Corpus-based Schema Matching

Jayant Madhavan
University of Washington
jayant@cs.washington.edu

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

AnHai Doan
UIUC
anhai@cs.uiuc.edu

Alon Halevy
University of Washington
alon@cs.washington.edu

Abstract

Schema Matching is the problem of identifying corresponding elements in different schemas. Discovering these correspondences or matches is inherently difficult to automate. Past solutions have proposed a principled combination of multiple algorithms. However, these solutions sometimes perform rather poorly due to the lack of sufficient evidence in the schemas being matched. In this paper we show how a corpus of schemas and mappings can be used to augment the evidence about the schemas being matched, so they can be matched better. Such a corpus typically contains multiple schemas that model similar concepts and hence enables us to learn variations in the elements and their properties. We exploit such a corpus in two ways. First, we increase the evidence about each element being matched by including evidence from similar elements in the corpus. Second, we learn statistics about elements and their relationships and use them to infer constraints that we use to prune candidate mappings. We also describe how to use known mappings to learn the importance of domain and generic constraints. We present experimental results that demonstrate corpus-based matching outperforms direct matching (without the benefit of a corpus) in multiple domains.

1 Introduction

Semantic heterogeneity is a key problem in any data sharing system, be it a federated database [24], data integration system [27], message passing system, web service, or peer-data management system [20]. The data sources involved are typically designed independently, and hence use different schemas. To obtain meaningful inter-operation, one needs a semantic mapping between the schemas, i.e., a set of expressions that specify how the data in one source corresponds to the data in the other.

This paper considers the problem of *schema matching*: determining a set of *correspondences* (a.k.a. *matches*) that identify similar elements in different schemas. Matching is typically the first phase in generating schema mappings. Schema matching is inherently a difficult task to automate mostly because the exact semantics of the data are only completely understood by the designers of the schema, and not fully captured by the schema itself. In part, this is due to the limited expressive-power of the data model, and often is further hindered by poor database design and documentation. As a result, the process of producing semantic mappings requires a human in the loop and is typically labor-intensive, causing a significant bottleneck in building and maintaining data sharing applications.

Schema matching (a.k.a ontology alignment) has received steady attention in the database and AI communities over the years (see [21] for a recent survey and [19, 1, 4, 6, 3, 12, 13, 16, 29, 26] for work since). A key conclusion from this body of research is that an effective schema matching tool requires a principled combination of several *base techniques*, such as linguistic matching of names of schema elements, detecting overlap in the choice of data types and representation of data values, considering patterns in relationships between elements, and using domain knowledge.

However, current solutions are often very brittle. In part, this is because they only exploit evidence that is present in the two schemas being matched. These schemas often lack sufficient evidence to be able to discover matches. For example, consider table definitions `T1` and `T2` in Figure 1(a). While both of these tables describe the availability of items, it is almost impossible to find a match by considering them in isolation.

This paper describes *corpus-based matching*, an approach that leverages a corpus of schemas and mappings in a particular domain to improve the robustness of schema matching algorithms. A corpus offers a storehouse of *alternative representations* of concepts in the domain, and therefore can be leveraged for multiple purposes. This paper focuses on using it to improve schema matching, and establishes some fundamental methods for building and exploiting a corpus. In analogy, the success of techniques in the fields of Information Retrieval and Natural Language Processing are based on analyzing large corpora of texts. This paper is the first step in showing that large corpora of

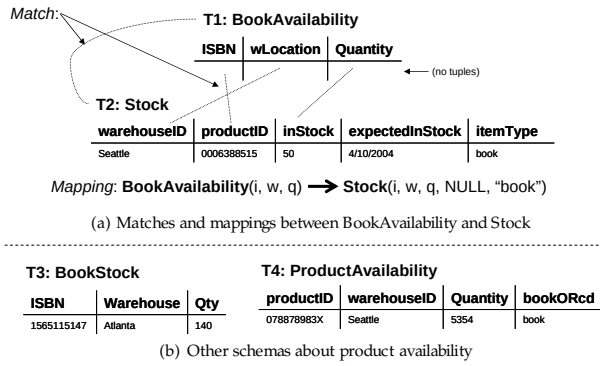


Figure 1. Schemas T1 and T2 can be better matched with the additional knowledge of schemas T3 and T4 in the same domain.

schemas can be analyzed to benefit difficult data management challenges [11].

To illustrate the intuition behind our techniques for exploiting a corpus, suppose that for any element e (e.g., table or attribute) in a schema S, we are able to identify the set of elements, C_e, in the corpus that are similar to the element e. We can use C_e to *augment* the knowledge that we have about e. In our example, the table T1.BookAvailability is very similar to the table T3.BookStock in the corpus (their columns are also similar to each other). Similarly, T2.Stock is similar to T4.ProductAvailability. It is easy to see that combining the evidence in T3 with T1 and T4 with T2 better enables us to match T1 with T2: first there is increased evidence for particular matching techniques, e.g., alternative names for an element, and second there is now evidence for matching techniques where there earlier was none, e.g., considering T3 with T1 includes data instances (tuples) where there were none initially.

Another method of exploiting the corpus is to estimate statistics about schemas and their elements. For example, given a corpus of inventory schemas, we can learn that Availability tables always have columns similar to ProductID and are likely to have a foreign key to a table about Warehouses. These statistics can be used to learn *domain constraints* about schemas (e.g., a table cannot match Availability if it does not have a column similar to ProductID). We show how to learn such constraints, and how to use them to further improve schema matching. We note that previous work has shown that exploiting domain constraints is crucial to achieving high matching accuracy, but such constraints have always been specified manually.

We note that there are many potential sources for schema corpora. Portals such as XML.org and OASIS.org list schema standards organized by domain. Some large application vendors publish their schemas. Companies often accumulate schemas and mappings from matching tasks done by their staff: a corpus-based matcher can be initialized with a small number of schemas and evolve with each new matching task.

The specific contributions of the paper are the following.

- We describe a general technique for leveraging a corpus of schemas and mappings to improve schema matching. Our technique uses the corpus to augment the evidence about elements in the schemas being matched, thereby boosting the accuracy of any schema matching technique.
- We describe how schema statistics can be gleaned from the corpus and used to infer domain constraints. We also show how the relative importance of the different constraints can be learned. This is important for their effective use in schema matching.
- We describe a comprehensive set of experiments that demonstrate that corpus-based matching has better performance than direct matching, and in particular better than previously described methods such as Similarity Flooding [16] and Glue [6]. Moreover, the improvement is even more pronounced in the case of difficult matching tasks. We also show that the constraints we learn from the corpus further improve the performance of schema matching.

Our experimental results (see Section 5.1) are based on a much larger and broader set of schemas: manually and automatically extracted web forms and small and medium-sized relational schemas, and the results are averaged over large number of matching tasks. Most previous work have either been restricted to web-forms or small schemas or have only reported anecdotal results for medium and large schemas. The use of previous schema and mapping knowledge has been proposed in the past, but in very restricted settings: to map multiple data sources to a *single* mediated schema [5], or to compose known mappings to a common schema [4]. Our goal is significantly more ambitious: we show that a corpus can be used to discover matches between two as *yet unseen* schemas.

The paper is organized as follows. Section 2 gives an overview of our approach. Section 3 describes our main corpus-based augmentation method, and Section 4 describes how to extract constraints from the corpus. Section 5 describes our experimental study. Section 6 discusses related work, and Section 7 concludes.

2 Overview

We first define the schema matching problem, and summarize the corpus-based matching approach.

2.1 Schema Matching

Semantic mappings are at the core of any data sharing architecture. Mappings are expressions that relate data in

multiple data sources and are typically expressed in some formal mapping language such as GLaV (see [14] for a survey), XQuery or SQL. Schema matching is a first step toward constructing mappings that identify elements in the disparate schemas that are related to each other. Informally, a *match* can be defined as follows: *given two schemas S and T, we say that element s in S matches t in T if the mapping between S and T has a (possibly complex) expression that relates s and t.* For example, Figure 1 shows the matches and the mapping between the tables BookAvailability and Stock. The problem of creating mappings from the match result is considered in [18, 29].

More specifically, our goal is as follows. For each element in two given schemas S and T, identify the element in the other schema that it matches, if any. Note that some elements in S can match multiple elements in T, and vice versa.

We note that while we address the problem of discovering correspondences, the basic techniques proposed can be extended to the case of richer mappings. For example, techniques proposed in [3] can be adapted to discover one-many mappings with arithmetic or string operations.

Previous schema matching algorithms: Previous schema matching algorithms can be characterized, broadly, as follows. Given two schemas, S and T, start by building models M_s and M_t for each element s in S and t in T, respectively. Each model includes all the information that is known about the element and is relevant to discovering matches, such as its name, sample data instances, type information, and related elements. A variety of matching algorithms are applied to each element pair, $\langle s \in S, t \in T \rangle$. Each algorithm compares different aspects of the elements. For example, a name matcher compares the names by computing the edit distance between the two, while a type matcher uses a data type compatibility table. The result of such a comparison is typically a similarity value. The values from different aspects are combined into a single similarity value. This step can potentially involve analysis of the results of the individual comparisons and analysis of the similarities of other element pairs. The result is a similarity matrix that has a score for each pair of elements. Matches are chosen from the similarity matrix to satisfy some criteria for a desirable mapping. The vast majority of known approaches build models for schema elements based *only* on information available in the schemas being matched. Hence, all matching decisions are essentially based on direct comparisons between the two schemas.

2.2 Corpus-based Matching

The intuition underlying corpus-based schema matching is that often direct comparison does not suffice, and we can leverage knowledge gleaned from other schemas and matches. The corpus is a collection of schemas and mappings between some schema pairs. Each schema includes its elements (e.g., relation and attribute names) and their data type, relationships between elements, sample data instances, and other knowledge that can be used to relate it to other schemas. Schemas in the corpus are only loosely related – they roughly belong to a single domain but need not be mapped to each other.

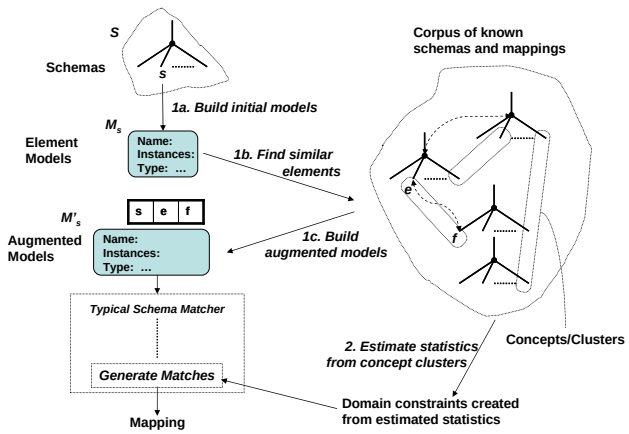


Figure 2. A corpus improves a typical schema matcher by (1) augmenting knowledge about elements in schemas, and (2) learning constraints from schema statistics to aid match generation.

Since the schemas were defined for a variety of purposes and by different designers, the corpus has a number of representations of each concept in the domain, some of which are related through the mappings in the corpus. The Corpus-Based Augment Method (augment), which we describe in the next section, leverages this variety of representations by using the corpus to add information to each element's model.

Specifically, given the element s in schema S that is not in the corpus, it finds other elements, say e and f, in the corpus that are alternate representations of the same underlying concept. Elements e and f will differ from s in some ways since they belong to different schemas. For example, e can have the same data instances as s but a different name and will thus contribute to a more general name model for s. By contrast, f can have a similar name and data instances, but different relationships with other elements in the domain and will thus contribute to a better relationship model. The augment method builds a model M'_s for s that includes knowledge about e and f, and uses M'_s in the matching process instead of M_s. The augment method is depicted in Figure 2.

Learning Schema Statistics: Another use of the corpus is to estimate various statistics about elements and relations in a domain. For example, given a collection of relational schemas, the frequently occurring tables in the schemas

give a clear indication of the important object types in a domain; the columns of multiple similar tables help us identify the various possible attributes of these objects; and the table-column and foreign key relations help us identify various possible ways in which the objects might be related. This type of knowledge will help us develop a better understanding of the domain. While these statistics can be exploited in many ways, we use them to design constraints that we use to prune candidate mappings during match generation. As we shall see in Section 4 such constraints can be used in concert with others as an input to the match generation module of a schema matcher. Previous work only exploited constraints that were given manually.

3 The Augment Method

We now describe the augment method in detail. We first describe how we find elements in the corpus that are similar to a schema element s, and then show how to use these elements to drive schema matching.

3.1 Models for corpus elements

In order to find elements in the corpus that are similar to a given schema element s, we compute an *interpretation vector*, I_s, for s. I_s is a vector, $\langle \ldots, p_{e,s}, p_{f,s}, \ldots \rangle$, where $p_{e,s}$ is an estimate of how similar s is to element e in the corpus. We use machine learning to estimate these similarities. Specifically, for each element of the corpus, e, we *learn* a model; given an element s, the model of e predicts how similar e is to s. The model for each element is created via an ensemble of *base learners*, each of which exploits different evidence about the element. An example of a base learner is a name learner that determines the word roots that are most characteristic of the name of an element (as compared to the names of other elements). Likewise, a data instance learner determines the words and special symbols (if any) that are most characteristic in instances of an element. The predictions of the base learners are combined via a *meta-learner*.

Learning the models for each element requires training data. We describe how training data is obtained and then some of the base learners that we use.

Training Data

Training a learner requires learner-specific positive and negative examples for the element on which it is being trained. For any element $s \in S$, these training examples can be extracted from S and from schemas that have known mappings to S.

- *Within* S: An element is a positive example of itself. All the other elements in the schema are negative examples, except for those that are duplicated in the schema with the same properties.
- *Outside* S: If, in some mapping, s is deemed similar to an element t in T, then the training examples for t can be added to the training examples for s. If s matches more than one element in T, then each of them contributes positive examples, and the other elements in T contribute negative examples. Mappings enable us to obtain more training data for elements and hence learn more general models.

Base Learners

We use the following base learners.

Name Learner: The name of an element typically contains words that are descriptive of the element's semantics. However, names are not always easy to exploit and contain abbreviations and special characters that vary between schemas. Our name learner tries to identify frequent word roots in the element names. We use two separate schemes. In the first we split the names of the elements based on capitalization and stem the resulting fragments, e.g., ProductAvailability becomes (Product Availability). Second, we split the names into their corresponding *n-grams* e.g., the 3-grams of the name Quantity are (qua uan ant nti tit ity). N-grams have been shown to work well in the presence of short forms, incomplete names and spelling errors that are common in schema names [4]. In many schemas in our data sets, we found that names could not be easily split into smaller sensible sub-words, and hence n-grams seemed to be especially effective. Each name fragment or n-gram set for an element contributes one training example to the name learner. The name learner can be any text classifier such as TF/IDF [23] or Naive Bayes [7].

Text Learner: Text descriptions of elements typically explain the meaning of elements more than the names themselves. But the quality of these descriptions varies a lot. We extract any text annotations that we can obtain from the elements. We also add the name fragments used for the Name Learner to the text annotations to account for overlapping information in the names and annotations. Finally, we eliminate non-significant words such as prepositions. A text classifier is trained on the resulting examples.

Data Instance Learner: The data instances of similar elements are often similar: the same values can appear, e.g., makes of cars; the same words can appear, e.g., adjectives such as good, poor, or excellent reviews; or the values might share common patterns (e.g., monetary units). For each element, we place all of its instances into a single training example. In addition to the instances themselves, we add special tokens for each symbol and for numbers. The instance learner is built by training a text classifier on these examples.

Context Learner: Similar elements are typically related to

elements that are, in turn, similar to each other. To exploit this context information, we determine for each element the set of other elements to which it is related, e.g., for columns of tables this is the table and other columns in that table. For each element we create a context example containing the tokenized names of all the elements in that set, and then train a text classifier on these context examples.

Our framework is general enough to incorporate additional base learners as well as non-learning techniques. For example, we also have a simple name comparator that uses string edit distance between the names to estimate the similarity of two elements.

Meta Learner: Given an element s, a base learner L_k makes a prediction $p^k_{e,s}$ that s is similar to the corpus element e. We combine these predictions of the different base learners into a single similarity score between e and s using *logistic regression*:

$$p_{e,s} = \sigma(\sum_k a_k p^k_{e,s} - b_k), \text{where } \sigma(x) = \frac{1}{1+e^{-x}}$$

The *sigmoid* (σ) function has been shown to have a number of desirable properties as a combination function [6]. In practice we found this to work better than a simple linear combination. The parameters a_k and b_k are learned from training separately for each element in the corpus, using the *stacking* technique [25].

Given the model created by the meta-learner, we can compute the interpretation vector, $I_s = \langle p_{e,s} \rangle$, for an element s. Recall that the interpretation vector of s has an entry for each element e in the corpus. Each $p_{e,s}$ is the estimated similarity between s and e.

3.2 Augmenting element models

The goal of the augment method is to enrich the models we build for each element of the schemas being matched, thereby improving our ability to predict matches. Our first step is to find the elements in the corpus that will contribute to the enriched model.

Given the interpretation vectors computed as described above, the most similar elements to s are picked from the interpretation vector using a combination of two simple criteria:

- **Threshold** α: Pick e such that $p_{e,s} \geq \alpha$.
- **Top N**: Pick the N elements most similar to s.

For each element s, let $C_s(\alpha, N)$ (henceforth C_s) be the elements e in the corpus that are among the N most similar to s and have $p_{e,s} \geq \alpha$.

The augmented models are constructed in a similar way to building models for each element in the corpus. Having determined C_s, we build the training set for s as follows: include as positive examples for s the union of the positive examples of s and each of the elements in C_s. The negative examples for s are the union of the different negative examples, excluding those that appear as positive examples. As described earlier, mappings in the corpus can be used to obtain more examples for any element. Thus in addition to C_s, other elements that are known to map to elements in C_s also contribute to the examples for s.

We construct the augmented model $M_s^{'k}$ for each s in a schema and base learner L_k. We note that while the training phase for elements in the corpus is done offline, the training for the augmented models is done during the matching process.

The training examples obtained from the corpus can be weighted by their similarity score in I_s. This will enable us to differentiate the relevance of the different training examples and can also be used to prevent the corpus information from dominating the examples already in s.

Since there are multiple schemas in the corpus, it is easier to find some elements that are similar to s rather that directly matching to elements in schema T. Even if a few elements are incorrectly added to the augmented model, there are benefits as long as there are fewer of them than correctly added elements.

3.3 Matching based on augmented models

The augmented models for each element are also an ensemble of models, as are the models for the elements in the corpus. We use them for matching elements $s \in S$ and $t \in T$ as follows.

For each base learner L_k, the augmented model for s can be applied on t to estimate the similarity $p^k_{s,t}$. Similarly, the augmented model for t can be applied on s to estimate $p^k_{t,s}$. The similarity $p_{s,t}$ is obtained by combining the individual $p^k_{s,t}$'s using the meta-learner for s (and similarly for $p_{t,s}$). We compute the similarity $sim(s,t)$ as the average of $p_{s,t}$ and $p_{t,s}$.

We contrast the augment method with a more naive method, pivot, which was described in our initial explorations of corpus-based matching [15]. With pivot, the similarity between two elements s and t is obtained directly from the interpretation vectors I_s and I_t by computing their normalized vector dot product (the *cosine measure*). However, pivot has two significant drawbacks. First, in many cases, the learned models are not robust because of few training examples. Second, pivot biases the matching process to rely only on the corpus, and hence ignores information in S and T that can contribute to the mapping. We compare pivot and augment experimentally in Section 5.

The result of both the augment and the pivot methods is a similarity matrix: for each pair of elements $s \in S$ and

$t \in T$, we have an estimate for their similarity (in the $[0, 1]$ range). Correspondence or matches can be selected using this matrix in a number of ways, as we describe next.

4 Corpus-aided Match Generation

The task of generating matches consists of picking the element-to-element correspondences between the two schemas being matched. As observed in previous work [16, 5], relying only on the similarity values does not suffice for two reasons. First, certain matching heuristics cannot be captured by similarity values (e.g., when two elements are related in a schema, then their matches should also be related). Second, knowledge of constraints plays a key role in pruning away candidate matches.

Constraints can either be generic or dependent on a particular domain. As examples of the latter, if a table matches Books, then it must have a column similar to ISBN. If a column matches DiscountPrice, then there is likely another column that matches ListPrice. Most prior work has used only generic constraints, and when domain constraints have been used, they have been provided manually and only in the context where there is a single mediated schema for the domain [5].

In this section we make two contributions. First, we show how to learn domain constraints automatically from the corpus. Second, we show how to use the mappings in our corpus to learn the relevance of different constraints to match generation. The latter is important because many of the constraints we employ are *soft* constraints, i.e., they can be violated in some schemas. We note that the constraints we learn from the corpus can also have other applications, such as helping the design of new schemas in a particular domain. However, here we focus on exploiting them for schema matching.

4.1 Computing Schema Statistics

To extract constraints from the corpus, we first estimate various statistics on its contents. For example, given a collection of Availability schemas, we might find that all of them have a column for ProductID, 50% of them have product information in separate tables, and 75% have Quantity and ExpectedInStock information in the same table.

In order to estimate meaningful statistics about an element, we must have a *set* of examples for that element, e.g., we can make statements about tables of Availability only when we have seen a few examples of similar tables. We now show how we group together elements in our corpus into clusters that intuitively correspond to *concepts*. We then compute our statistics in terms of these concepts.

Clustering Algorithm: We use a hierarchical clustering algorithm (outlined in Figure 3). We start with each element being a separate cluster. We iteratively try to merge the two most similar current concepts into a single concept, until no further merges are possible. For lack of space, we only highlight some of the salient features of our algorithm.

```
clusterElements (elements: set of elements from all the schemas in the corpus)
concepts = elements
for(i=1; i<= number of base learners + 1; i++)
    concepts = clusterConcepts(concepts, combinationParameters(i))
return concepts

clusterConcepts (iConcepts: input set of concepts,
                 m_c: base learner combination parameters)
train ensemble of base learners for each concept in iConcepts
sim_[1..n]:[1:n] = computeInterConceptSimilarity(iConcepts, m_c)
oConcepts = iConcepts
cm = sort-decreasing(iConcepts, sim)
while(top(cm).sim > th_m)
    remove(oConcepts, top(cm)). c_new = merge(top(cm)). add(oConcepts, c_new)
    eliminate from cm all pairs including concepts in top(cm)
    computeSimilarity(c_new, oConcepts)
    insert new candidate pairs in order into cm
return oConcepts
```

Figure 3. Clustering elements in the corpus into concepts to estimate schema statistics.

The basic clustering procedure *clusterConcepts* is invoked $n + 1$ times, where n is the number of base learners. In each of the first n iterations, the ith base learner is heavily weighted compared to the others during the similarity computation. In the last iteration all the learners are equally weighted. This enables us to obtain clean initial clusters at the end of the first n iterations, and then collapse them more aggressively in the final iteration.

In each invocation of *clusterConcepts* the base learners are retrained, and the inter-concept similarity re-computed. This exploits the clustering from earlier iterations. The inter-concept similarity is computed by applying the learned models for the concepts on each other (outlined in Section 3.3). *computeSimilarities* re-computes the similarity of a new concept with all the existing concepts to be the maximum similarity among sub-concepts: $sim[c_{new}, c] = \max(sim[c_i, c], sim[c_j, c])$, if $c_{new} \leftarrow \text{merge}(c_i, c_j)$.

We ensure that two elements from a schema are never in the same concept by eliminating merge candidates that might have elements from the same schema. The intuition for this is that the same concept is never duplicated within one schema (similar to [12, 28]).

We note that clustering elements into concepts can also benefit the augment method. Instead of searching for similar elements to a given element, the algorithm can search for similar concepts. Thus the augmented model for any element e will include evidence from elements that are not directly similar to e, but are similar to other elements in the corpus similar to e.

Example schema statistics: We now give a flavor for some of the statistics that are computed.

Tables and Columns: For relational schemas, we separately identify table concepts and column concepts. We

compute for each table concept t_i and each column concept c_j, the conditional probability $P(t_i|c_j)$. This helps us identify the contexts in which columns occur. For example, the ProductID column most likely occurs in a Product table or an Availability table (as foreign key), but never in a Warehouse table.

Neighborhood: We compute for each concept the most likely other concepts they are related to. Briefly, we construct itemsets from the relationship neighborhoods of each element, and learn association rules from these. For example, we learn that AvailableQuantity $\rightarrow$ WarehouseID, i.e., availability is typically w.r.t. a particular warehouse.

Ordering: If the elements in a schema have a natural ordering (*e.g.* the input fields in a web form, or the sub-elements of an XML element), then we can determine the likelihood of one concept preceding another. For example, in our auto domain (Section 5), all web forms are such that the make is always before the model and price inputs, but after the new-or-used input.

4.2 Constraint-based Match Generation

Given two schemas S and T, our goal is to select for each element in S the best corresponding element in T (and vice-versa). Specifically, for each element e in S we will assign either an element f in schema T, or *no match* (ϕ). Let M represent such a mapping. Thus, $M = \cup_i \{e_i \leftarrow f_i\}$, where $e_i \in S$ and $f_i \in T \cup \{\phi\}$.

We assign a cost to any such mapping M that is dependent on our estimated similarity values and constraints:

$$Cost(M) = -\sum_i log\ sim[e_i, f_i] + \sum_j w_j \times K_j(M) \quad (1)$$

where $sim[e_i, f_i]$ is the estimated similarity of elements e_i and f_i, each K_j (≥ 0) is some penalty on the mapping M for violating the jth constraint, and w_j is a weight that indicates the contribution of the jth constraint. The first sum is an estimate of the total log likelihood of the mapping (if the similarities were interpreted as probabilities). The second sum is the penalty for violating various constraints. The task of generating the mapping from S to T is now reduced to the task of picking the mapping M with the minimal cost. We use A* search [22] to pick the best mapping M, which guarantees finding the match with the lowest cost. Our constraints are encoded as functions, K_j, that produce a value in the interval $[0, 1]$. We do not provide the details for each K_j, but note that the weight-learning algorithm described in the next section adapts w_j to the values K_j evaluates to.

A* explores the space of solutions by maintaining a frontier of partial solutions. Each partial solution is associated with an under-estimated cost of any eventual complete solution. At each step the partial solution with the least cost is examined, and replaced by new partial solutions that consider the different match options for an as yet unmatched element. The under-estimated cost of a partial solution is calculated by assuming that unmatched elements match their highest similarity candidate element without incurring additional constraint violations. The search stops when the first complete solution is reached. For very large schemas, we approximate by bounding the size of the frontier and shrinking it periodically to retain only a few of the best current partial solutions.

Some of the constraints we use are the following. For generic constraints, **uniqueness** states that each element must match with a distinct element in the target schema, and **mutual** (*pep* in [16]) states that e can match f only if e is one of the most similar elements of f and mutually f is one of the most similar elements of e. As domain constraints obtained from the corpus, we have the following: (1) **same-concept**: if two elements are to be matched, then they have to be similar to the same concept(s) in the corpus; (2) **likely-table**: if column e matches f, then e's table must be a likely table for f to be in; (3) **neighbors**: if element f is to be assigned to e, then elements that are likely related to f must be related to e; and (4) **ordering**: if element f is to be assigned to e, then the ordering corpus statistics should not be violated.

4.3 Learning Constraint Weights

Since many of the constraints we use in our matching algorithm are soft, i.e., encode preferences rather than strict conditions, the choice of weight for each constraint is crucial. Prior work has always hard-coded these constraint weights. We now describe how these weights can actually be learned from known mappings.

Consider a matching task between source schema S and target schema T. Consider a mapping M in which the correct matches are known for all elements in S except e. If f were the correct match for element e, then in order that e is also correctly matched with f, given the exact matches of other elements, the following condition must hold.

$$\forall f_i, f_i \neq f, Cost(M|e \leftarrow f) < Cost(M|e \leftarrow f_i) \quad (2)$$

This can be re-written for each f_i as below.

$$L(M, sim, e, f_i, \bar{w}) = \log\ sim[e, f] - \log\ sim[e, f_i] + \sum_j w_j \times [K_j(M|e \leftarrow f) - K_j(M|e \leftarrow f_i)] > 0 \quad (3)$$

An incorrect match is chosen for e if $L(M, sim, e, f_i, \bar{w}) \leq 0$. The best matches will be generated if the number of incorrect decisions are minimized. We try to achieve this by learning $\bar{w}$ from known mappings: specifically, given a set of known mappings $M_1, \ldots, M_n$, we try to find $\bar{w}$ such that $\| \{(e, f_i) : L(M_k, sim, e, f_i, \bar{w}) \leq 0\} \|$ is minimized. The similarity matrix sim has the results from matching the schemas in each M.

domain	type	# schemas	# elements (min-max,	average,	std.deviation)	# mappings	evidence
auto	webforms	30	3-28	11	6.7	74	text, variable names, select options
real estate	webforms	20	3-20	7	4.1	37	
invsmall	relational	26	11-33	18	4.9	39	names, examples, descriptions, context
inventory	relational	34	26-90	41	16.5	30	

Table 1. Characteristics of domains.

We find the w_js using *hill climbing* search [22]. Specifically, we start with a random initial guess for $\bar{w} = \langle w_j = 0 \rangle$. We repeatedly compute the change in number of violations of condition 3 by modifying each w_j separately, while keeping the others constant. The modification that results in the best change is accepted (with ties broken randomly) and this process is repeated. Since we can land in a local minimum we perform multiple restarts that can result in different solutions (because of random tie breaks), and choose the best overall best solution.

Increasing Precision: We note that the above training procedure tries to minimize the total number of incorrect decisions. Alternatively, we can modify the weight learning procedure to maximize *precision*, i.e., make only correct predictions at the expense of making fewer predictions. The intuition is that we do not penalize *no matches* (i.e., $e \leftarrow \phi$), and hence we omit these examples from the minimization.

5 Experimental results

We now present experimental results that demonstrate the performance of corpus-based matching. We show that corpus-based matching works well in a number of domains, and in general has better results than matching schemas directly. Furthermore, we show our techniques are especially effective on schema pairs that are *harder* to match directly. We also show that the results from corpus-based matching can be used to boost the performance of other matching techniques.

5.1 Datasets

Table 1 provides detailed information about each of our domains. We describe them briefly below. All the schemas we used in our experiments are available at [2].

Web forms: While a web form is not strictly a schema, there has been recent interest in automatic understanding and matching such structures [12, 28, 26]. Matching in this context is the problem of identifying corresponding input fields in the web forms. Each web form schema is a set of elements, one for each input. The properties of each input include: the human readable text, the hidden input name (that is passed to the server when the form is processed), and sample values in the options in a select box. We consider the auto domain and the real estate domain. In the auto domain the schemas are automatically extracted. As a result, there is some noise in identifying the human readable text with each input, but we do *not* make any manual corrections. The extractor does simple filtering of HTML tags and assigns each word of text to its nearest input. The unclean extraction makes matching tricky and challenging, but more realistic. The forms in the real estate domain *are* manually cleaned to identify human readable text correctly with inputs. The forms in the auto and real estate domain were obtained from the original lists in [17] and [9] respectively.

Relational Schemas: This data set was created by students in several database course offerings from similar, but not identical English domain descriptions of an online book and music seller with inventory in multiple warehouses. The elements in each schema include the tables and columns, a small number (2-5) of tuples per table, and foreign key constraints. We created two datasets from our collection. Each schema in invsmall is a subset of the original schema and includes only tables and columns about books and their availability. The schemas in inventory also have information about other products and warehouses. There are significant differences among the schemas, most importantly normalization: the number of tables varied from 3 to 12, and the standard deviation of number of elements is 16.5.

We also collected a dataset that consists of real-world schemas that we obtained from sources on the Internet (e.g., at xml.org), in the domain of purchase orders and customer information. The trends we observed in that data set were similar to other domains, and hence we do not report them in detail.

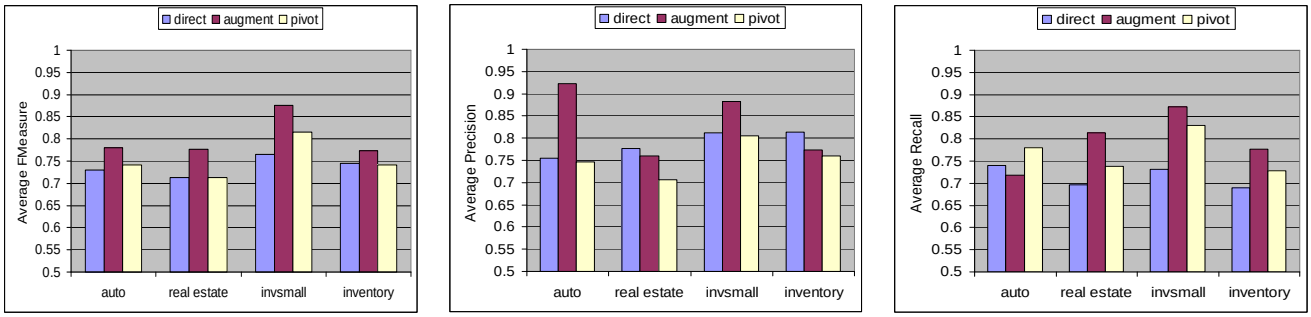
In each domain, we manually created mappings between randomly chosen schema pairs. The matches were *one-many*, i.e., an element can match any number of elements in the other schema. These manually-created mappings are used as training data and as a *gold standard* to compare the mapping performance of the different methods.

5.2 Experimental Methodology

We compared three methods: augment, direct, and pivot. augment is our corpus-based solution as described in the previous sections. direct uses the same base learners described in Section 3.1, but the training data for these learners is extracted *only* from the schemas being matched. direct is similar to the Glue system [6] and can be considered a fair representative of direct-matching methods. pivot, as described in Section 3, is the method that computes cosine distance of the interpretation vectors of two elements directly.

In each domain, we compared each manually created mapping pair against the mapping that was predicted by the different methods. The corpus contents (schemas and mappings) were chosen at random and are mentioned in each experiment.

Constraint Weight Training: In order to fairly compare the above three techniques, we selected a random 25% of the manually created mappings in each domain, and learned the constraint weights with them as training data (these



(a) Average F-Measure (b) Average Precision (c) Average Recall

Figure 4. (a) shows that augment **performs better overall than** direct **and** pivot **in all domains, and improves either precision (b) or recall (c) or both in each domain.**

mappings were *not* part of the corpus and hence not exploited by the base learners of augment). The constraint weights were learned separately for augment, direct, and pivot. The reported results are on the remaining 75% of the mappings.

5.3 Measuring Matching Performance

The result of each of our methods is a directional match: for each element in a schema, an element from the other schema is chosen such that the cost of entire mapping is minimized. If the gold standard has a match in which s matches a set of elements E, then a matcher is said to have predicted it correctly if s is predicted to match any one element in E, and every element in E is predicted to match s. As a result, any $1:m$ mapping is considered as $m+1$ separate matches.

We report matching performance in terms of three measures: *Recall*, *Precision*, and *F-Measure*. For a given schema pair, let c be the number of elements in the two schemas for which a match is predicted and the predicted match is correct. If a match was predicted for n elements, and a match exists in the gold standard for m elements, precision(p) is the fraction of elements with correct predictions, recall(r) is the fraction of matches in the manual mapping that were correctly predicted, and f-measure is the harmonic mean of precision and recall.

$$\text{FMeasure} = \frac{2pr}{p+r}, \text{where } p = \frac{c}{n}, r = \frac{c}{m}$$

Optimizing for f-measure tries to balance the inverse relation between precision and recall, and is used commonly in IR. An increase in f-measure indicates a better ability to predict correct matches and also identify non-matches. We report the f-measure averaged over multiple runs of *all* matching tasks in a domain.

5.4 Comparing Matching Techniques

Figure 4 compares the results of direct, augment, and pivot in each of the four domains. There are 22 (auto), 16(real estate), 19(invsmall), 16(inventory) schemas respectively and 6 mappings in the corpus. augment achieves a better f-measure than direct and pivot in all domains. There is a $0.03-0.11$ increase in f-measure as compared to direct, and a $0.04-0.06$ increase as compared to pivot. In general, augment has a significantly better recall as compared to direct in three out of four domains. This shows that augmenting the evidence about schemas leads to the discovery of new matches. The precision of augment is better than direct in two of the domains (auto and invsmall), lower in one (inventory) and about the same in the other.

The lower precision in the inventory domain is due to the presence of many ambiguous matching columns (e.g. multiple address, price and id columns) with similar names and very few data instances. However there is a noticeable increase in recall in this domain. The reason for the dramatic increase in precision in the auto domain is that there are a number of elements in the schemas that do not have any matches, and augment is able to prune them away with domain constraints.

Except for the recall on the auto domain, augment also performs better than pivot, which is just comparable to that of direct. This suggests that mere comparison of the interpretation vectors is not sufficient. For example, when two elements are very similar, but there is nothing in the corpus similar to these elements, pivot cannot predict a match while augment can.

The results in this section consider only the single best match for each element. In an interactive schema-matching system, we typically offer the user the top few matches when there is a doubt. When we consider the top-3 candidate matches, then augment is able to identify the correct matches for 97.2% of the elements, as opposed to 91.1% by direct and 96.7% by pivot.

5.5 Difficult versus Easy matching tasks

Our central claim is that corpus-based matching offers benefits when there is insufficient *direct* evidence in the schemas. To validate this claim, we divided the manual

mappings in each domain into two sets - easy and difficult: all the schema pairs in the test set were matched by direct and then sorted by direct's matching performance. The top 50% were identified as the easy pairs and the bottom 50% as the difficult pairs.

Figure 5a compares the average f-measure over the difficult matching tasks, showing that augment outperforms direct in these tasks. More importantly, the improvement in f-measure over direct is much more significant ($0.04-0.16$) than in Figure 4a, e.g., there is an improvement of 0.16 in the invsmall domain, and 0.12 in the real estate domain as compared to the 0.11 and 0.07 increases when all tasks are considered.

Figure 5b shows the same comparison over the easy tasks. The performance of augment for these tasks, while still very good, is in fact slightly worse than direct in one domain and the improvements are much less in the other domains. This is quite intuitive in retrospect. When two schemas are rather similar (easy to match directly), including additional evidence can lead the matcher astray.

In summary, for the schema pairs that are difficult to match we are able to leverage the corpus to significantly improve our matching performance, while on schema pairs that are relatively easier to match we get smaller improvements.

5.6 Constraints and Match Generation

We now look at the utility of constraints in the generation of matches. Given the similarities computed by augment, we compare two cases: using corpus and generic constraints ($C\&G$), against only using generic constraints(G). We also separately consider the cases of learning to improve overall match generation and that of learning for high precision results (Section 4.3).

Maximizing F-Measure: We compared $C\&G$ and G against a *best-match* strategy that does not use constraints: for $e \in S$, select element $f \in T$ with the highest similarity provided it is greater than a threshold th. Threshold th is determined from the same mappings that were used for training constraint weights. The f-measure for $C\&G$ (same as in Figure 4a) was better than the best-match in all the domains. The improvements were auto:0.02, real estate:0.12, invsmall:0.05, and inventory:0.02 respectively. There is no significant improvement for $C\&G$ over G (within 0.02 of each other in all the domains). The better performance of G and $C\&G$ indicate that the weights learned for the constraints are reasonable.

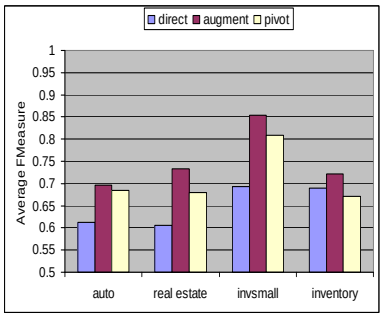
Maximizing Precision: In some contexts where schema matching systems are used, we may prefer to present the user only the matches of which the system is very confident, or equivalently, emphasize high precision. We claim that corpus constraints are useful in determining more matches under high precision requirements. Suppose, for example, that our goal is to achieve a precision of 0.9, i.e., 9 in 10 predictions are correct. Figure 5c shows the utility of constraints in achieving this high precision. The figure compares the augment method with two oracle strategies. *Best Naive* is the same as *best-match* except that the best threshold is chosen by trying all thresholds in increments of 0.1 and reporting the results when the precision is closest to 0.9. *Best Generic* is more sophisticated: in addition to *Best Naive* the **mutual** constraint described in Section 4 is strictly enforced. In a sense this captures the notion of best possible performance of G. Note that these are oracles because the best thresholds are chosen by comparing to the gold standard.

In three domains (real estate, invsmall, and inventory), the recall achieved by $C\&G$ is higher than both of the oracle solutions. G is able to achieve such a high precision only in one domain (invsmall). In this domain using the generic constraints appears to perform better. These results suggest that the corpus constraints are effective in pruning incorrect matches, and ensuring higher recall under very high precision requirements.

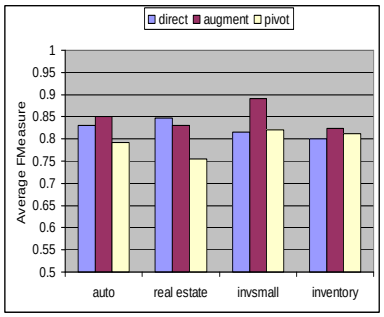
The poor performance of $C\&G$ in the auto domain is probably due to the overfitting of the constraint weights in this domain. Notice from Figure 4a that when optimized for f-measure (as opposed to precision), the precision obtained 0.92 is actually as required, and the precision is also high.

5.7 Interaction with other Matchers

We investigated using a corpus-based matcher as an independent module that contributes to other matching algorithms, and in particular, to the *Similarity Flooding* algorithm [16]. simflood is a graph matching algorithm that iteratively modifies the similarity of an element pair based on the similarities of other elements the pair are related to. We modified the code so that the similarities between elements in the schemas is initialized using the similarity matrix that results from direct (direct→simflood) and augment (augment→simflood). Figure 5d shows the results of this modification. direct→simflood does significantly better than simflood in all domains demonstrating that direct was a good matcher to be comparing against in our earlier experiments. augment→simflood significantly improves upon it in invsmall. There is a small improvement in real estate and inventory and a small drop in auto. The small changes in the webform domains are not surprising since these schemas do not have much structure for the graph matching to exploit. The rich structure (tables and columns) already present in inventory schemas results in augment not being able to contribute much in this domain. But in the evidence-parched schema pairs in invsmall there is the noticeable improvement. Note that in all four domains the f-measure of all variations of simflood is less than that of augment in Figure 4a. This is because simflood is a *conservative* matcher and makes only confident predictions (hence the high pre-



(a) difficult match tasks



(b) easy match tasks

	Corpus + Generic		Generic		Best Naïve*		Best Generic*	
	P	R	P	R	P	R	P	R
auto	0.88	0.57	0.74	0.80	0.90	0.78	0.90	0.72
real estate	0.89	0.51	0.76	0.75	0.81	0.49	0.84	0.49
invsmall	0.92	0.74	0.90	0.83	0.92	0.58	0.94	0.57
inventory	0.93	0.48	0.83	0.69	0.91	0.33	0.92	0.33

(c) corpus and generic constraints with augment

	simflood			direct → simflood			augment → simflood		
	P	R	F	P	R	F	P	R	F
auto	0.60	0.52	0.53	0.76	0.71	**0.71**	0.70	0.72	0.69
real estate	0.63	0.38	0.44	0.70	0.65	0.66	0.74	0.69	**0.69**
invsmall	0.84	0.46	0.59	0.85	0.54	0.65	0.88	0.67	**0.76**
inventory	0.87	0.47	0.61	0.90	0.59	0.71	0.90	0.61	**0.73**

(d) direct and augment with simflood

Figure 5. (a) and (b) show that augment **performs significantly better on difficult match tasks; (c) shows that corpus constraints are able to discover more matches at high precision; and (d) shows that a corpus can also improve the performance of other matchers.**

cision). Despite this conservative strategy, initializing with augment increases the recall while keeping precision intact, thereby demonstrating the utility of the corpus.

5.8 Additional experiments

We briefly mention two other experiments we performed to better understand the utility of the corpus.

Corpus Evolution: We studied the variation in augment as the number of schemas and mappings in the corpus are increased. In general there is a steady increase in f-measure as the number of schemas are increased and the number of mappings are increased. In the inventory domain just increasing the number of schemas with no mappings decreases the average f-measure, but increasing the number of mappings increases it. This is because these schemas are rather large and have a lot of ambiguity. Hence incorrect matches made during the clustering of elements into concepts lead to polluted clusters and hence the drop in performance. Adding mappings to the corpus results in cleaner clusters and the increase in performance.

Hand-tuned corpus: As an initial experiment into the various options of hand-tuning a corpus, we performed the following experiment. For each domain we designed a mediated schema and put it in the corpus, as well as mappings from each of the schemas in the corpus to the mediated schema. We experimented with variations of augment and pivot that used the mediated schema to define clusters rather than automatically learning them. We found that the f-measure is higher than those in Figure 4a only in the auto domain. This is primarily due to the fact that it was easy to define a reasonable mediated schema in that domain, whereas it was difficult to do so in the real estate and invsmall domains. Since the algorithm relied on the mediated schema, its performance was poor when the mediated schema was deficient.

6 Related Work

The use of previous schema and mapping knowledge has been proposed in the past, but in two very restricted settings. They either use previous mappings to map multiple data sources to a *single* mediated schema [5], or compose known mappings to a common schema [4]. Our goal is significantly more ambitious: we show that a corpus of schemas and mappings can be leveraged in many different ways to discover matches between two as *yet unseen* schemas. In mapping to a single mediated schema, the learning is more directed: we learn classifiers for the elements of that schema. In our context, we need to learn about the entire domain, rather than a single schema.

Another use for a corpus of schemas is described in [12], where the authors construct a mediated schema for a domain of web forms. They estimate the single most likely mediated schema that could generate all the web forms in a given collection. However, this approach to schema matching can only work when the domain is simple and clearly delimited. In [28] the authors collectively match a number of related web forms by clustering their fields. As seen in Section 3, we consider such clustering as one possible way of organizing the information in our corpus.

In [29], the authors propose matching two schemas by matching them both to a single domain ontology and then composing the two sets of matches. Both the domain ontology and the rules for matching the domain ontology elements to the different schemas are manually specified. Our corpus can be thought of as a domain ontology, however neither the concepts nor the rules are manually created. However, such a domain ontology when available can be used in pivot and augment and as a source for domain constraints.

In [8], it is shown how a corpus of web-services can be used to find clusters of parameter names that correspond to concepts. These concepts can be used instead of the parameters to improve search for similar web-service operations.

7 Conclusion

We described *corpus-based schema matching*, a set of techniques that leverages a collection of schemas and matches to improve schema matching. In a sense, corpus-based schema matching tries to mirror the main technique used in Information Retrieval, where similarity of queries to concepts are computed mostly based on analyzing large corpora of text. However, unlike documents, that can be abstracted as bags of words, schemas are complex artifacts and do not typically have large amounts of evidence. So leveraging a schema corpora requires different techniques.

Our main contribution is the Corpus-Based Augment method, which uses the corpus to increase the evidence taken into consideration in the matching process. In addition, we showed how we can leverage the corpus to discover concepts in the domain and domain constraints that further improve schema matching. Finally, we described an extensive set of experiments that validated the utility of corpus-based schema matching and studied some of the tradeoffs involved. The most important observation is that corpus-based schema matching is especially effective in hard-to-match schema pairs.

As future work, we would like to extend our techniques to larger schemas and complex mappings. Observe that matching large schemas does not require a corpora of large schemas: information extracted from smaller related schemas can be used to match similar fragments within larger schemas. Another direction is the incorporation of user-feedback in maintaining schema corpora, e.g., in the case of ambiguity in clustering elements into concepts user input can result in better formed clusters. Finally, we believe our techniques are a first step at capturing the intuition that schema corpora can be leveraged for several data management tasks [10].

Acknowledgment

We would like to thank Pedro Domingos, Rachel Pottinger, and Pradeep Shenoy for many helpful discussions, and the reviewers for their insightful comments. This work was supported by NSF ITR grant IIS-0205635 and NSF CAREER grant IIS-9985114.

References

[1] J. Berlin and A. Motro. Database Schema Matching Using Machine Learning with Feature Selection. In *CAiSE*, 2002.

[2] http://www.cs.washington.edu/homes/jayant/corpus.

[3] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *SIGMOD*, 2004.

[4] H.-H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, 2002.

[5] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *SIGMOD*, 2001.

[6] A. Doan, J. Madhavan, P. Domingos, and A. Y. Halevy. Learning to Map between Ontologies on the Semantic Web. In *WWW*, 2002.

[7] P. Domingos and M. Pazzani. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, 29, 1997.

[8] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *VLDB*, 2004.

[9] http://metaquerier.cs.uiuc.edu/repository/datasets/icq.

[10] A. Halevy and J. Madhavan. Corpus-based Knowledge Representation. In *IJCAI*, 2003.

[11] A. Y. Halevy. Structures, semantics and statistics. In *VLDB*, 2004.

[12] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD*, 2003.

[13] J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *SIGMOD*, 2003.

[14] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, 2002.

[15] J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based Schema Matching. In *Information Integration Workshop at IJCAI*, 2003.

[16] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm. In *ICDE*, 2002.

[17] http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8.

[18] R. J. Miller, L. M. Haas, and M. Hernandez. Schema Matching as Query Discovery. In *VLDB*, 2000.

[19] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI*, 2000.

[20] B. C. Ooi and K.-L. Tan. Special Edition on Peer-to-Peer Data Management. *TKDE*, 16(7), July 2004.

[21] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[22] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. 2nd edition, 2003.

[23] G. Salton, editor. *The SMART Retrieval System—Experiments in Automatic Document Retrieval*, 1971.

[24] A. P. Sheth and J. A. Larson. Federated database systems for managing, distributed, heterogenous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[25] K. M. Ting and I. H. Witten. Issues in Stacked Generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[26] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based Schema Matching for Web Databases by Domain-specific Query Probing. In *VLDB*, 2004.

[27] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, March 1992.

[28] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. In *SIGMOD*, 2004.

[29] L. Xu and D. Embley. Discovering Direct and Indirect Matches for Schema Elements. In *DASFAA*, 2003.