

Interactive Schema Integration with Sphinx

François Barbançon, Daniel P. Miranker

Department of Computer Sciences, 1 University Station C0500
The University of Texas at Austin, Austin TX 78712-0233 USA
{francois, miranker}@cs.utexas.edu

Abstract. The Internet has instigated a critical need for automated tools that facilitate integrating countless databases. Since non-technical end users are often the ultimate repositories of the domain information required to distinguish differences in data types, we suppose an effective solution must integrate simple GUI based data browsing tools and automatic mapping methods that eliminate technical users from the solution. We develop a meta-model of data integration as the basis for absorbing feedback from an end-user. The schema integration algorithm draws examples from the data and learns integrating view definitions by asking a user simple yes or no questions. The meta-model enables a search mechanism that is guaranteed to converge to a correct integrating view definition without the user having to know a view definition language such as SQL or even having to inspect the final view definition. We show how data catalog statistics, normally used to optimize queries, can be exploited to parameterize the search heuristics and improve the convergence of the learning algorithm.

1 Introduction

Integrating data from a given set of databases requires making choices about the nature and validity of potential semantic relationships between elements in the database schemas. When the semantic structure of the databases is fully declared, it is possible to derive semantically correct view definitions and schema mappings to federate those databases [14]. On the other hand, omissions, or uncertainties about underlying semantic properties of the data can result in several competing view definitions, each of them corresponding to a different interpretation of the underlying data representation. We postulate that end-users, through domain expertise, intuitively possess the necessary knowledge to discriminate between competing interpretations.

Many problems in heterogeneous database integration can already be resolved by encompassing user interaction in a point and click interface. This is the case with the specification of attribute mappings in the schema mapping problem, and with building synonym correspondences and unit conversions for a data dictionary. This is not the case for schema integration (data deriving from more than one table). In addition to the above, schema integration requires the specification of query-operators including JOIN and/or higher-order query constructs [12], which have not admitted simple GUI interfaces. We propose a prototype system named Sphinx, designed to extract this knowledge without requiring any abstraction skills from the user.

The reader may recognize that the Clio system [29] entails a similar problem definition and Sphinx, like Clio, will combine both user interface and machine learning techniques. Clio’s learning component examines possible join paths in the query graph and ranks possible view definitions by estimating their likelihood. Clio then uses data examples to help the user decide between alternative mappings. The user examines a set of illustrative examples as well as the output of the system for the top-ranked view definitions. The user examines those examples and renders judgment as to whether it has converged on a correct transformation. If it hasn’t, the user can provide further assistance through an interaction with the system, based on a range of operators which help guide its convergence to the correct result: “Clio helps the user understand the results of the mappings being formed and allows the user to verify that the transformations that result are what she intended” [29].

Our work is distinguished from Clio by having first developed a linguistic meta-model of integrating view definitions enabling an active learning algorithm. The benefits include the guarantee that through a simple, question-based, interaction, Sphinx will converge to a correct result and report to the user when it has done so. Further, the user will not need to verify or refine the resulting view definition.

Sphinx extends the Version Spaces learning algorithm to represent a meta-model of all possible view definitions. We detail the Version Space model by considering, syntactically, a subset G of SQL, which may appear in the learned view definition. The generated language $L(G)$ has few semantic limitations, but the limitations we do place allow us to contain the size of the search space within tractable bounds. We feel we do so without significantly compromising the language’s practical expressiveness.

The active learning system seeks to learn a representative instance in $L(G)$ by generating data examples chosen to maximize information gain, and to reduce the total number of examples viewed by the user. In addition to identifying and ordering heuristics by virtue of our own knowledge of data integration problems, the system chooses a heuristic based on a cost function parameterized by using system catalog statistics in a manner analogous to optimizing query costs.

Coupling Version Spaces with sample selection from an existing database required us to extend the paradigm. In particular, a new kind of rule for *missing examples* is introduced to deal with ambiguities introduced by various functional dependencies and constraints that occur in schemas and data sets.

2 Defining an Intentional View

Consider two sources: Product Reviews and Supplier Catalog, with overlapping content. In order to make information from both those sources simultaneously accessible from a single search interface, we define a federated schema. By expressing a federated view as a query over the source data, we fully specify the underlying data integration.

A mapping of attributes from the source schema to the target (federated) schema is immediately apparent (Figure 1). Each arrow shows a correspondence between fields in the source and the target schema. This schema mapping process is natural and intuitive given the clear correspondences in our example. However this schema mapping is not sufficient to unambiguously define a federated view over the source data.

Several questions must be answered hinging on the nature of semantic relationship between elements in the source schema.

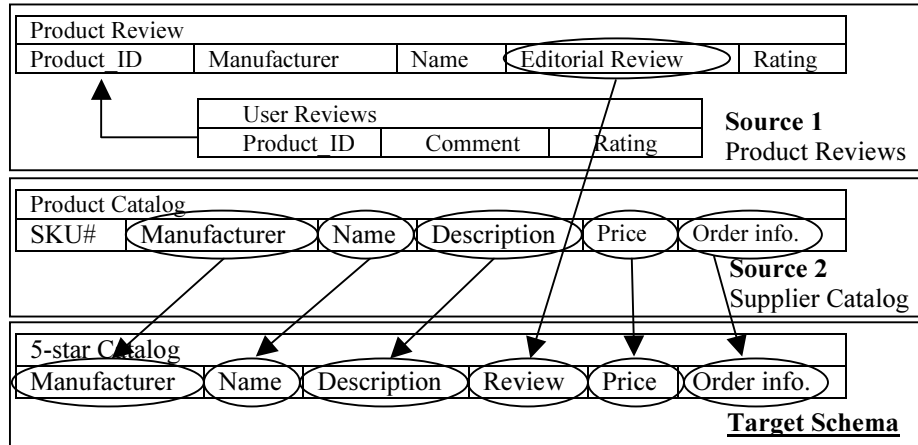


Fig. 1. Federating Product, and Review catalogs

Identical products between the **Product Review** and the **Supplier Catalog** tables should be matched with a join operation in order to populate the target schema with product listings and their corresponding reviews. There are three main possibilities to join **Product Review** and **Product Catalog**: (SKU# \rightarrow Product ID), (Name \rightarrow Name) and (Manufacturer, Name \rightarrow Manufacturer, Name). A fourth possible join is a right outer join, which will map products listed in the catalog, even if no matching reviews can be found. Further, as the name '5-star Catalog' indicates, only products with '5-star' ratings in their reviews should populate the target schema.

Consider four of the many possible view definitions (Figure 2). To determine which of those alternatives is correct is to precisely answer the questions raised above about the nature of the join between **Product Reviews** and **Supplier Catalog**.

3 Search Space for Query Discovery

Sphinx explores a hypothesis space, containing all the possible queries under consideration. The exploration of this space will yield the correct query. We propose two methods to initialize this search. The first method we present allows us to merge an initial set of possible view definition into a covering space. This space will contain each of those view definitions, as well as points in between representing combinations. We also present a method, which initializes a default search space starting from a user defined data-mapping example, rather than an initial set of view definitions.

<pre> Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed- Review, PC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Product_ID = SC.SKU# and PR.rating="5" </pre>	<pre> Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, SC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Name = SC.Name and PR.Manufacturer = SC.Manufacturer and PR.rating="5" </pre>
<pre> Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed- Review, PC.Price, PC.OrderInfo From Product Review PR, Supplier Catalog SC Where PR.Name = SC.Name and PR.rating="5" </pre>	<pre> Select SC.Manufacturer, SC.Name, SC.Description, PR.Ed-Review, SC.Price, SC.OrderInfo From Product Review PR, Supplier Catalog SC Where [(PR.Name = SC.Name and PR.Manufacturer = SC.Manufacturer) OR ((SC.Name, SC.Manufacturer) not in (Select PR.Name, PR.Manufacturer From Product Review))] and PR.rating="5" </pre>

Fig. 2. Four possible view definitions.

3.1 Initialization by Query merging

Let us consider a set of view definitions VD_1, VD_2, \dots, VD_n . We write each of those view definitions as an algebra sentence using projections, selections and a Cartesian product. We assume that all those view definitions are compatible with the same attribute to attribute schema mapping (such as in Figure 1). In practice this assumption means that each of those view definitions has the same projection operator to the target view $\Pi(a_1, \dots, a_t)$, and that the attributes involved originate from the same set of relations. Taking into account the bag semantics of the Cartesian product, we can perform a bag maximum operation [10] of the Cartesian products in all view definitions, so that each VD_1, VD_2, \dots, VD_n can be rewritten using a common Cartesian product $O_1 \times O_2 \times \dots \times O_c$ as shown in (1). In a calculus (or SQL) sentential form, the projected attributes will then originate from the same variable assignment.

$$\begin{aligned}
VD_1 &= \Pi(a_1, \dots, a_t) (\sigma_{1,1} \circ \sigma_{1,2} \circ \dots \circ \sigma_{1, \lambda(1)} (O_1 \times O_2 \times \dots \times O_c)) \\
VD_2 &= \Pi(a_1, \dots, a_t) (\sigma_{2,1} \circ \sigma_{2,2} \circ \dots \circ \sigma_{2, \lambda(2)} (O_1 \times O_2 \times \dots \times O_c)) \\
&\dots \\
VD_n &= \Pi(a_1, \dots, a_t) (\sigma_{n,1} \circ \sigma_{n,2} \circ \dots \circ \sigma_{n, \lambda(n)} (O_1 \times O_2 \times \dots \times O_c))
\end{aligned} \tag{1}$$

$$\begin{aligned}
VD(F_1, F_2, \dots, F_{pf}) &= \Pi(a_1, \dots, a_i) \\
& (F_1 \sigma_{1,1} \circ F_2 \sigma_{1,2} \circ \dots \circ F_{\lambda(1)} \sigma_{n, \lambda(n)} \\
& F_{\lambda(1)+1} \sigma_{2,1} \circ F_{\lambda(1)+2} \sigma_{2,2} \circ \dots \circ F_{\lambda(1)+\lambda(2)} \sigma_{2, \lambda(2)} \\
& \dots \\
& F_{\Sigma\lambda(i)+1} \sigma_{n,1} \circ F_{\Sigma\lambda(i)+2} \sigma_{n,2} \dots F_{pf} \sigma_{n, \lambda(n)} (O_1 \times O_2 \times \dots \times O_c)
\end{aligned} \tag{2}$$

$$\begin{aligned}
VD_1 &= VD(F_1=True, F_2=True, \dots, F_{\lambda(1)}=True, False, False, \dots, False) \\
VD_2 &= VD(False, \dots, False, F_{\lambda(1)+1}=True, \dots, F_{\lambda(1)+\lambda(2)}=True, False, \dots, False) \\
&\dots \\
VD_n &= VD(False, \dots, False, F_{\Sigma\lambda(i-1)}=True, \dots, F_{\Sigma\lambda(i)}=True)
\end{aligned} \tag{3}$$

In (2), we merge those view definitions into a formula parameterized by Boolean variables F_1, F_2, \dots, F_{pf} . Each of those Boolean variables F_i is combined with a selection operator σ_i , such that σ_i is omitted if and only if F_i is false. In (3), each of the original view definitions is expressed using the new formula, for some Boolean assignment of the variables F_i .

The resulting formula: $VD(F_1, \dots, F_{pf})$ represents the search space of possible view definitions. The learning algorithm will seek to converge by finding the correct assignments to each Boolean variable (also called feature) F_1, F_2, \dots, F_{pf} .

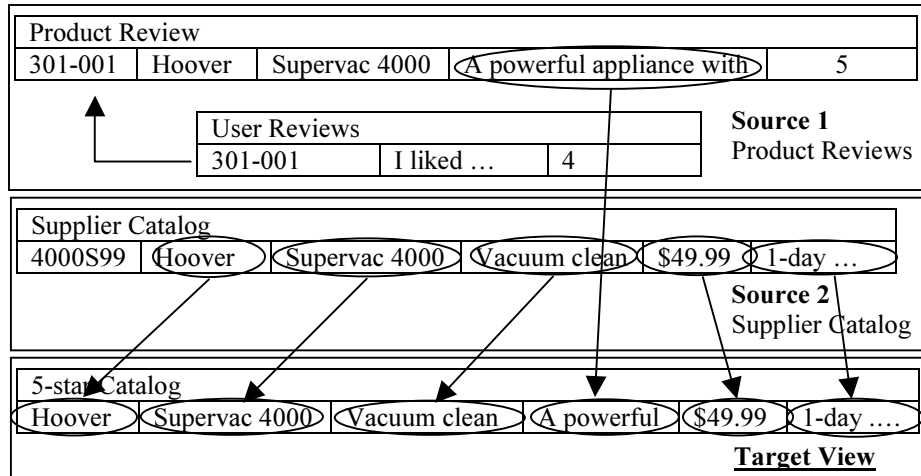


Fig. 3. A data-mapping example.

3.2 Default Initialization

A search space can also be initialized using an initial data mapping as shown in Figure 3. That data-mapping example is constructed from data values taken in displayed source tables using a QBE-like, point and click interface. This does not require any abstraction skills from the user. However we assume that as a domain expert, the user knows and understands the data sufficiently to produce a correct mapping. This data mapping immediately gives us the schema mapping of attributes from source to tar-

get. However, it also contains more information than the schema mapping, because it is grounded with actual data instances from each source. This enables us to initialize a search space for query discovery.

With the following steps, we define a parameterized Boolean formula (4), which will represent the initialized search space:

$$VD(F_1, \dots, F_{pf}) = \Pi(a_1, \dots, a_i) (F_1 \sigma_1 \circ \dots \circ F_{pf} \sigma_{pf} (O_1 \times O_2 \times \dots \times O_c)) \quad (4)$$

Steps:

1. Introduce a Cartesian product between all the tables (O_1, O_2, \dots, O_c), which appear in the data-mapping example.
2. Introduce the projections necessary to form the federated (target) view from the Cartesian product: $\Pi(a_1, \dots, a_i)$.
3. Create equality selection predicates $\sigma_1, \sigma_2, \dots, \sigma_k$ for every value in the data mapping.
4. Create equality join predicates $\sigma_{k+1}, \sigma_{k+2}, \sigma_{pf}$ for every pair of values in the data mapping which are in distinct tables and are equal.

We detail these steps for the data-mapping example illustrated in Figure 3:

- The Cartesian product is **Product Reviews** x **Supplier Catalog**
- It follows that the projection operator is $\Pi_{(SC.Manufacturer, SC.Name, SC.Description, PR.Ed.Review, SC.Price, SC.Orderinfo)}$
- The selection predicates are σ_1 : “PR.SKU = ’301-001’”, σ_2 : “PR.Manufacturer = ’Hoover’”, ..., σ_{11} : “SC.Orderinfo = ’1-day shipping’”.
- The join predicates σ_{12} : “PR.Manufacturer = SC.Manufacturer” and σ_{13} : “PR.Name=SC.Name”.

This data-mapping example immediately excludes a join on attributes Product_ID and SKU# and no corresponding predicate is created.

The default search space defined in (4), while fairly general, is not complete. All legal equality predicates consistent with the initial data mapping are generated but any other arbitrary predicates are not. Selection predicates are preferred to self-joins. This has been sufficient for applications in both e-commerce and bioinformatics, due to the nature of the mappings between databases in similar domains [17]. As determined by ongoing experience, additional features may be added directly to the initialization of the search space or by expansion of the user interface (we expect outer-joins and query paths may be added to the learnable feature set without greatly expanding the search space). Learning inequality predicates with arbitrary threshold values is likely to be intractable, or even undecidable [18], and thus such functionality must be added by expanding the user interface.

4 User Interaction Algorithm

Many types of mappings necessary to perform database integration can be represented easily in a user interface. This is the case with the mappings in previous sections, but

also with synonym mappings, and unit conversions. Building such dictionaries can be fastidious, but the basic process is simple and can be grasped by anyone familiar with the data. The goal of Sphinx is to propose a user interaction model for those semantic properties, which cannot be easily expressed in a simple user interface format. Sphinx does not ask the user to understand abstract specifications of view definitions, whether they are expressed in an abstract language (such as SQL) or expressed as symbols in a diagram. To solve the query discovery problem, Sphinx will limit its interaction to a simple and rigid framework. The only kind of interaction permitted with the user is showing several elements of input data combined together to form a row of output (e.g. Figure 3). The user must simply examine the output data instance and decide if it belongs to the target view.

4.1 User Interaction Model

Our user interaction framework takes away from the user the burden of making complex or abstract decisions. In particular, it is the active learning, which will construct and choose new data instances to submit them to the user. The Versions Spaces model will keep track of the search space exploration, allowing the learning algorithm decide when the correct view definition has been derived and end the process. The performance of the system is measured in terms of the burden imposed on the user, by counting the number of examples, or questions submitted before termination.

4.2 Question and Answers game

We leave all formal expositions of the Sphinx algorithm to the full version of this text [2]. Instead, we will seek to illustrate here, with a simple question and answer game, how Sphinx modifies Version Spaces. The reader familiar with Mitchell's Version Spaces [19] will recognize some of the concepts exposed here.

Consider a database containing descriptions of a group of people, where each person has exactly three characteristics: first name, married or single, and male or female. This constitutes a source database. The user chooses a target group of people, for which the system will try to state a definition. A set of features will characterize the target group's definition. The system tries to deduce what those features are by asking if certain individuals in the database are in the target group.

The user starts by declaring that John, single, male is an individual in the target group. The system builds a search space, inventing a predicate for each feature F_1 , F_2 and F_3 . F_1 has "name = John", F_2 has "is single", F_3 has "is male". These are the only 3 *equality* predicates, which may filter the target group from the database without excluding John. We define a feature vector of size 3, for queries q on the database: (q_1, q_2, q_3) . Our search space is illustrated in Figure 4(a): there are 8 delimited areas. Each area corresponds to a truth assignment of the feature vector and a possible target group.

As the question and answers game progresses, possibilities for the target group's feature vector are eliminated. There are only three situations, which can lead to a reduction in the size of the search space. We illustrate each with an example.

4.2.1 Positive Example Convergence Rule

The system asks the user if ‘Mary, single, and not male’ is in the target group and the user answers yes. The system deduces that F_1 and F_3 are not in the target group’s feature vector. Either of those features would eliminate Mary from the target group, so they are excluded. In Figure 4(b), the excluded areas for the target group are grayed out. The search space is reduced to the non-gray areas: its size is now 2, since the only possibilities left for the target group’s feature vector are $(0, 1, 0)$ and $(0, 0, 0)$.

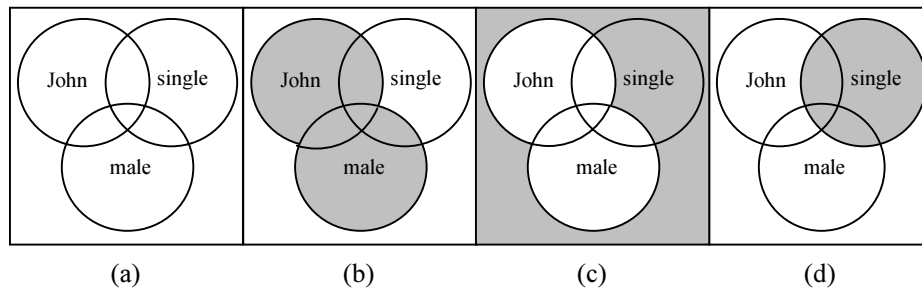


Fig. 4. (a) Search space for Q&A. (b) Positive example: Mary, single, not male. (c) Negative example: Mary, single, not male. (d) Cannot find a married person in the database.

4.2.2 Negative Example Convergence Rule

Assume the game is in its initial state again, and that the system asks the user if ‘Mary, single, and not male’ is in the target group and the user answers no. The system can deduce that either F_1 or F_3 must be in the target group’s feature vector. The exclusion of both would put Mary in the target group. In Figure 4(c), the excluded areas for the target group are grayed out. The search space is reduced to size 6, since the only possibilities excluded for the target group were $(0, 1, 0)$ and $(0, 0, 0)$

4.2.3 Missing Example Convergence Rule

Assume the game is again in its initial state. Sample selection, limits the system to asking questions about people who are listed in the source database. Assume that the system discovers that all the people in the database are single. In that case we can say that feature F_2 is indifferent. The feature is indifferent because its inclusion in the target group’s feature vector makes no difference as to the final composition of the target group: there are no married people to be included or excluded from the target group. Since F_2 is indifferent, we decide to eliminate all possibilities in the search space, which do not include F_2 . Any correct definition of the target group which does not include F_2 , will remain correct when F_2 is added. As illustrated in Figure 4(d), this halves the size of the search space to 4.

We arbitrarily excluded 4 *example feature vectors*: $(0, 1, 0)$, $(0, 1, 1)$, $(1, 1, 0)$, $(1, 1, 1)$ for which no instances could be exhibited. Those 4 example feature vectors are labeled missing, and together, those four lead to a reduction of search space from 8 to 4 possible definitions of the target group.

5. Correctness and Limitations

The Question & Answer game mirrors the user interaction process used in Sphinx. The target group the system is trying to learn represents the federated view definition (the target view or target query) and the people used as examples represent the data-mapping examples, which the user will label as positive or negative. It should be apparent to the reader that the Question&Answer game above is sure to converge to a correct result within a finite number of steps, provided that the correct view definition was in its initial search space. In the general case, we prove in extenso, that with its modifications to Versions Spaces, Sphinx remains a consistent learner [2]. Further, because of the three convergence rules, progress is guaranteed, since every possible vector in the hypothesis space can always be assigned one of three labels. When all 2^{pf} vector values have been labeled, the algorithm will have converged, which guarantees termination in a finite number of steps.

Missing labels are the result of dependencies in the data: examples necessary to discriminate between two competing hypothesis cannot be instantiated with existing data. When we identify such cases when competing hypothesis cannot be disambiguated, it is possible to generate a trigger equivalent to the disambiguating example. If the trigger is activated by changes in the underlying data sources, then the conflicting example can be submitted to the user. The target concept will be disambiguated and the trigger removed. This does not require a new initialization of the learning algorithm, but rather supposes a simple additional user interaction step.

Adopting the Version Spaces model limits Sphinx to learning queries which can be represented by a vector of binary features. However, it is possible to represent any arbitrary predicate, as long as the WHERE clause of the query is formed by a conjunction of such predicates (i.e. expressed in CNF). On the other hand, SQL queries represented with Version Spaces have no flexibility in their SELECT and FROM clauses: competing schema mappings cannot be evaluated since the projection operator must remain a common denominator when queries are merged. For this reason, view definitions with competing aggregation/sorting operators or with nested sub-queries cannot be handled by Sphinx, unless those operators and nested queries appear exclusively within a predicate in the WHERE clause. In Figure 2, one of the queries contains an outer join, which is represented using a nested sub-query inside of a predicate. Effectively Sphinx can represent both an inner and an outer join simultaneously, and will treat them as competing hypotheses, finding examples to discriminate between the two. Although disjunctions may appear inside query predicates, Version Spaces representation does not handle UNION on the same level as conjunctive predicates. Likewise there is no explicit handling of negation.

6. Sample Selection

The search space considered by Sphinx is always finite, and we guaranteed that the algorithm would converge to a result in a finite number of steps. However the size of the search space is 2^{pf} , where pf is the cardinality of the potential feature set. Thus, the challenge is to limit the number of examples (positive or negative) presented to the

user during the search. To a lesser degree limiting missing examples is also important since they require computation time. Thus we look at the active learning and sample selection strategy, which is the part of the system selecting the questions in the Q&A game.

Any strategy wishing to conclude the Q&A phase quickly, will select examples, which shrink the search space by as much as possible. As illustrated earlier, there is a tremendous difference in value between positive, negative and missing examples. To be precise, a positive example negating k features, reduces the size of the search space from 2^{pf} to 2^{pf-k} . In contrast reducing the search space from 2^{pf} to 2^{pf-1} requires exactly 2^{pf-1} missing examples. It can be shown that a strategy collecting no positive examples would require 2^{pf} examples (missing or negative) to converge: an ineffective method. Thus any strategy must focus on finding at least one, possibly several, examples labeled *positive* by the user.

In order to obtain a positive example, the system should seek to present to the user an example where only a very unlikely set of features are negated. By negating only a set of very unlikely features, the system can effectively bet that it has produced a positive example. Drawing an analogy in our Q&A game, assume the system can estimate that “Name= John” is unlikely to be a defining feature of the target group. In that case, the system would try to obtain a positive example by asking the user if “Paul”, single, male is in the target group. The rationale here is that “Paul” or “John”, the first name is unlikely to determine membership in the target group.

6.1 Join Feature Bias

In the rest of our analysis we proceed to motivate and experiment with heuristics to explore the initial default search space in particular, rather than one built by query merging. This allows us to use Sphinx as a standalone system. And, as the goal is to explore a large space with an acceptable amount of user interaction, the search space generated by default presents a greater challenge, because of all the extraneous predicates, which are generated in the default method. Query merging, producing search spaces containing far fewer features, represents the less challenging of the two problems.

A simple count shows that the overwhelming majority of the potential features are selection predicates (Table 1). Most production databases contain a very large number of attributes, and for each of those, a selection predicate is generated. A join feature is created only when supported by the initial data-mapping example: a relationship exists between two objects in the data mapping.

An accidental match between a “9.99” as the price \$9.99 and a “9.99” as September 99 is possible, but is an unlikely event given any pair of objects chosen at random by the user. Thus almost all join features observed in the data-mapping example are not flukes and do represent existing semantic relationships. Thus two factors combine here to privilege join features: a pure Cartesian product without a join predicate is a very unlikely operation, and most observed join features between objects belonging to different tables are not accidental.

We elaborate a baseline strategy S_b based on this observation. In its initial phase S_b searches for positive examples by choosing sub-goals that do not negate any potential

join predicates. Instead, strategy S_b will search for examples, which negate a small given set of selection predicates (never more than 10). The search progresses by modifying the set until a positive example is found or the algorithm converges. The small set of negated selection features is chosen with an initial randomization and modified in an incremental and randomized search pattern (the choice of the latter having no observed effect on the success of the search). Once positive examples have been found, S_b enters its second phase, and searches for examples in which both join and selection predicates will be negated.

It should be observed that in addition to its bias for join features vs. selection features, strategy S_b possesses another built-in bias: it consistently bets that of all the potential features (a large number), only a very small number are likely to appear in the *Where* clause of the target query. This is consistent with our ad-hoc observation in the course of solving schema integration problems, that federating view definitions rarely contain selection predicates, if at all.

6.2 Information Gain Bias

We make the observation that not all features are equally likely. Consider the following feature predicate: “Name = ‘Supervac 4000’”. It is unlikely to appear in a view definition because few rows in the source, perhaps only one, will fulfill that predicate. On the other hand a feature such as “Rating = ‘5-star’” is more likely. A significant portion of the rows may well fall in the ‘5-star’ category and building a view with those rows might be of use.

Using catalog statistics, we can measure for each selection predicate their information gain. The basic assumption is that the information gain will serve to estimate the likelihood of a predicate. A selection predicate with very low information gain will split the rows in the database in two groups, one of which is very small (only a few rows), the other very large (all the other rows). A selection predicate with very high information gain will split the rows into groups of comparable size. The information gain is maximal for a predicate, which splits the rows into two groups of equal size. A comparable bias could be introduced to estimate the likelihood of individual join predicates, however the overall small number of join predicates precludes the need.

We introduce a new strategy S_c , a refinement of S_b based on this information gain bias. This strategy does not require likelihood estimations to be accurate. The likelihood function should, above all, cluster predicates into two major categories: the most unlikely predicates (very low information gain), and the other predicates (low to high information gain). This clustering will replace the random process used in S_b .

6.3 Experiments

We implemented a prototype system with a graphic-user interface. This prototype handles the kind of data mapping instances presented here, as well as mappings, in which meta-data elements are mapped to data [12]. This small higher-order generalization allows for a broader range of restructuring queries across schematically disparate sources, without any substantial changes to the overall system.

We chose four domains to experiment with data integration. The first three problems were actual internet database integration tasks conducted under contract, in an ad-hoc fashion by a web services consulting firm. Almost all source data was derived from live websites in HTML form. The source sites were wrapped to produce structured results. Those schema integration tasks are created by experimenting with Sphinx, and are ranked in Table 1, by increasing level of empirical complexity. In the first domain the target queries populate a Healthcare provider directory database merging two data sources. The second domain is based on a sport statistics databases for web publishing. The third domain is the ‘5 Star Catalog’ for electronics and comes from the area of online price monitoring for B2B merchandise distributors. A simplified version of that schema is used as an illustrative example throughout this exposition. The fourth domain is based on a bioinformatics integration effort to combine the functionalities of two application platforms for handling microarray data: SMD (Stanford Microarray Database) and BASE (BioArray Software Environment). Both these production systems carry out the analysis of gene expression microarrays in various laboratories throughout the world.

Each test set includes source databases, a target schema, and two interesting target queries populating different tables of the target schema. The number of features (i.e. Selection and Join predicates) appearing in the *Where* clause of each query is shown (e.g. 1J, 0S: 1 Join and no Selection predicates) in the *Query Size* column (Table 2). The number of examples Sphinx requires to reach the target query is averaged over ten runs for each query and shown in Table 2. The decimal averages are a product of random factors present in both heuristics and sample selection.

Table 2 results show the randomized strategy S_b is sometimes slightly more successful than the refined strategy S_c for target queries which do not have a selection predicate in their *Where* clause. The performance of S_b degrades quickly when the target query contains even a single selection predicate and can fail in large search spaces. S_c shows more stability, its performance slowly decreases when the complexity of the target query increases. This is because S_c uses data catalog statistics to differentiate among the selection features, and picks those which can be excluded from the target query with high probability.

Table 3 compares the performance of both active learning strategies for Sphinx with two experiments in which Sphinx is hobbled to become a passive learning system. These two experiments do not represent valid strategies but are designed to identify bounds, both lower (oracle) and upper (random) on the number of examples an active learning algorithm may require. In these passive learning experiments, the user carries the burden of constructing data mapping examples as well as labeling them. Sphinx merely indicates when the system has converged to a target query. To measure a lower-bound, an omniscient user (us), constructed an optimal sequence of examples to converge Sphinx as quickly as possible. To measure an upper-bound a naive user chooses examples at random from the set of possible data mappings. At each step there is an equal probability of a positive or a negative example being chosen. Each example is picked randomly from its respective population of positive or negative examples. Unlike the oracle the random user is not in a feedback loop with the learning system, and does not know which examples need to be picked next in order to finish converging the system.

We observe that the number of required examples spikes when predicates are added to the target query, and that regardless of the number of potential features, the simplest target queries require a small number of examples. These experiments suggest that even with imperfect heuristics, the observed complexity is mostly correlated with the size and complexity of the target query rather than with the number of potential features. This is an unusual result for a machine-learning algorithm, due to the extremely specific and well-defined nature of our target concepts.

Table 1. Selection vs. Join Features

		Potential Features	Potential Selection Features	Potential Join Features
Healthcare	Query 1	14	14	0
Directory	Query 2	15	15	0
Sports Sta-	Query 3	25	24	1
tistics	Query 4	25	24	1
5 Star	Query 5	30	28	2
Catalog	Query 6	30	28	2
SMD →	Query 7	57	57	0
Base	Query 8	35	33	2

Table 2. Active learning experiments with Sphinx

		Strategy S_c				Strategy S_b		
Query Size	Potential Features	Number of Examples			Number of Examples			
		Total	Pos.	Neg.	Total	Pos.	Neg.	
Query 1	0J, 0S	14	2.2	2.2	0.0	1.4	1.4	0.0
Query 2	0J, 1S	15	4.7	2.0	2.7	5.5	2.5	3.0
Query 3	1J, 0S	25	4.9	3.7	1.2	2.5	1.5	1.0
Query 4	1J, 1S	25	4.8	1.8	3.0	11.4	1.6	9.8
Query 5	2J, 0S	30	5.9	3.2	2.7	4.0	2.0	2.0
Query 6	2J, 1S	30	8.9	2.7	6.2	11.7	2.0	9.7
Query 7	0J, 0S	57	4.5	4.5	0.0	x	x	x
Query 8	1J, 0S	35	3.2	2.2	1.0	3.0	2.0	1.0

x – interactive search did not complete in a reasonable amount of time

Table 3. Active learning vs. Passive learning.

	S_c	S_b	Oracle			Random		
	Total	Total	Total	Pos.	Neg.	Total	Pos.	Neg.
Query 1	2.2	1.4	1	1	0	26	13	13
Query 2	4.7	5.5	2	1	1	26	13	13
Query 3	4.9	2.5	2	1	1	7.5	3.5	3.0
Query 4	4.8	11.4	3	1	2	22	11	11
Query 5	5.9	4.0	4	1	3	12	5.5	6.0
Query 6	8.9	11.7	5	1	4	42	21	21

7. Related Work

Mediator-based architectures to federate heterogeneous databases have drawn a lot of interest [7], [8], [15], [25], [26], [28] to cite a few. In these systems, the basic assumption is that some highly qualified engineers may become domain experts and in one form or another write the specifications that will drive the data integration. In that line of work, several general-purpose query languages for specifying heterogeneous data integration include SchemaSQL and SchemaLog [13], and XQuery [27]. As shown by Krishnamurthy, Litwin and Kent [14], such languages must possess higher order features to bridge schematic heterogeneities across data sources [11], [28]. Semantic schema integration methods take into account semantic relationships represented as first order assertions, and correspondences between schema elements [1], [9], [12], [20], [22], [26] or can be built into meta-dictionaries of data and meta-data elements [3]. In both cases federated schema integration can be derived from semantic descriptions.

With the maturation of those systems, the problem of generating semantic specifications to federate data sources garnered more attention. Milo and Zohar [17] observe that the vast majority of mappings between schema elements in heterogeneous databases are trivial and lend themselves to automation. Thus, automated schema matching tools were developed with the goal of helping an engineer cope with the plethora of domain information, which must be reconciled to federate heterogeneous databases. Most mappings can be derived automatically, and user expertise can be saved for a smaller number of truly complex mappings. Rahm and Bernstein [21] offer a taxonomic survey of these automated schema-matching tools. While they often have high accuracy, these tools cannot guarantee that a correct mapping has been derived. A database specialist with domain expertise must examine the system's final output to verify the correctness. The average non-technical user cannot be expected to use the advanced query or semantic modeling languages commonly used to express those mappings. Thus because their founding principles in terms of input and expected output (as well as projected application) are so radically different from the approach proposed by Sphinx and Clio, it is impossible to compare the performance of those systems with ours by any measure.

In machine learning, the Version Spaces algorithm [19] is often associated with domains where the input data is noiseless. To our knowledge, we are the first to exploit it in the context of databases. Cohn, Atlas and Ladner [4] proposed a practical approach where the learning algorithm controls the selection of labeled examples that it is learning from. This approach greatly improves the algorithm's capacity to learn from a fixed number of examples, and has been applied to other machine learning systems [5], [15], [16], [24]. The exploration of formal learning theory related to this active form of learning has also been the subject of much work [6], [23].

8. Conclusion

Many challenges of data integration such as attribute correspondence and unit conversion can be solved with point and click interfaces. If this were enough to solve the

general problem of database integration the subject would be closed. Sphinx addresses the open issue of schema integration where both simple and potentially complex queries over multiple data sources are required to populate the federated views.

We built a Version Spaces model for query discovery by example and developed the Sphinx learning algorithm, by adding a new kind of label and a new learning rule to the two labels and the two rules in the original Version Spaces algorithm. This new algorithm allows full and accurate verification by a user of potential mappings of semantic relationships. This is accomplished entirely by example, where only the initial data-mapping example needs to be supplied by the user. The active learning and sample selection system incorporated in Sphinx generate additional examples, which are labeled positive or negative by the user. We present a search heuristic, which is greatly improved by the use of catalog statistics to minimize the number of examples submitted to the user.

Ongoing work can quickly address a wide range of practical issues in schema integration within the framework we established. Currently Sphinx relies entirely upon the user to provide semantic knowledge and does not seek to derive it from context or from existing data. This strict split between the user's and system's responsibilities was motivated by our specific goals. However, by incorporating additional user interaction, Sphinx could easily initialize a more sophisticated default search space producing a more complete set of view definitions. Simultaneously, by incorporating more data mining and reverse engineering techniques, Sphinx could exploit better heuristics in order to navigate the resulting expanded search space.

Future work could also seek to address those features, which cannot be handled by the current Version Spaces framework. While not strictly necessary, the inclusion of a UNION operator would significantly expand the ability of Sphinx to handle a large set of real world problems. As for handling xml data, nested relational queries with competing levels of nesting cannot be compared with Sphinx. They require the introduction of the NEST and UNNEST operators in the algebra, which is beyond the expressive power of the current Boolean representation.

References

1. Arens Y., C. Knoblock, WM. Shen (1996): Query Reformulation for Dynamic Information Integration. *JIS* 6(2/3): 99-130.
2. Barbaçon F., D. Miranker (2004): Active Learning of Schema Integration Queries. The University of Texas at Austin, Dept. of Computer Sciences Tech. Report CS-TR-04-23 (submitted for publication).
3. Chen Y., W. Benn: Building DD to Support Query Processing in Federated Systems. *KRDB* 1997: 5.1-5.10.
4. Cohn D., L. Atlas, R. Ladner: Improving Generalization with Active Learning. *Machine Learning* 15(2): 201-221 (1994).
5. Dagan I., S. Engelson: Committee-Based Sampling for Training Probabilistic Classifiers. *ICML* 1995:150-157.
6. Gasarch W., C. Smith: Recursion Theoretic Models of Learning: Some Results and Intuitions. *Annals of Mathematics and Artificial Intelligence*, 15:151-166 (1995).

7. Garcia-Molina H., Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom: The TSIMMIS Approach to Mediation: Data Models and Languages. *JIIS* 8(2): 117-132 (1997)
8. Haas L., D. Kossman, E. Wimmers, J. Yang: Optimizing Queries Across Diverse Data Sources. *VLDB* 1997: 276-285
9. Johannesson P.: Using Conceptual Graph Theory to Support Schema Integration. *ER* 1993: 283-296.
10. Kent W. (1992): Profile Functions and Bag Theory. Hewlett-Packard Company. Technology Department, Hewlett-Packard Laboratories. Palo Alto, California.
11. Kent W.: Solving Domain Mismatch and Schema Mismatch Problems with an Object-Oriented Database Programming Language. *VLDB* 1991: 147-160
12. Krishnamurthy R., W. Litwin, W. Kent: Language Features for Interoperability of Databases with Schematic Discrepancies. *SIGMOD Conference* 1991: 40-49.
13. Lakshmanan L., F. Sadri, I. Subramanian: SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. *VLDB* 1996: 239-250
14. Levy A., A. Rajaraman, J. Ordille: Querying Heterogeneous Information Sources Using Source Descriptions. *VLDB* 1996: 251-262.
15. Lewis D., J. Catlett: Heterogeneous Uncertainty Sampling for Supervised Learning. *ICML* 1994:148-156.
16. Liere R., P. Tadepalli: Active Learning with Committees for Text Categorization. *AAAI* 1997: 591-596.
17. Milo T., S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. *VLDB* 1998: 122-133.
18. Miranker D., M. Taylor, A. Padmanaban: A Tractable Query Cache by Approximation. *SARA* 2002: 140-151.
19. Mitchell T.: Version Spaces: A Candidate Elimination Approach to Rule Learning. *IJCAI* 1977: 305-310
20. Mitra P., G. Wiederhold, M. Kersten: A Graph-Oriented Model for Articulation of Ontology Interdependencies. *EDBT* 2000: 86-100
21. Rahm E., P. Bernstein: A survey of approaches to automatic schema matching. *VLDB Journal* 10(4): 334-350 (2001).
22. Spaccapietra S., C. Parent, Y. Dupont: Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal* 1(1): 81-126 (1992).
23. Stephan F.: Learning via Queries and Oracles. *COLT* 1995:162-169.
24. Thompson C., M. Califf, R. Mooney: Active Learning for Natural Language Parsing and Information Extraction. *ICML* 1999: 406-414.
25. Tomasic A., L. Raschid, P. Valduriez: Scaling Heterogeneous Databases and the Design of Disco. *ICDCS* 1996: 449-457.
26. Vassalos V., Y. Papakonstantinou: Describing and Using Query Capabilities of Heterogeneous Sources. *VLDB* 1997: 256-265.
27. XML Query: <http://www.w3.org/TR/XQuery>
28. Yan L., M. Özsu, L. Liu: Accessing Heterogeneous Data Through Homogenization and Integration Mediators. *CoopIS* 1997: 130-139
29. Yan L., R. Miller, L. Haas, R. Fagin: Data Driven Understanding and Refinement of Schema Mappings. *SIGMOD Conference* 2001.