SMARTINT: A System for Answering Queries over Web Databases Using Attribute Dependencies

Ravi Gummadi [1], Anupam Khulbe [2], Aravind Kalavagattu [3], Sanil Salvi [4], Subbarao Kambhampati [5]

Department of Computer Science, Arizona State University
Tempe, AZ, USA
[1]gummadi@asu.edu [2]akhulbe@asu.edu [3]aravindk@asu.edu [4]sdsalvi@asu.edu [5]rao@asu.edu

***Abstract*— Many web databases can be seen as providing partial and overlapping information about entities in the world. To answer queries effectively, we need to integrate the information about the individual entities that are fragmented over multiple sources. At first blush this is just the inverse of traditional database normalization problem - rather than go from a universal relation to normalized tables, we want to reconstruct the universal relation given the tables (sources). The standard way of reconstructing the entities will involve joining the tables. Unfortunately, because of the autonomous and decentralized way in which the sources are populated, they often do not have Primary Key - Foreign Key relations. While tables do share attributes, naive joins over these shared attributes can result in reconstruction of many spurious entities thus seriously compromising precision. Our system, SMARTINT is aimed at addressing the problem of data integration in such scenarios. Given a query, our system uses the Approximate Functional Dependencies(AFDs) to piece together a tree of relevant tables and schemas for joining them. The result tuples produced by our system are able to strike a favorable balance between precision and recall.**

I. INTRODUCTION

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. The database administrator (who ensures lossless normalization) is replaced by *independent data providers*, and specialized users (who are aware of database querying language) are replaced by *lay users*. These changes have two important implications:

- **Ad hoc Normalization by providers**: Primary key-Foreign key (PK-FK) 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 by data providers into different tables, and synthetic keys (such as vehicle ids, model ids, employee ids) are simply ignored.[1]
- **Imprecise queries by lay users**: 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" [1] in that they may miss requesting some of the relevant attributes about the entities under consideration.

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

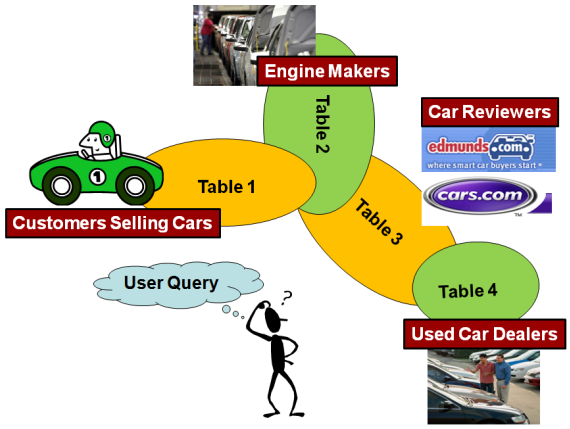

Fig. 1. Overlapping tables in the database

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 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. However, many reasonable solutions, such as SIMIFLOOD [2], 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.)

Our aim is to provide a fully automated solution to this problem. 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 such mappings, and the difficulty of constructing sound and complete mappings in the absence of PK-FK relationships.

Motivating Example: As a motivating scenario, let us consider a set of tables (with different schemas) populated in a Vehicle domain (Figure 1). The universal schema of entity 'Vehicle' can be described as follows: *Vehicle (VIN, vehicle-type, location, year, door-count, model, make, review, airbags, brakes, year, condition, price, color, engine, cylinders, capac-*

ity, 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) and 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 shared understanding 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.
- Because of the decentralized way the sources are populated, it is hard for the sources to agree on "synthetic keys" (that sometimes have to be generated during traditional normalization).
- The primary key may be intentionally masked, since it describes sensitive information about the entity (e.g. social security number).

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 a 4-cylinder engine.**[2] The above query would translate to the following SQL notation.

[2]We use this example throughout the paper to illustrate working of different modules of the system

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	...

```
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.

Limitations of join-based solutions: 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 the 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 constraints would lead to erroneous tuples in the final result set which do not conform to the query constraints.
- **Direct Join**: The second and a seemingly more reasonable approach is joining the tables using whatever shared attribute(s) are available. 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. The absence of Primary Key - Foreign Key relationship between these two tables has lead to spurious results.

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

- 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 [2]. However, these alone would not be

TABLE V
Results of Query Q just 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	...

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 the knowledge of attribute dependencies, which are not apparent.

Our Approach

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. Intuitively, the base table should contain important attributes whose values cannot be predicted accurately, but which can help in predicting the values of the other relevant attributes. The prediction/completion of the tuples is made possible by approximate functional dependencies (AFDs). As a simple illustration of the idea, suppose the following simple AFDs[3] are mined from our tables: (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.

II. SmartInt System

In this section, we present our framework for query answering and learning called SmartInt. Conceptually, the operation of SmartInt can be understood in terms of (i) a *learning component,* which mines AFDs and source statistics from different sources and (ii) a *query answering component* which actively uses the learned statistics to propagate constraints and retrieve attributes from other non-joinable tables. Figure 2 shows the SmartInt system architecture. In the next few sections, we explain each module of SmartInt in detail.

A. Query Answering

Query answering takes care of answering the queries by selecting the most appropriate tables and subsequently operating on those tables to get the final result set.

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

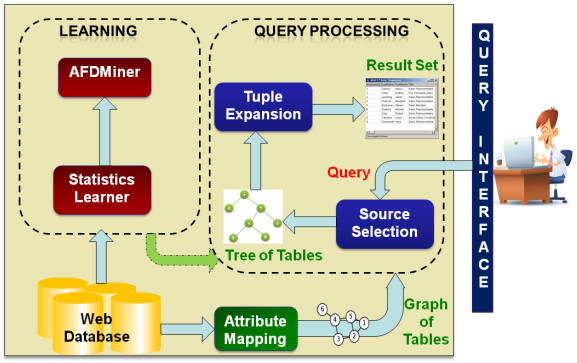

Fig. 2. Architecture of SmartInt System

Source Selector: The first step, after the user submits the query, is to select most relevant tables from the potentially large number of tables present in the database. Source selector outputs a 'tree of relevant tables'. In order to construct the tree, it first picks the top n tables (where n is the maximum size of the tree). Then it evaluates the relevance of all the subgraphs of tables of size k and picks the most relevant subgraph. It then picks the most relevant tree within that graph and returns it. Estimating the relevance of tables with respect to the query during query processing can be costly. SmartInt uses 'Source Statistics' learned beforehand to calculate the degree of relevance of a table for a particular query.

Tuple Expander: The next step in the query answering phase is to use the 'tree of tables' returned by the source selector to construct the final result set. Tuple expander first constructs the hierarchical schema from the table tree. It uses AFDs to determine which attributes from the child table are appended to the schema. Once the schema is constructed, it starts populating the tuples corresponding to this schema. It first queries the 'root table' from the 'table tree' and then starts appending the determined attributes using the stored values from 'Source Statistics' module.

B. Learning

In the previous section, we have seen that query answering phase uses both AFDs and source statistics extensively. Learning these is thus an integral part of SmartInt system. In this section we describe how attribute dependencies and source statistics are learnt.

AFD Miner (Intra-table Learning): As we described in Section I, we extensively use AFDs in 'Tuple Expansion' phase. We define and mine AFDs as condensed representations of association rules. For example, an AFD (*Model*$\rightsquigarrow$*Make*) is a condensation of association rules like, (*Model:Accord*$\rightsquigarrow$*Make:Honda*), (*Model:Camry*$\rightsquigarrow$*Make:Toyota*) *etc.* We define two metrics, namely *confidence* and *specificity* analogous to the standard metrics confidence and support used in association rules respectively, to get high quality AFDs. $AFDMiner$

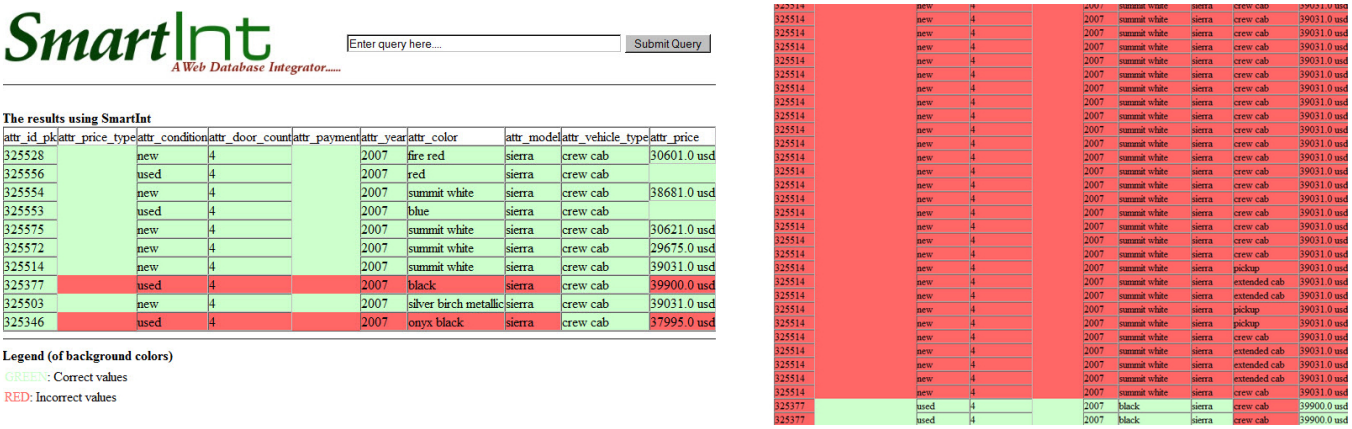

Fig. 3. Screenshots of demo: A. Result Set using SMARTINT - B. Result Set using Direct Join

efficiently mines high quality AFDs using efficient pruning strategies. It performs a bottom-up search in the attribute lattice to find all AFDs and FDs that fall within the given *confidence* and *specificity* thresholds. Complete details can be found in [4].

Stat Learner (Source Statistics): Source statistics are extensively used in both 'Source Selector' and 'Tuple Expander'. It might seem that Stat Learner would require another scan of the database to mine useful statistics. But since we mined AFDs by rolling up association rules, and confidence of an association rule is nothing but the conditional probability of the attribute-value pairs involved, we store them during the AFD mining process. Apart from these, we also store information related to value distribution in each column to calculate the extent of overlap between two tables. This helps us in approximating the relevance measure during query time.

Inter-table Learning: We use the AFDs learnt within each table along with the attribute mappings(which serve as anchor points) to learn rules across tables. While combining AFDs across tables, we multiply their confidence to get the confidence of the final rule. When the tables are adjacent, learning rules across tables is trivial. But when two tables are not directly connected and can have multiple ways in which a rule can be determined, we need to evaluate all the possible rules possible and pick the best among them.

III. DEMONSTRATION

The demonstration of SMARTINT system will illustrate the improvement in accuracy of results over standard approaches like direct join and single table. The system searches a set of tables related to vehicle domain crawled from Google Base. The system would construct a tree of most relevant tables with a designated base table and predicts the values of other attributes from neighboring tables.

Scenario 1. User Experience: When SMARTINT operates on a real web database, each value in the tuple has different conforming probability. Since showing the actual probabilities will confuse the user, SMARTINT will color the values based on the interval into which their probability falls (for example, green for 90-100% confidence, red for less than 10% confidence).

Scenario 2. Evaluation: Just showing the result set with confidence would not help in comparing SMARTINT with other approaches. In order to give the user an intuitive sense of how accurate results given by different approaches are, we compare them with ground truth (or universal table) and highlight the values which do not match in red. A comparison of the result set between SMARTINT (Figure 3.A) and direct join (Figure 3.B), shows that SMARTINT generates more accurate results.

Scenario 3. Learning: Since learning in SMARTINT is done offline before query processing, we will show the different Approximate Functional Dependencies(AFDs) mined from Web databases. This will help the users understand the kind of attribute dependencies present in the data and how they help in query processing.

Scenario 4. Source Selection: One important aspect of SMARTINT is to pick a base table and then construct the tree of tables. Since the final result set does not help the user in inferring these steps, we clearly show how the source selection step works by showing the most relevant graph and tree selected from the available tables in the database.

Additional Details: A more comprehensive description and evaluation of SMARTINT system can be found in [5].

Acknowledgements: We thank Pat Langley for helpful discussions and feedback. This research is supported in part by the NSF grant IIS-0738317, the ONR grant N000140910032 and a Google research award.

REFERENCES

[1] U. Nambiar and S. Kambhampati, "Answering imprecise queries over autonomous web databases," in *ICDE*, 2006, p. 45.

[2] 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*, 2002, pp. 117–128.

[3] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen, "TANE: An efficient algorithm for discovering functional and approximate dependencies," *The Computer Journal*, vol. 42, no. 2, pp. 100–111, 1999.

[4] A. K. Kalavagattu, "Mining approximate functional dependencies as condensed representations of association rules," Master's thesis, Arizona State University, 2008. [Online]. Available: http://rakaposhi.eas.asu.edu/Aravind-MSThesis.pdf

[5] R. Gummadi, A. Khulbe, A. Kalavagattu, and S. Kambhampati, "Improving retrieval accuracy in web databases by learning intra-table and inter-table dependencies," Department of Computer Science, Arizona State University, Tempe, Arizona, Tech. Rep. TR-09-011, March 2009. [Online]. Available: http://sci.asu.edu/news/technical/TR_PDF/TR-09-011.pdf