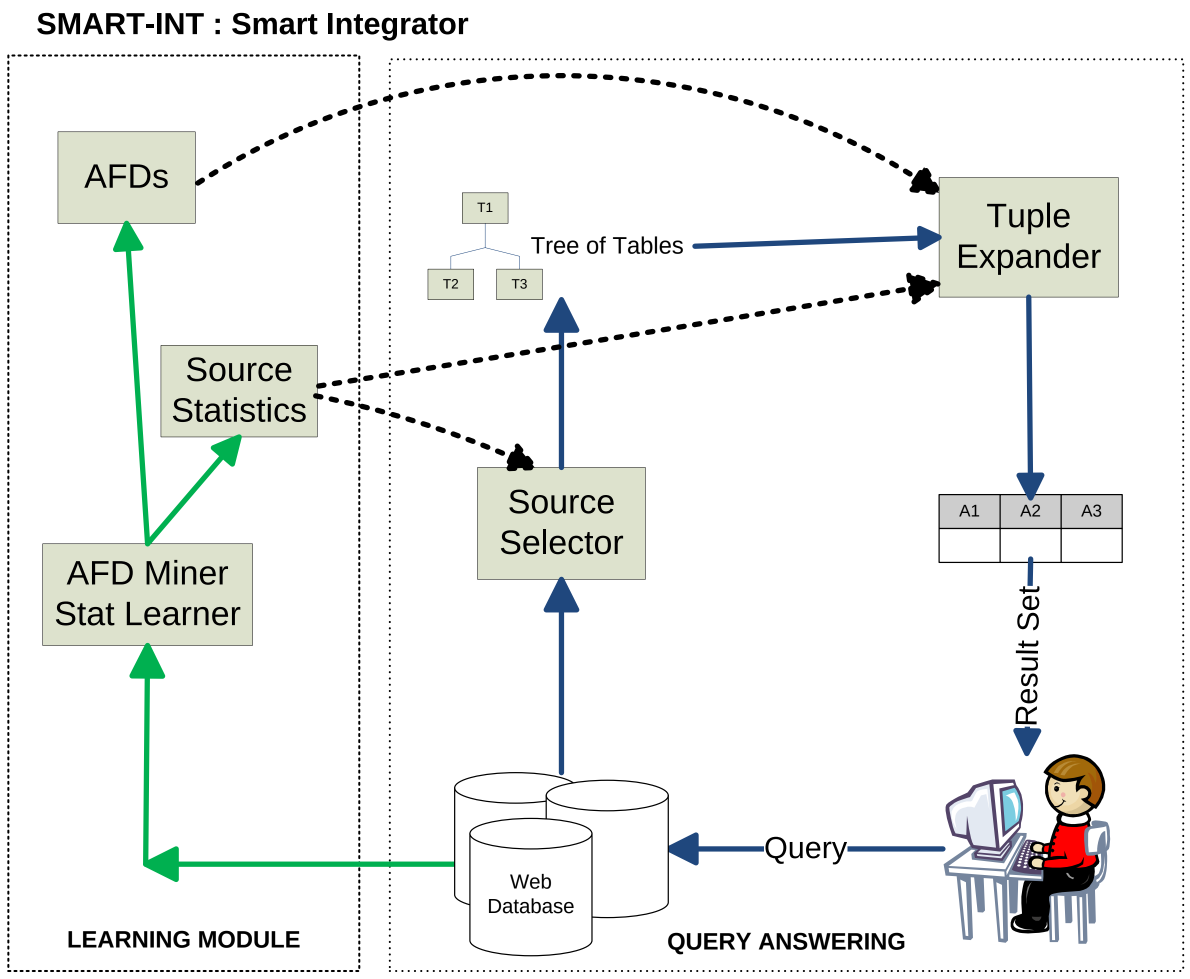

Fig. 2. Architecture of SMARTINT System

Conceptually, the operation of SMARTINT can thus be understood in terms of (i) mining AFDs and source statistics from different tables and (ii) actively using them to propagate constraints and retrieve attributes from other non-joinable tables. Figure 2 shows the SMARTINT system architecture. In this framework a user issues a query on the web databases. The system has two distinct components, query answering and learning. Query answering comes into picture immediately after the user issues a query. When user submits a query, the source selector first selects the most relevant 'tree' of tables from the available set of tables. Source selector uses the source statistics mined from the tables to pick the tree of tables. The tuple expander module operates on the tree of tables provided by the source selector and then generates the final result set. Tuple expander first constructs the expanded schema using the AFDs learned by AFDMiner and then populates the values in the schema using source statistics. As shown in the Figure 2, SMARTINT learns AFDs and source statistics from individual tables and stores them which are used by source selector and tuple expander. In the Section 6 and Section 7, we discuss in detail how the two modules, Query Answering and Learning work.

3.2 CONTRIBUTIONS

The specific contributions of SMARTINT system can be summarized as follows:

1. We have developed a query answering mechanism that utilizes attribute dependencies to recover entities fragmented over tables, even in the absence of primary key–foreign key relations.

2. We have developed a source selection method using novel relevance metrics that exploit the automatically mined AFDs to pick the most appropriate set of tables.

3. We have developed techniques to efficiently mine approximate attribute dependencies.

The rest of the paper is organized as follows. Section 4 discusses related work about current approaches for query answering over web databases. Section 5 discusses some preliminaries. Section 6 proposes a model for source selection and query answering using attribute dependencies. Section 7 provides details about the methods for learning attribute dependencies. Section 8 presents a comprehensive empirical evaluation of our approach. Section 9 provides conclusion and future work.

4 RELATED WORK

4.1 Data Integration

The standard approaches investigated in the database community for the problem recovering information split across multiple tables is of course data integration [12, 10]. The approaches involve defining a global (or mediator) schema that contains all attributes of relevance, and establishing mappings between the global schema and the source schemas. This latter step can be done either by defining global schema relations as views on source relations (called GAV approach), or defining the source relations as views on the global schema (called LAV approach). Once such mappings are provided, queries on the global schema can be reformulated as queries on the source schemas. While this methodology looks like a hand-in-glove solution to our problem, its impracticality lies in the fact that it requires manally established mappings between global and source schemas. This is infeasible in our context where lay users may not even know the set of available tables, and even if they do, the absence of PK-FK relations makes establishment of non-lossy (sound and complete) mappings impossible. In contrast, our approach is a fully automated solution that does not depend on the availability of GAV/LAV mappings.

4.2 Keyword Search on Databases

There has been considerable research on keyword search over databases [4] [3] [1] [2]. The work on Kite system extends keyword search to multiple databases as well [16]. While Kite doesn't assume that PK-FK relations are pre-declared, it nevertheless assumes that the columns corresponding to PK-FK relations are physically present in the different tables if only under different names. In the context of our running example, Kite would assume that the model id column is present in the tables, but not explicitly declared as a PK-FK relation. Thus Kite focuses on identifying the rekevant PK-FK columns using key discovery techniques (c.f. [5]). Their techniques do not work in the scenarios we consider where the key columns are simply absent (as we have argued in our motivating example).

4.3 Learning Attribute Dependencies

Though rule mining is popular in the database community, the problem of AFD mining is largely under explored. Earlier attempts were made to define AFDs as an approximation to FDs ([6], [5]) with few error tuples failing to satisfy the dependency. In these lines, CORDs [6] introduced the notion of Soft-FDs. But, the major shortcoming of their approach is, they are restricted to rules of the type C1→C2, where C1 and C2 are only singleton sets of attributes. TANE [5] provides and efficient algorithm to mine FDs and also talks about a variant of it to learn AFDs. But, their approach is restricted to minimal pass rules (Once an AFD of type ($X \rightsquigarrow Y$) is learnt, the search process stops without generating AFDs of the type ($Z \rightsquigarrow Y$), where $X \subset Z$. Moreover, these techniques are restricted to a single table, but we are interested in learning AFDs from multiple tables and AFDs involving shared attributes. In this paper, we provide a learning technique that treats AFDs as a condensed representation of association rules (and not just approximations to FDs), define appropriate metrics, and develop efficient algorithms to learn all the intra and inter-table dependencies. This unified learning approach has an added advantage of computing all the interesting association rules as well as the AFDs in a single run.

4.4 Querying Incomplete Databases

Given a query involving multiple attributes, SmartInt starts with a base table containing a subset of them, and for each of the tuples in the base table, aims to predict the remaining query attributes. In this sense it is related to systems such as QPIAD [18]. However, unlike QPIAD which uses AFDs learned from a single table to complete null-valued tuples, SMARTINT attempts to extend the base table by predicting entire columns using AFDs learned from other tables. Viewed this way, the critical challenge in SMARTINT is the selection of base table, which in turn is based on the confidences of the mined AFDs (see Section 6.A).

5 PRELIMINARIES

This section describes different aspects of our query answering system: Semantics of the query, attribute dependencies and the architecture of the system.

5.1 QUERY

Our system assumes that the user does not have knowledge about different tables in the database and has limited knowledge about attributes he is interested in querying (This is a reasonable assumption, since most web databases do not expose the tables to the users). So we model the query in the following form where the user just needs to specify the attributes and constraints:

Definition 5.1.1 (Query). $\mathcal{Q} =< \bar{A}, \bar{C} >$ *where* $\bar{A}$ *are the projected attributes which are of interest to the user and* $\bar{C}$ *are the set of constraints (i.e. attribute-label, value pairs)*

5.2 ATTRIBUTE DEPENDENCIES

In section 2, we mentioned that we use attribute dependencies in our query answering mechanism. Here we describe the details about the particular kind of attribute dependencies we use. Attribute dependencies are represented in the form of **Functional Dependency**. A **functional dependency (FD)** is a constraint between two sets of attributes in a relation from a database. Given a relation R, a set of attributes X in R is said to functionally determine another attribute Y, also in R, (written X $\rightarrow$ Y) if and only if each X value is associated with precisely one Y value. Since the real world data is often noisy and incomplete, we use approximate dependencies to represent the attribute dependencies.

An **Approximate Functional Dependency (AFD)** is an approximate determination of the form $X \rightsquigarrow A$ over relation R, which implies that **attribute set X, known as the determining set**, approximately determines **A, called the determined attribute.** An AFD

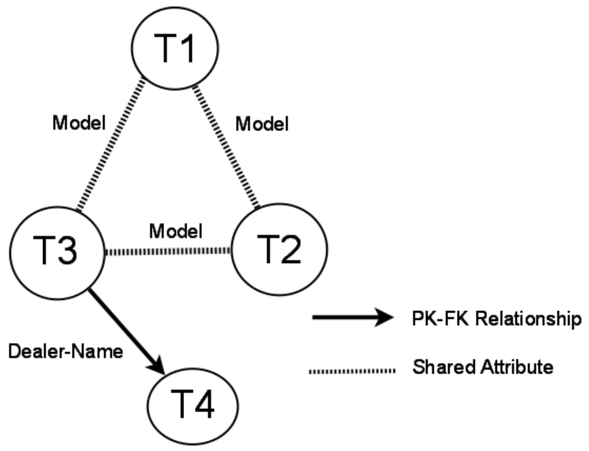

Fig. 3. Graph Representation of Connected Tables in the Database.

is a functional dependency that holds on all but a small fraction of tuples. For example, an AFD $model \rightsquigarrow body\_style$ indicates that the value of a car model usually (but not always) determines the value of body_style.

5.3 Graph of Tables

The inter-connections between different tables in the database are modelled as a graph(refer to Figure 3). Each attribute match is represented as an undirected edge and any PK-FK relationship is represented as a directed edge pointing towards the table containing the primary key. This model would help in using the PK-FK relationships available—either as part of the input, or can be identified automatically (as done in KITE [16]). When neither is feasible, we rely on tuple completion.

6 QUERY ANSWERING

In this section, we provide an overview of our query answering approach and its detailed description. We assume that attribute dependencies are provided upfront for the system and describe how our work uses them to answer the queries over multiple tables. We outline our approach in terms of solutions to challenges identified earlier in Section 2:

1. Information distributed across tables needs to be integrated: The information needs to be integrated since both answering queries with attributes spanning over multiple tables and providing additional information to the user needs horizontal integration of the tuples across tables. In the absence of PK-FK relationships, performing meaningful joins to integrate data is not feasible (as illustrated in Section 2). Instead we start with a 'base set of tuples' (from a designated base table chosen by the source selector) and successively expand those tuples horizontally by appending attribute values predicted by the attribute dependencies. This expansion is done recursively till the system cannot chain further or it reconstructs the universal relation. We use attribute determinations along with attribute mappings to identify attributes available in other tables, whose values can be predicted using values of the selected attributes.

2. Constraints need to be translated: The base table provides a set of tuples, i.e. tuples which conform to the query constraints. Generation of 'base set of tuples' requires taking into account constraints on non-base tables. We use attribute mappings and attribute determinations for translating constraints onto the base table. Basically, we need to translate the constraint on a non-base table attribute to a base table attribute through attribute determinations. In the example discussed in Section 2, suppose T_1 is designated as a base table and T_3 is a non-base table which has an AFD (model $\rightsquigarrow$ vehicle-type). If the query constrains the attribute vehicle-type to be 'SUV', then this

constraint can be evaluated over the base table, if information about the likelihood of a model being an 'SUV' is given. Attribute determinations provide that information.

Now we explain how these solution approaches are embedded into SMARTINT framework. Query answering mechanism involves two main stages: Source Selection and Tuple Expansion. We explain these in detail in the next few sections.

6.1 SOURCE SELECTION

In a realistic setting, data is expected to be scattered across a large number of tables, and not all the tables would be equally relevant to the query. Hence, we require a source selection strategy aimed at selecting the top few tables most relevant to the query. Given our model of query answering, where we start with a set of tuples from the base table which are then successively expanded, it makes intuitive sense for tuple expansion to operate over a tree of tables. Therefore source selection aims at returning the most relevant tree of tables over which the Tuple Expander operates.

Given a user query, $\mathcal{Q} =< \bar{A}, \bar{C} >$ and a parameter 'k' (the number of relevant tables to be retrieved and examined for tuple expansion process), we define source selection as selecting a tree of tables of maximum size k which has the highest relevance to the query. The source selection mechanism goes through the following steps:

1. Generate a set of candidate tables T_c.

$$T_c = \{\mathcal{T} \in T | relevance(\mathcal{T}) \geq threshold\}$$

   This acts as a pruning stage, where tables with low relevance are removed from further consideration.

2. As observed in Figure 3, not all tables have a shared attribute. We need to pick a connected sub-graph of tables, G_c, with highest relevance.

3. Select the tree with the highest relevance, among all the trees possible in G_c. This step involves generating and comparing the trees in G_c, which can be computationally expensive if G_c is large. We heuristically to estimate the best tree with the highest relevance to the query among all the trees. The relevance metrics used are explained below.

We will explain how source selection works in the context of the example described in introduction. In order to answer the query Q, `SELECT make,model WHERE price <`
`$15000 AND cylinders = '4'`, we can observe that the projected attributes $make$, $model$ and constraint $price <$ \$15000 are present in Table I and constraint $cylinders =$ '$4'$ is present in Table III. Given this simple scenario, we can select either Table I or Table III as the base table. If we select Table III as the base table, we should translate the constraint $price <$ \$15000 from Table I to Table III using the AFD, $model \rightsquigarrow price$. On the other hand if we designate Table I as base table, we would need to translate the constraint $cylinders =$ '$4'$ from Table III to Table I using the AFD, $model \rightsquigarrow cylinders$. Intuitively we can observe that the AFD $model \rightsquigarrow cylinders$ generalizes well for a larger number of tuples than $model \rightsquigarrow price$. Source selection tries to select the table which emanates high quality AFDs as the base table and hence yield more precise results.

Here we discuss the different *relevance functions* employed by the source selection stage:

- **Relevance of a table:** Ideally the relevance of a table $\mathcal{T}$ has to be calculated by summing the relevance over each tuple. But evaluating the relevance of each tuple during query time is not feasible. Hence we approximate it as

$$relevance(\mathcal{T}, q) \approx P_{\mathcal{T}}(\bar{C}) * tupleCount_{\mathcal{T}} * \frac{|A_{\mathcal{T}} \cap \bar{A}|}{|\bar{A}|}$$

where $Pr_{\mathcal{T}}(\bar{C})$ is the probability that a random tuple from $\mathcal{T}$ conforms to constraints $\bar{C}$, $tupleCount_{\mathcal{T}}$ is the number of tuples in $\mathcal{T}$, and $A_{\mathcal{T}}$ is the set of attributes in $\mathcal{T}$. The fraction measures the ratio of query attributes provided by the table.[1]

- **Relevance of a tree:** While selecting the tree of relevant tables, the source selection stage needs to estimate the relevance of tree. As we have explained above, the relevance of tree takes into account the confidence of AFDs emanating out of the table. So the relevance function should also capture the same intuition. Relevance of a tree T_r rooted at table $\mathcal{T}$ w.r.t query $Q < \bar{A}, \bar{C} >$ can be expressed as:

  $$relevance(T_r, q) = relevance(\mathcal{T}, q) + \sum_{a \in \bar{A} - A_b} pred\_accuracy(a)$$

  where A_b are the set of attributes present in the base table, $pred\_accuracy(a)$ gives the accuracy with which the attribute a can be predicted. When the attribute is in the neighbouring table it is equal to the confidence of AFD and when its not in the immediate neighbour its calculated the same way as in AFD chanining (Explained in Section 7).

The above relevance functions rely on the conformance probability $P_{\mathcal{T}}(C) = \Pi_i P_{\mathcal{T}}(C_i)$. $P_{\mathcal{T}}(C_i)$ denotes the probability that a random tuple from $\mathcal{T}$ conforms to the constraint C_i (of the form $X = v$), and is estimated as:

[1]Presently we give equal weight to all the attributes in the system, this can be generalized to account for attributes with different levels of importance.

- $P_{\mathcal{T}}(C_i) = P_{\mathcal{T}}(X = v), if X \in A_{\mathcal{T}}$, where $A_{\mathcal{T}}$ is the set of attributes in $\mathcal{T}$
- $P_{\mathcal{T}}(C_i) = \sum_i P_{\mathcal{T}}(Y = v_i) * P_{\mathcal{T}'}(X = v|Z = v_i)$, if $\mathcal{T} : Y = \mathcal{T}' : Z$, i.e. $\mathcal{T}$'s neighboring table $\mathcal{T}'$ provides attribute X.[2]
- $P_{\mathcal{T}}(C_i) = \epsilon$ (small non-zero probability), otherwise

Algorithm 1 Source Selection

```
Require: Query q, Threshold $\tau$, Number of tables k, Set of AFDs $\bar{A}$
 1: $T_c$ = $\{\emptyset\}$
 2: for all table $\mathcal{T}$ in T do
 3:     if relevance($\mathcal{T}$, q) $\geq \tau$ then
 4:         add $\mathcal{T}$ to $T_c$
 5: $G_c$ := Set of connected graphs over $T_c$ up to size k
 6: $Trees$ = $\{\emptyset\}$
 7: for all $g \in G_c$ do
 8:     $Trees_g$ = Set of trees from graph g
 9:     add $Trees_g$ to $Trees$
10: $tree_{sel} = \arg\max_{tree \in Trees} relevance(tree, q)$
11: return $tree_{sel}$
```

In this section we explained the source selection mechanism. We discuss how the tuple expansion mechanism answers the query from the selected sources in the next section.

6.2 Tuple Expansion

Source selection module gives a tree of tables which is most relevant to the query. Tuple expansion operates on the tree of tables given by that module. One of the key contributions of our work is returning the result tuples with schema as close to the universal relation as possible. We need to first construct the schema for the final result set and then populate tuples that correspond to that particular schema from other tables. These steps are described in detail in the sections that follow.

[2]These probabilities are learnt as source statistics.

6.2.1 Constructing the Schema

One important aspect of tuple expansion is that it is a hierarchical expansion. The schema grows in the form of a tree because attributes retrieved from other tables are relevant only to the determining attribute(s)(refer to the definition of AFD in Section 5). This module returns a hierarchical list of attributes, *AttrbTree*, rather than a flat list. This is more clearly illustrated by the attribute tree generated for query discussed in Section 2 shown in Figure 4. The base table (T_1) contains attributes *Make, Model, Price*. Tables T_1, T_2 and T_3 share the attribute *Model*. In table T_2, we have the AFDs *Model* $\rightsquigarrow$ *Cylinders* and *Model* $\rightsquigarrow$ *Engine*. These two determined attributes are added to the base answer set, but these are only relevant to the attribute 'model', so they form a branch under the attribute 'model'. Similarly, *review, dealer* and *vehicle type* form another branch under 'Model'. In the next level, T_3 and T_4 share 'dealer-name' attribute. 'Dealer-Name' is a key in T_4, therefore all the attributes in T_4 ('dealer-address', 'phone-number' etc) are attached to the *AttrbTree*. The final attribute tree is shown in the Figure 4. Algorithm 2 gives formal description of the schema construction algorithm.

Algorithm 2 Construct Attribute Tree

```
Require: Source-table-tree $\mathcal{S}$, Set of AFDs $\bar{A}$;
1: $AttrbTreeA_t$ := $\{\emptyset\}$
2: $S_0$ := Get all the attributes from base table $T_b$
3: while $\mathcal{S}$ has children do
4:     $\mathcal{D}$ := Get all determined attributes in current child $c$
5:     add child $\mathcal{D}$ to $A_t$ for corresponding attribute
6: return $AttrbTreeT$
```

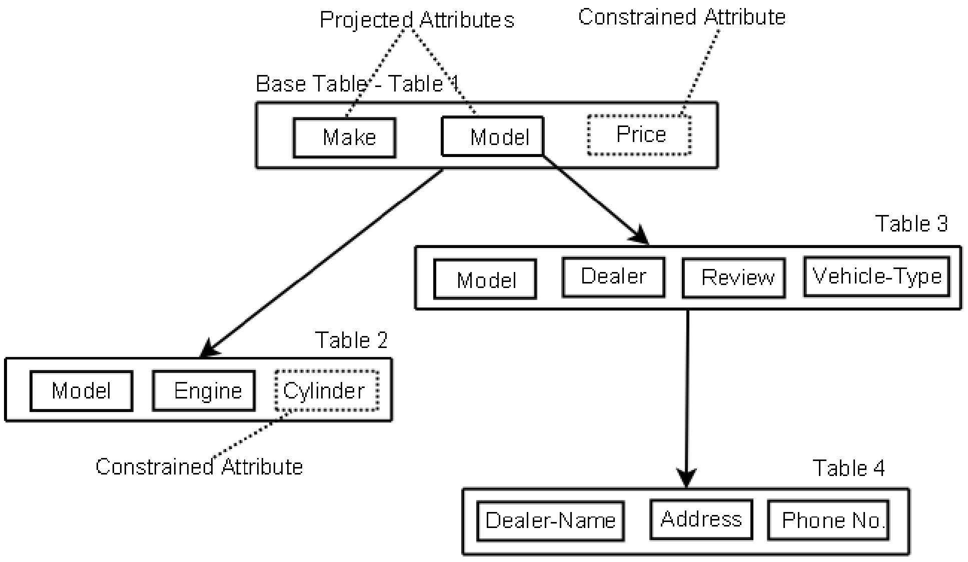

Fig. 4. Expanded Attribute Tree for the Query

6.2.2 POPULATING THE TUPLES

The root of the selected tree of tables given by the source selection is designated as the *base table*. Once the attribute hierarchy is constructed, the system generates a 'base set' of tuples from the base table which form the 'seed' answers. We refer to this base set as the *most likely tuples* in the base table which conform to the constraints mentioned in the query. We call them 'most likely' tuples because when constraints are specified on one of the children of the base table, we propagate constraints from child to base table. But since we have approximate dependencies between attributes, the translated constraints do not always hold on the base set. To clearly illustrate this, let us revisit the example of *Vehicle* domain from Section 2. We assume that Table I has been designated as the base table. The constraint $price <$ \$15000 is local for the base table and hence each tuple can be readily evaluated for conformance. The constraint *cylinders = '4'*, on the other hand, is over Table III and needs to be translated on to the base table. Notice that these two tables share the attribute 'model' and this attribute can approximately determine *cylinder* in Table III ($model \rightsquigarrow Cylinders$

). ($model \leadsto Cylinders$) implies that the likelihood of a model having certain number of cylinders can be estimated, which can be used to estimate the probability that a tuple in Table 1 would conform to the constraint $Cylinders =$ '$4'$. We can see that model 'Civic' is more likely to be in the base set than 'Accord' or 'Camry'.

Once the base tuple set has been generated, each of those tuples are expanded horizontally by predicting the values for the attributes pulled from children tables. Given a tuple from the base set, all the children tables (to the base table) are looked up for determined attributes, and the most likely value is used to expand the tuple. Further, values picked from the children tables are used to pick determined attributes from their children tables and so on. In this way, the base tuple set provided by the root table is expanded using the learned value dependencies from child tables.

In tuple expansion, if the number of shared attributes between tables is greater than one, getting the associated values from other tables would be an interesting challenge. For instance, in our running example, Table I also had the year attribute and Table II is selected as the base table. We need to predict the value of price from Table I. If we consider both Model and Year to predict the price, results would be more accurate, but we do not have the values of all combinations of Model and Year in Table I to predict the price. However, if we just use Model to predict the price, the precision might go down. Another interesting scenario where taking multiple attributes might not boost the prediction accuracy is the following: *Model, Number_tyres* $\leadsto$ *Price* is no better than *Model* $\leadsto$ *Price*. In order to counter this problem, we propose a *fall back* approach of the AFDs to ensure high precision and recall.

This method can be formally described as this: If $\mathcal{X}$ is the set of shared attributes between two tables T_1 and T_2, where T_1 is the base table and T_2 is the child table. We need to predict the values of attribute Y from T_2 and populate the result attribute tree. If the size of $\mathcal{X}$ is equal to n ($n \geq 1$), we would first start with AFDs having n attributes in determining set and 'significantly higher' confidence than any of their AFDs. We need 'significantly higher' confidence because if the additional attributes do not boost the confidence much, they will not increase the accuracy of prediction as well. If the AFDs do not find matching values between two tables to predict values, we 'fall back' to the AFDs with smaller determining set. We do this until we would be able to predict the value from the other table. Algorithm 3 describes it.

Algorithm 3 Tuple Expansion

```
Require: Source-table-tree $\mathcal{S}_t$; Result-attribute-tree $\mathcal{A}_t$, Set of AFDs $\bar{A}$
 1: $R$ := $\{\emptyset\}$ {Initializing the result set with schema $\mathcal{A}_t$ }
 2: $b$ := $Root(\mathcal{S}_t)$ {Setting the base table}
 3: Translate the constraints onto base table
 4: Populate all the attributes in level 0 of $\mathcal{A}_t$ from $b$
 5: for all child $c$ in $\mathcal{A}_t$ do
 6:    if $b$ and $c$ share $n$ attributes then
 7:       $fd$ = AFDs with $n$ attrbs in detSet
 8:       while $n > 0$ do
 9:          if $c$ has the specified combination then
10:             Populate $R$ using predicted values using $fd$ from $c$
11:             break
12:          $fd$ = Pick AFDs with $n - 1$ attributes in detSet
13: return Result Set $R$
```

7 LEARNING ATTRIBUTE DEPENDENCIES

We have seen in the previous section how attribute dependencies within and across tables help us in query answering by discovering related attributes from other tables. But it is highly unlikely that these dependencies will be provided up front by autonomous web sources. In fact, in most cases the dependencies are not apparent or easily identifiable. We need an automated learning approach to mine these dependencies.

As we have seen in the Section 6, we extensively use both attribute-level dependencies (AFDs) and value-level dependencies. The value level dependencies are nothing but *association rules*. The notion of mining AFDs as condensed representations of association rules is discussed in detail in [7]. Our work adapts the same notion, since it helps us in learning dependencies both at attribute and value level. The following sections describe how rules are mined within the table and how they are propagated across tables.

7.1 INTRA-TABLE LEARNING

In this subsection we describe the process of learning AFDs from a single table. But, only few of these AFDs are useful to us. To capture this, we define two metrics *Confidence* and *Specificity* for an AFD and learn those AFDs that have high *Confidence* and low *Specificity* values above the specified thresholds.

7.1.1 CONFIDENCE

If an Association rule is of the form $(\alpha \rightsquigarrow \beta)$, it means that if we find all of α in a row, then we have a good chance of finding β. The probability of finding β for us to accept this rule is called the confidence of the rule. Confidence denotes the conditional probability of head given the body of the rule.

TABLE VIII
FRAGMENT OF A CAR DATABASE

ID	Make	Model	Year	Body Style
1	Honda	Accord	2001	Sedan
2	Honda	Accord	2002	Sedan
3	Honda	Accord	2005	Coupe
4	Honda	Civic	2003	Coupe
5	Honda	Civic	1999	Sedan
6	Toyota	Sequoia	2007	SUV
7	Toyota	Camry	2001	Sedan
8	Toyota	Camry	2002	Sedan

To define confidence of an AFD, we would expect a similar definition, that it should denote the chance of finding the value for the dependent attribute, given the values of the attributes in the determining set. We define *Confidence* in terms of the confidences of the underlying association rules. Specifically, we define it in terms of picking the best association rule for every distinct value-combination of the body of the association rules. For example, if there are two association rules (Honda $\rightsquigarrow$ Accord) and (Honda $\rightsquigarrow$ Civic), given Honda, the probability of occurrence of Accord is greater than the probability of occurrence of Civic. Thus, (Honda $\rightsquigarrow$ Accord) is the best association rule, for (Make = Honda) as the body.

$$\begin{aligned} \textit{Confidence (X} \rightsquigarrow A) &= \sum_{x}^{N'} \arg \max_{y \in [1, N_j]} (\textit{support}\,(\alpha_x) \times \\ &\quad Confidence(\alpha_x \rightsquigarrow \beta_y)) \end{aligned} \tag{1}$$

Here, N' denotes the number of distinct values for the determining set X in the relation. This can also be written as,

$$\textit{Confidence (X} \rightsquigarrow A) = \sum_{x}^{N'} \arg \max_{y \in [1, N_j]} (\textit{support}\,(\alpha_x) \rightsquigarrow \beta_y) \tag{2}$$

Example: For the database relation displayed in table VIII, Confidence of the AFD $(Make \rightsquigarrow Model)$ = Support $(Make : Honda \rightsquigarrow Model : Accord)$ + Support $(Make : Toyota \rightsquigarrow Model : Camry)$ = $\frac{3}{8}$ + $\frac{2}{8}$ = $\frac{5}{8}$. [1]

7.1.2 Specificity-based Pruning

The distribution of values for the determining set is an important measure to judge the "usefulness" of an AFD. For an AFD $X \rightsquigarrow A$, the fewer distinct values of X and the more tuples in the database that have the same value, potentially the more relevant possible answers can be retrieved through each query, and thus a better recall. To quantify this, we first define the *support of a value* α_i of an attribute set X, $support(\alpha_i)$, as the occurrence frequency of value α_i in the training set.

$$support(\alpha_i) = count(\alpha_i)/N,$$

where N is the number of tuples in the training set.

Now we measure how the values of an attribute set X are distributed using *Specificity*. *Specificity* is defined as the information entropy of the set of all possible values of attribute set X: $\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, normalized by the maximal possible entropy (which is achieved when X is a key). Thus, *Specificity* is a value that lies between 0 and 1.

$$\textit{Specificity}\,(X) \quad = \quad \frac{-\sum_1^m support(\alpha_i) \times \log_2(support(\alpha_i))}{\log_2(N)}$$

[1]It is interesting to see that, this turns out to be equal to (1 - g_3), where g_3 is one of the standard error measures for defining AFDs. The g_3 error measure [8] has a natural interpretation as the fraction of tuples with exceptions or errors affecting the dependency.

When there is only one possible value of X, then this value has the maximum support and is the least specific, thus we have *Specificity* equals to 0. When all values of X are distinct, each value has the minimum support and is most specific. In fact, X is a key in this case and has *Specificity* equal to 1.

Now we overload the concept of *Specificity* on AFDs. The *Specificity* of an AFD is defined as the *Specificity* of its determining set.

$$\textit{Specificity}\,(X \rightsquigarrow A) = \textit{Specificity}\,(X)$$

The lower *Specificity* of an AFD, potentially the more relevant possible answers can be retrieved using the rewritten queries generated by this AFD, and thus a higher recall for a given number of rewritten queries.

Intuitively, *Specificity* increases when the number of distinct values for a set of attributes increases. Consider two attribute sets X and Y such that Y⊃X. Since Y has more attributes than X, the number of distinct values of Y is no less than that of X, *Specificity* (Y) is no less than *Specificity* (X).

Definition 7.1.1 (Monotonicity of *Specificity*)**.** *For any two attribute sets X and Y such that Y⊃X, Specificity (Y) $\geq$ Specificity (X). Thus, adding more attributes to the attribute set X can only increase the Specificity of X. Hence, Specificity is monotonically increasing w.r.t increase in the number of attributes.*

This property is exploited in pruning the AFDs during the mining, by eliminating the search space of rules with *Specificity* less than the given threshold.

Algorithms for mining AFDs face two costs: the combinatorial cost of searching the rule space and the cost of scanning the data to calculate the required metrics for the rules. In query processing the AFDs which we are mostly interested are the ones with the shared attributes in determining set of the rule. If $X \rightsquigarrow A$ is an AFD, we are interested in rules where $X \in S$, where S is the set of shared attributes between two tables. Since number of such attributes is typically small, we can use this as a one of the heuristics to prune off unwanted rules.

7.1.3 AFDMiner Algorithm

We formally define the AFD Mining problem in general as below:

Definition 7.1.2 (AFD Mining Problem). *Given a database relation **r**, and user-specified thresholds* minConf *(minimum confidence) and* max*Specificity (maximum Specificity), generate all the Approximate Functional Dependencies (AFDs) of the form ($X \rightsquigarrow A$) from r for which Confidence (X$\rightsquigarrow A$) $\geq minConf$ and Specificity (X) $\leq$* max*Specificity*

To find all dependencies according to the definition above, we search through the space of non-trivial dependencies and test the validity of each dependency. We follow a breadth first search strategy and perform a level-wise search in the lattice of attributes, for all the required AFDs. Bottom-up search in the lattice starts with singleton sets and proceeds upwards level-wise in the lattice, searching bigger sets. For AFDs, the level-wise bottom-up algorithm has a powerful mechanism for pruning the search space, especially the pruning based on *Specificity* .

Search starts from singleton sets of attributes and works its way to larger attribute sets through the set containment lattice level by level. When the algorithm is processing a set X, it tests AFDs of the form $X \setminus A \rightsquigarrow A$, where A $\in$ X.

Algorithm 4 briefly presents the main $AFDMiner$ algorithm.

Algorithm 4 AFDMiner: Levelwise search of dependencies

```
1: $L_0 := \{\emptyset\}$
2: $L_1 := \{\{A\} \mid A \in R\}$
3: $\ell := 1$
4: while $L_\ell \neq \emptyset$ do
5:     ComputeDependenciesAtALevel($L_\ell$)
6:     PRUNE($L_\ell$)
7:     $L_{\ell+1}$ := GenerateNextLevel($L_\ell$)
8:     $\ell := \ell + 1$
```

GenerateNextLevel computes the level $L_{\ell+1}$ from L_ℓ. The level $L_{\ell+1}$ will contain only those attribute sets of size $\ell + 1$ which have their subsets of size ℓ in L_ℓ. (ComputeDependenciesAtALevel(L_ℓ)) computes all the AFDs that hold true at the given level of the lattice. In this process, it computes the confidence of eah association rule constituting the AFDs. PRUNE(L_ℓ) implements the pruning strategies and prunes the search space of AFDs. It computes the *Specificity* of each rule, and if it is less than the specified threshold, eliminates all the rules with whose determining sets are supersets of it.

The process of mining AFDs for all attributes can be viewed as finding the best feature set for each attribute in the dataset. It is easy to see that the number of possible AFDs in a database table is exponential to the number of attributes in it; thus AFD mining is in general expensive. In order to achieve some computational savings, we consider only the attributes shared across tables in the determining set and employ a pruning strategy to stop the search based on the defined metrics. A more efficient alternative, which we may consider in the future, is to incorporate greedy approaches for learning determinations. Techniques for decision table induction, [9, 11], which use wrapper methods to select relevant features, provide an efficient mechanism for this purpose. In contrast to AFD mining, these greedy approaches scale linearly with the number of attributes.

7.2 Learning Source Statistics

Storing association rules: The probabilities which we used extensively in the query answering phase are nothing but the confidence of the association rules. So we store all the association rules mined during the process of AFD mining (specifically, in ComputeDependenciesAtALevel(L_ℓ))) and use them at query time. This saves us the additional cost of having to compute the association rules separately by traversing the whole lattice again.

Here we describe the value level source statistics gathered by the system, which are employed by the query answering module for constraint propagation and attribute value prediction. As mentioned earlier, AFD mining involves mining the underlying association rules. During association rule mining, following statistics are gathered from each source table $\mathcal{T}$:

1. $P_{\mathcal{T}}(X = x_i)$: Prior probabilities of distinct values for each attribute X in $A_{\mathcal{T}}$
2. $P_{\mathcal{T}}(X = x_i | Y = y_j)$: Conditional probabilities for distinct values of each attribute X conditioned on those of attribute Y in $A_{\mathcal{T}}$. Recall that this is nothing but the confidence of an association rule. Only the shared attributes are used as evidence variables, since value prediction and constraint propagation can only be performed across shared attributes.

7.3 Inter-table Chaining

After learning the AFDs within a table, we need to use them to derive inclusion dependencies which are used in query answering phase. In order to combine AFDs from different tables, we need anchor points. These anchor points are provided by the attribute mappings across tables, so we extend our attribute dependencies using them. When two AFDs between neighboring tables are combined, the resultant AFD would have a confidence equal to the product of the two confidences.

8 EXPERIMENTAL EVALUATION

In this section, we describe the implementation and an empirical evaluation of our system SMARTINT for query processing over multiple tables and learning attribute dependencies. Our prototype works on a local copy of the web databases for both efficiency as well as due to access restrictions for many of the Web databases. We implemented our system in Java and used MySQL database to store the tables. Before describing the experimental design, we state the experimental hypothesis.

Hypothesis: In the context of autonomous Web databases, if you learn Approximate Functional Dependencies (AFDs) and use them in query answering, then it would result in a better retrieval accuracy than using direct-join and single table approaches.

8.1 METHODOLOGY AND METRICS

To perform an oracular study based on a ground truth, we start with a *master table* which contains the tuples with universal relation and then randomly fragment it into multiple tables with varying number of shared attributes among them. This helps us not only in comparing SMARTINT with approaches without learning but also with ground truth. In order to compare the effectiveness of retrieving relevant answers and propagating constraints, we generalize the standard notion of precision and recall. As we shall see the, the generalization is needed as our answers can differ from ground truth(provided by the master table) both in terms of how many answers we get and how correct and complete each answer is. We explain how each of these metrics is measured below:

- **Correctness of a tuple** (cr_t): If the system returns a tuple with m attributes of n attributes in the universal relation, the correctness of a tuple is defined as the ratio of total number of correct values in the tuple to number of attributes returned.

$$cr_t = \frac{\text{Number of correct attribute values in the tuple}}{\text{Number of attributes in `returned result set' (m)}}$$

- **Precision of the result set** (P_{rs}): Precision of the result set is defined as the average of correctness of a tuple in the result set.

$$P_{rs} = \frac{\Sigma cr_t}{\text{Total number of tuples in `returned result set'}}$$

- **Completeness of a tuple**(cp_t): If the system returns a tuple with m attributes of n attributes in universal relation, the correctness of a tuple is defined as the ratio of total number of correct values in the tuple to number of attributes in universal relation.

$$cp_t = \frac{\text{Number of correct values in the tuple}}{\text{Number of attributes in `master table' (n)}}$$

- **Recall of the result set** (R_{rs}): is defined as the ratio of the cumulative completeness of the tuples returned by the system to the total number of answers.

$$R_{rs} = \frac{\Sigma cp_t}{\text{Number of tuples retrieved from `master table'}}$$

We compare SMARTINT results with two approaches discussed in Section 2 which do not learn attribute dependencies(1) SINGLE TABLE: In this approach, results are retrieved from a single table which has maximum number of attributes/constraints mentioned in the query mapped on it.(2) DIRECT-JOIN: The other approach we are comparing with is joining the tables based on the shared attributes. As explained in the introduction, it results in a lot of erroneous results which hurts the precision.

The query format allows two variable parameters, projected attributes and constraints. The implications of changing them are discussed below.

- **Number of Attributes:** With the increase in number of attributes, the source tables they map on to are also likely to increase, which implies having to do more data integration (through tuple expansion) in order to provide more complete answers.

- **Number of Constraints:** As the number of constraints in the query increase, it becomes more likely that they will map on to different tables. Hence, it requires more constraint propagation, in order to provide more precise answers.

8.2 EXPERIMENTAL DESIGN

To evaluate the SMARTINT system, we evaluated Vehicles database. We used around 350,000 records probed from Google Base for the experiments. We created a *master table* with 18 attributes which do not have 'null' values. We divided this *master table* into multiple child tables with overlapping attributes. This helps us in evaluating the returned 'result set' with respect to the results from master table and establish how our approach compares with the ground truth.We have divided the master table into 5 different tables with the following schema

- *Vehicles_Japanese: (condition, price_type, engine, model, VIN, vehicle_type, payment,door_count, mileage, price, color , body_style, make)*

- *Vehicles_Chevrolet: (condition, year, price, model, VIN, payment, mileage, price, color, make),*

- *Vehicles_Chevrolet_Extra: (Model, Door Count, Type, Engine)*

- *Vehicles_Rest: (condition, year, price ,model, VIN, payment, mileage, price, color, make)*

- *Vehicles_Rest_Extra: (Engine, Model, Vehicle Type, door count, body style)*

The following (implicit) attribute overlaps were present among the fragmented tables.

- *Vehicles_Chevrolet:Model ↔ Vehicles_Rest:Model*
- *Vehicles_Chevrolet:Year ↔ Vehicles_Rest:Year*
- *Vehicles_Rest:Year ↔ Vehicles_Rest_Extra:Year*
- *Vehicles_Chevrolet_Extra:Model ↔ Vehicles_Rest_Extra:Model.*

We posed automatically generated random queries to the system and measured the recall and precision (which are described earlier) of SMARTINT system. We compared the performance of SMARTINT with 'Single table' and 'direct Join' approach discussed in Section 2. The following are the input parameters which are changed: (1) Number of Attributes and (2) Number of Constraints. We measured the value of precision and recall by taking the average of the values for different queries. While measuring the value for a particular value of a parameter we varied the other parameter. While we are measuring precision for 'Number of attributes = 2', we posed queries to the system with 'Number of constraints = 2, 3 and 4' and took the average of all these values and plotted them. Similarly, we varied the 'Number of attributes' while we are measuring the Precision for each value of 'Number of constraints'. The same process is repeated for measuring the recall as well.

8.3 EXPERIMENTAL RESULTS

In this section, we present the results from experiments and analyze them.

In the simple case of queries mapping on to a single table, the precision and recall values are independent of attribute dependencies, since query answering does not involve constraint propagation or tuple expansion through attribute value prediction.

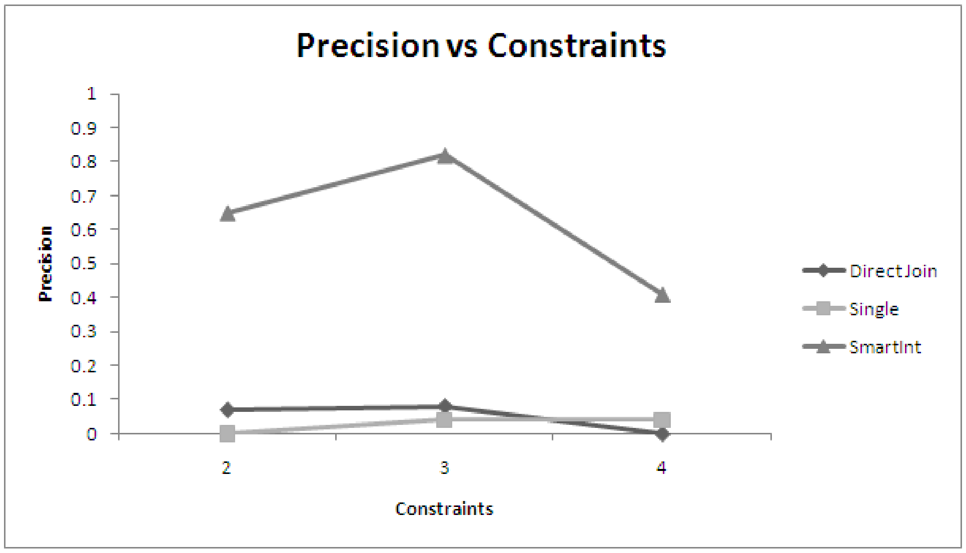

Fig. 5. Precision Vs Number of Constraints

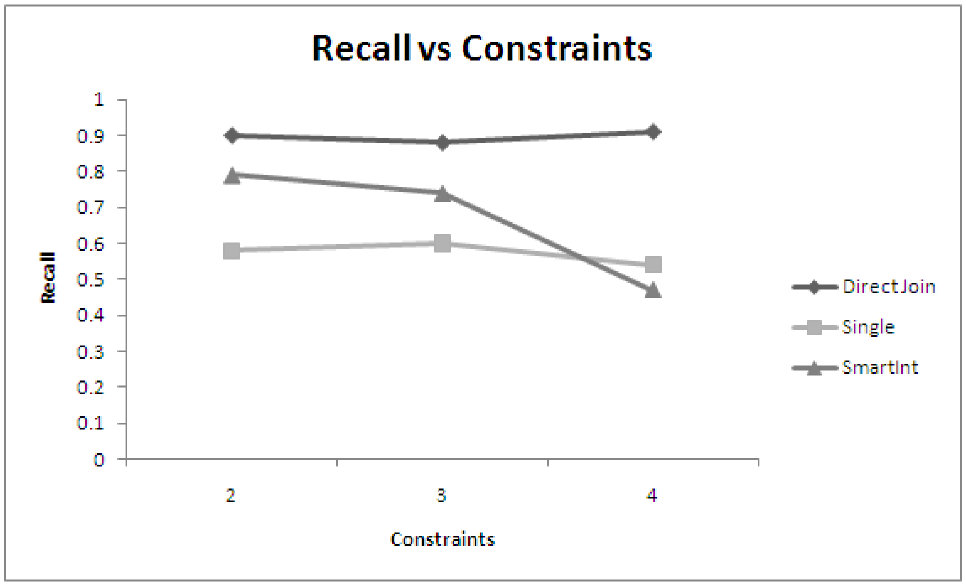

Fig. 6. Recall Vs Number of Constraints

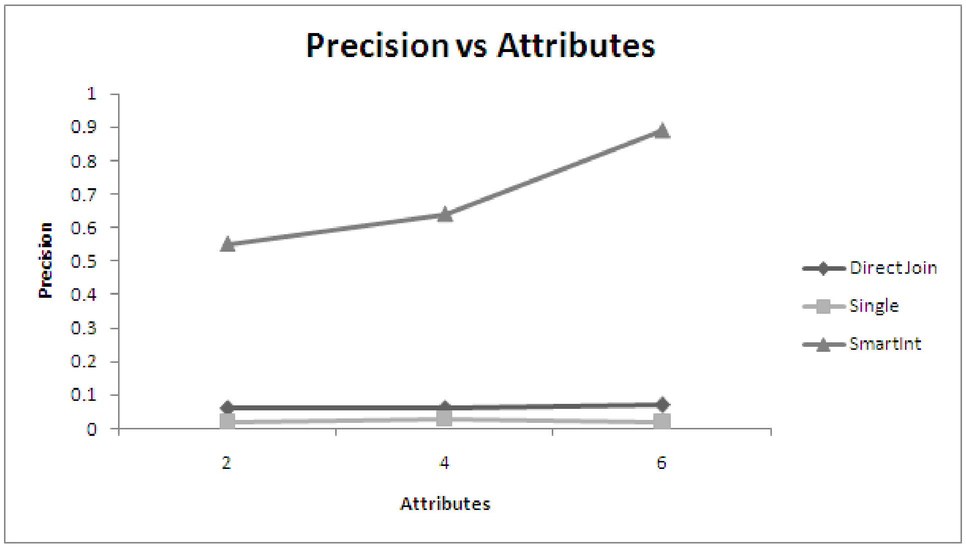

Fig. 7.   Precision Vs Number of Attributes

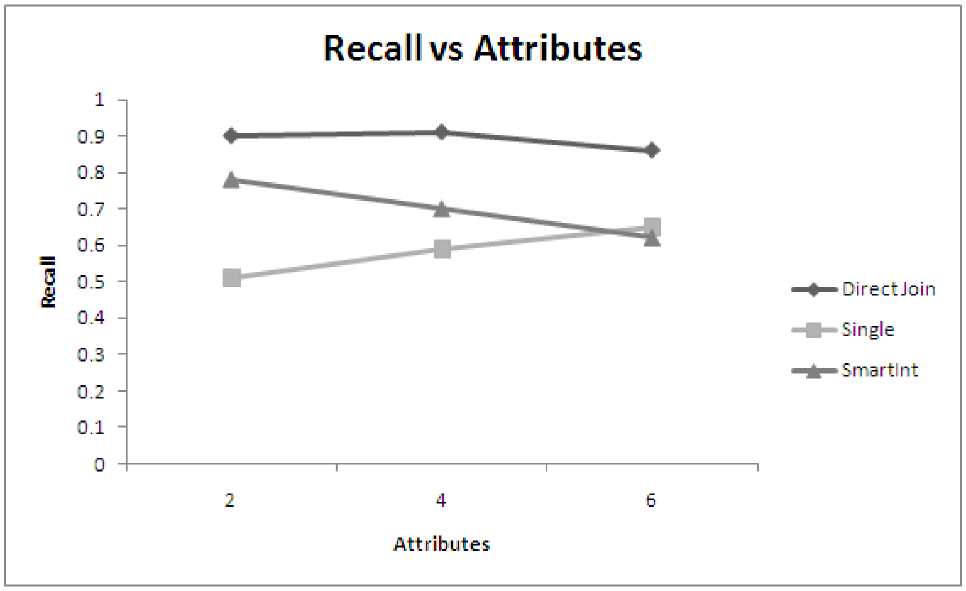

Fig. 8.   Recall Vs Number of Attributes

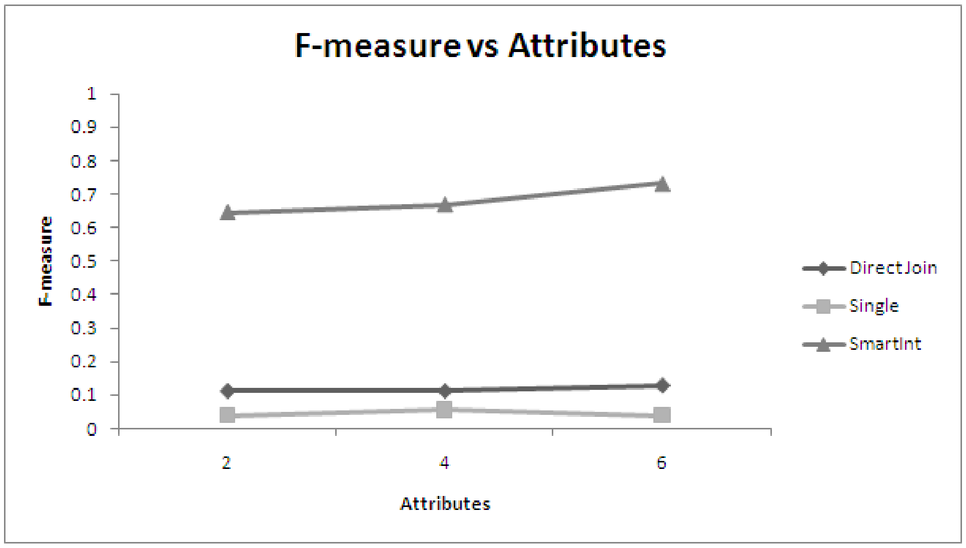

Fig. 9. F-measure Vs Number of Attributes

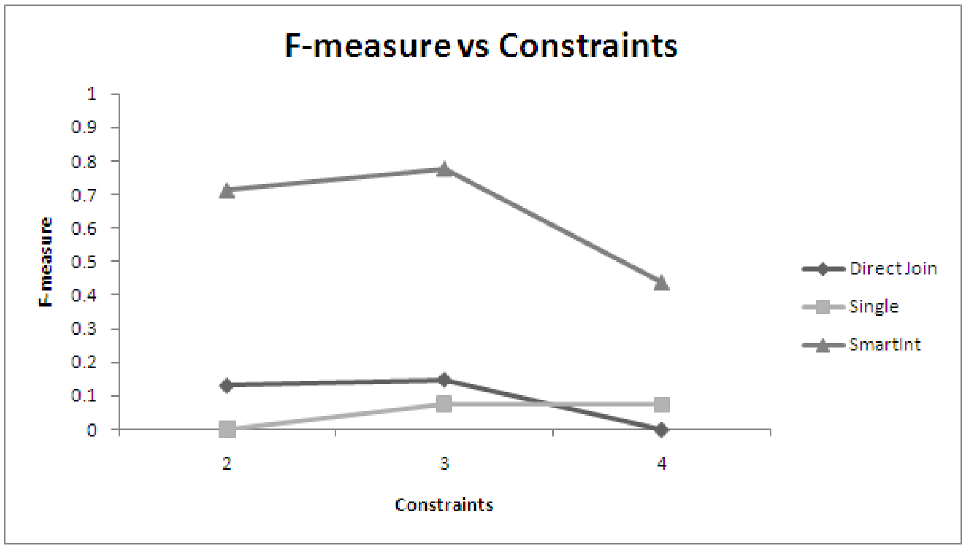

Fig. 10. F-measure Vs Number of Constraints

In cases where queries span multiple tables, some of the attribute values have to be predicted and constraints have to propagated across tables. Availability of attribute dependency information allows accurate prediction of attributes values and hence boosts precision. As shown in Figures 5 and 7, our approach scored over the other two in precision. Direct join approach, in absence of primary-foreign key relationships, ends up generating non-existent tuples through replication, which severely compromises the precision. In cases where query constraints span over multiple tables, single table approach ends up dropping all the constraints except the ones mapped on to the selected best table. This again results in low precision.

In terms of recall (Figures 6 and 8), performance is dominated by the direct join approach, which is not surprising. Since direct join combines partial answers from selected tables, the resulting tuple set contains most of the real answers, subject to completeness of individual tables. Single table approach, despite dropping constraints, performs poorly on recall. The selected table does not cover all the query attributes, and hence answer tuples are low on completeness, which affects recall.

When accurate attribute dependencies are available, our approach processes the distributed query constraints effectively and hence keeps the precision fairly high. At the same time, it performs chaining across tables to improve the recall. Figures 9 and 10 show that our approach scores higher on F-measure, hence suggesting that it achieves a better balance between precision and recall.

Time Taken for Query Execution: We have observed that the quality of SMARTINT results is better when compared to other approaches like direct join and single table. SMARTINT also exhibits lower query execution time than the direct join approach. In

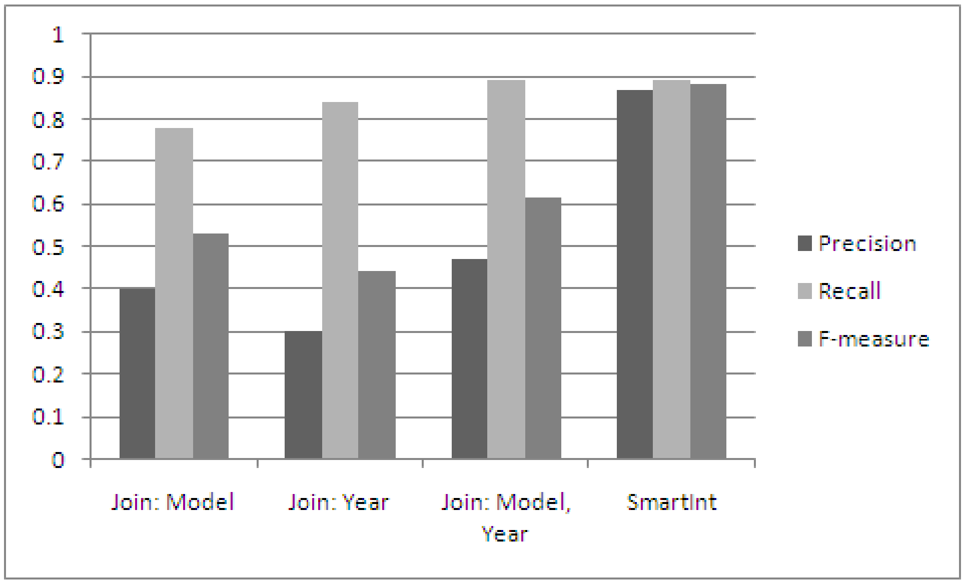

Fig. 11. SMARTINT Vs Multiple Join Paths

one experiment, SMARTINT took an average of 5.14 seconds for query execution, whereas direct join took 72.94 seconds, over ten times as long.

8.4 COMPARISON WITH MULTIPLE JOIN PATHS

In the previous evaluation the data model had one shared attribute between the tables, but there can be multiple shared attributes between the tables. In such scenarios, direct join can be done based any combination of the shared attributes. Unless one of the attribute happens to be a key column the precision of the joins is low. In order to illustrate this, we considered the data model with more than one shared attribute and measured the precision and recall for all the possible join paths between the tables. The experimental results (See Figure 11) show that SMARTINT had higher F-measure than all possible join paths.

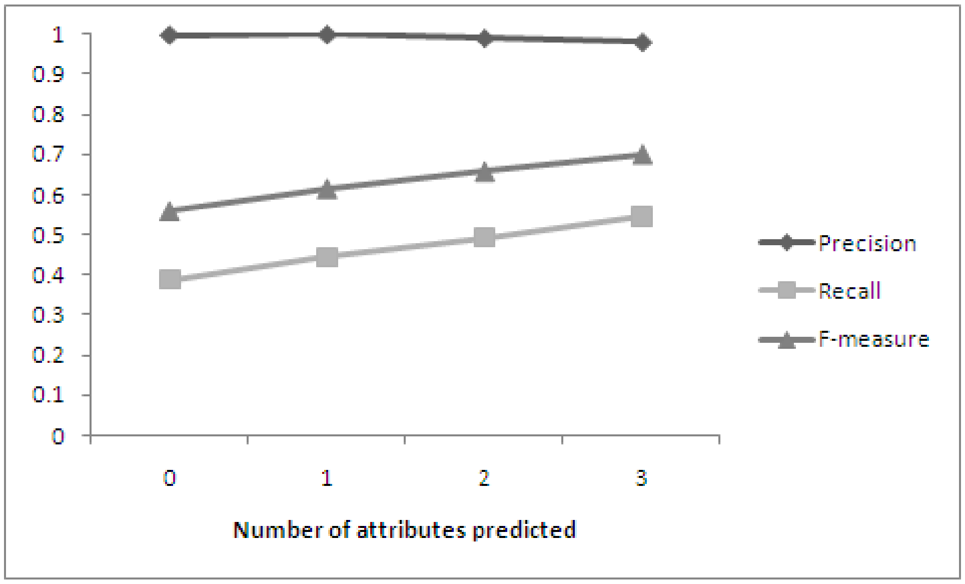

Fig. 12. Precision, Recall and F-measure Vs Width of the Tuple

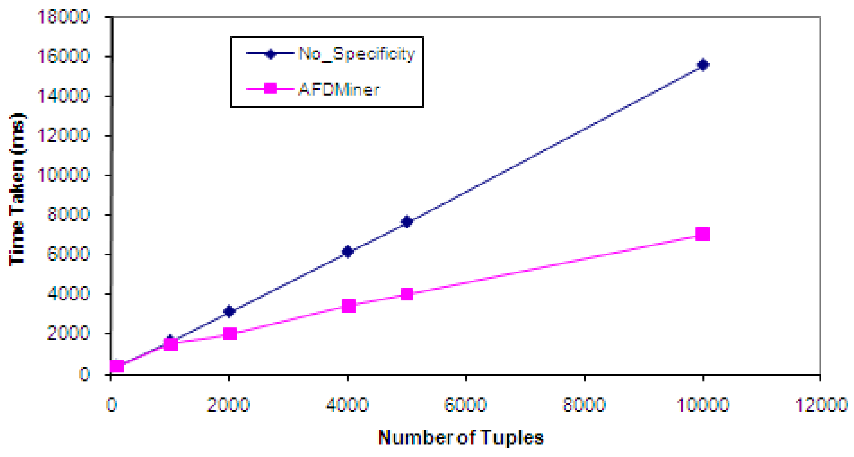

Fig. 13. Time Taken by AFDMiner Vs No. of Tuples

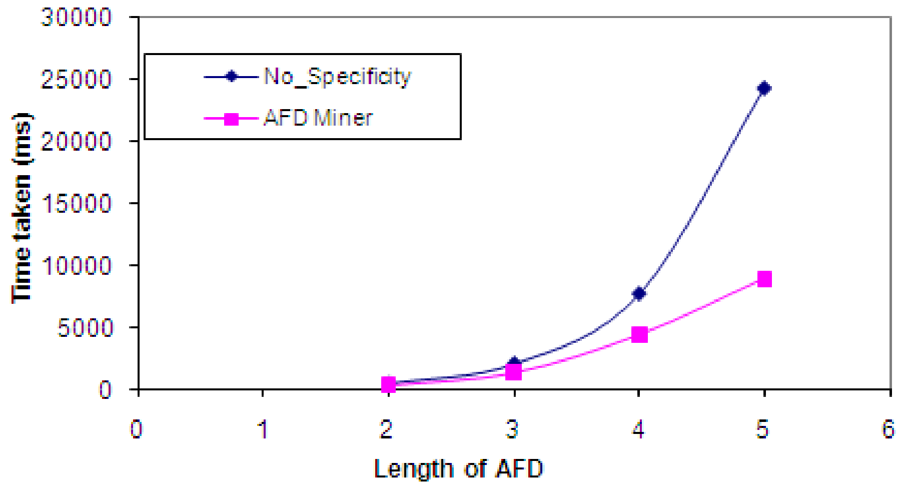

Fig. 14. Time Taken by AFDMiner Vs Length of AFD

8.5 TRADEOFFS IN NUMBER VS. COMPLETENESS OF THE ANSWERS

Normal query processing systems are only concerned about retrieving top-k results since the width(number of attributes) of the tuple is fixed. But SMARTINT chains across the tables to increase the extent of completion of the entity. This poses an interesting tradeoff: In a given time, the system can retrieve more tuples with less width or fewer tuples with more width. In addition to this, if user is only interested high confidence answers, each tuple can expand to variable width to give out high precision result set. We analyze how precision and recall varies with 'w' (number of attributes to be shown). The Figure 12 shows how precision, recall and F-measure varies as more number of attributes are predicted for a specific result set(The query constraints are make='BMW' and year='2003'). In scenarios, when SMARTINT has to deal with infinite width tuples, F-measure can be used to guide SMARTINT when to stop expanding.

8.6 AFDMiner Evaluation

We invoke AFDMiner to learn the association rules and the AFDs. As explained in Section V AFD mining is an expensive process and might end up taking huge time. For the current data setup, AFDMiner took approximately 15 minutes to learn the AFDs. Figure 13 and 14 show the comparison between the time taken for AFDMiner and the approach which does not use specificity metric, with varying tuplesize and the length of the AFD respectively. We can observe that the AFDMiner takes significantly less time than other approach and makes the mining process tractable. For a detailed experimentation on AFDMiner, we ask the reader to refer to [7]

9 CONCLUSION AND FUTURE WORK

In this section the thesis is summarized and the future extensions for the work are discussed.

9.1 CONCLUSION

Our work is an attempt to provide better query support for web databases having tables with shared attributes using learned attribute dependencies but missing primary key - foreign key relationship. We use learned attribute dependencies to make up for the missing PK-FK information and recover entities spread over multiple tables. Our experimental results demonstrate that the approach used by SMARTINT is able to strike a better balance between precision and recall than can be achieved by relying on a single table or employing direct joins.

9.2 FUTURE WORK ON QUERY PROCESSING

There are certain extensions to this work that we are aiming to pursue.

1. The current system assumes the user is equally interested in seeing each of the attributes specified in the query. Similarly, we assume that each of the additional attributes would contribute equally to the relevance. This model can be generalized with attributes having different weights of "interestingness".

2. In the current system we assume that there is a finite width universal relation and tuple expander would either terminate if it cannot get more relevant attributes or if it reaches the maximum number of attributes. If we do not assume the finite width model for universal relation, the tuple expander would continuously expand to increase it relevance and improve its recall. This would lead to result tuples with a large number of attributes and information overload. This problem can be countered

by employing a ‘discounted reward model’ which penalizes the tuple expander for expanding beyond a certain number of attributes.

3. As the number of tables increase, the number of relevant attributes base set can be quite large. Presenting all the relevant attributes would overwhelm the user with information. In such cases having more granular relevance for attributes for the user is important. [14] is similar in spirit, and we consider adapting such works for our scenarios. Analyzing query logs to mine attribute relevance would be an interesting extension of our work.

4. Furthermore, in the scenarios we consider, it is quite possible that all the query constraints cannot be evaluated on the sources. In such a case, it would be interesting to have constraints with degrees of “strictness”, which can then be evaluated according to their priority so as to keep the precision high.

9.3 Future Work on Learning

In this work, we learn rules within the table and chain them across tables using shared attributes (anchor points). We have observed that this technique works quite well for the setting which we described in section 2. In fact we could have done the chaining of rules before hand (as a part of learning module) and used them as rules across tables. But an interesting problem is to mine rules across the tables without need for anchor points. This would remove the limitation of the query answering module of using anchor points to predict the target value. These kind of generalized mappings would help in predicting the values which are not seen in the training data. Generalized mappings are useful when mapping across two different partitions of tables from the same universal relation. Apart from this, they are also useful in mapping between two different domains.

Learning these kind of dependencies could use ideas from Inductive Logical Programming (ILP). In order to use this approach for the learning task, the elements from the database tables must be represented in terms of relations and arguments. One such representational scheme would denote an attribute from a table by a relation, and the tuple identifier and the assigned value as the arguments. This representational scheme would enable learning at the level of select attributes and values, rather than using entire tables.

These directions for future research will build on the existing work, and offer promising possibilities for using Web databases in meaningful ways.

10 STATEMENT ABOUT INDIVIDUAL CONTRIBUTION

The work presented in this thesis is the outcome of combined efforts by Ravi Gummadi and me. This chapter states our individual and joint contributions towards the work, during different phases of the project.

10.1 Problem Identification and Solution Formulation

We were both equally involved in formulating the problem, as well as proposing a solution framework in terms of its high-level components. Thereafter, we worked towards developing separate components, where I mainly focussed on coming up with a source selection (section 6.1) mechanism, while Gummadi focussed on conceptualizing the tuple expansion (section 6.2) process. We both contributed equally towards the learning module (section 7).

10.2 Implementation and Evaluation

As with the conceptualization, I primarily focussed on the implementation of the source selection module, while Gummadi primarily contributed in developing the tuple expansion module. We both contributed towards the learning module and performing experimental evaluation of the system. I came up with the evaluation metrics, and designed and performed the benchmarking experiments aimed at comparing SmartInt with single-table and direct-joins based approaches.

10.3 Writing

Chapter 1 and chapter 9 from this thesis have been written completely by me. The rest of the chapters are largely based on the content which was earlier used for technical papers we submitted to SIGMOD, VLDB and ICDE. I have mainly contributed in structuring and writing the chapters 5, 7 and 8.

REFERENCES

[1] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 564–575. VLDB Endowment, 2004.

[2] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, S. Sudarshan, and I.I.T. Bombay. Keyword searching and browsing in databases using banks. *Data Engineering, International Conference on*, 0:0431, 2002.

[3] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 850–861. VLDB Endowment, 2003.

[4] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.

[5] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[6] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 647–658, New York, NY, USA, 2004. ACM.

[7] Aravind Krishna Kalavagattu. Mining approximate functional dependencies as condensed representations of association rules. Master's thesis, Arizona State University, 2008.

[8] Jyrki Kivinen and Heikki Mannila. Approximate dependency inference from relations. In *ICDT*, pages 86–98, 1992.

[9] Ron Kohavi. The power of decision tables. In *ECML '95: Proceedings of the 8th European Conference on Machine Learning*, pages 174–189, London, UK, 1995. Springer-Verlag.

[10] Eric Lambrecht, Subbarao Kambhampati, and Senthil Gnanaprakasam. Optimizing recursive information-gathering plans. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1204–1211, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[11] Pat Langley. Induction of condensed determinations. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96*, pages 327–330. AAAI Press, 1996.

[12] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[13] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 117–128, 2002.

[14] M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 356–365, April 2008.

[15] Ullas Nambiar and Subbarao Kambhampati. Answering imprecise queries over autonomous web databases. In *ICDE*, page 45, 2006.

[16] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. Efficient keyword search across heterogeneous relational databases. *Data Engineering, International Conference on*, 0:346–355, 2007.

[17] Garrett Wolf, Hemal Khatri, Yi Chen, and Subbarao Kambhampati. Quic: A system for handling imprecision & incompleteness in autonomous databases (demo). In *CIDR*, pages 263–268, 2007.

[18] Garrett Wolf, Hemal Khatri, Bhaumik Chokshi, Jianchun Fan, Yi Chen, and Subbarao Kambhampati. Query processing over incomplete autonomous databases. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 651–662. VLDB Endowment, 2007.