Analyzing and Revising Data Integration Schemas to Improve Their Matchability

Xiaoyong Chai, Mayssam Sayyadian, AnHai Doan
University of Wisconsin-Madison

Arnon Rosenthal, Len Seligman
The MITRE Corporation

ABSTRACT

Data integration systems often provide a uniform query interface, called a *mediated schema*, to a multitude of data sources. To answer user queries, such systems employ a set of *semantic matches* between the mediated schema and the data-source schemas. Finding such matches is well known to be difficult. Hence much work has focused on developing semi-automatic techniques to efficiently find the matches. In this paper we consider the complementary problem of *improving the mediated schema*, to make finding such matches easier. Specifically, a mediated schema S will typically be matched with many source schemas. Thus, *can the developer of S analyze and revise S in a way that preserves S's semantics, and yet makes it easier to match with in the future?*

In this paper we provide an affirmative answer to the above question, and outline a promising solution direction, called mSeer. Given a mediated schema S and a matching tool M, mSeer first computes a matchability score that quantifies how well S can be matched against using M. Next, mSeer uses this score to generate a matchability report that identifies the problems in matching S. Finally, mSeer addresses these problems by automatically suggesting changes to S (e.g., renaming an attribute, reformatting data values, etc.) that it believes will preserve the semantics of S and yet make it more amenable to matching. We present extensive experiments over several real-world domains that demonstrate the promise of the proposed approach.

1. INTRODUCTION

Data integration has been a long-standing challenge in the database and AI communities. The main integration approaches (whether they employ virtual integration, data warehouses, or information exchange via messaging) rely on development of a neutral schema and mappings between the neutral schema and the schemas of local data sources. In the remainder of this paper, we call this neutral schema a *mediated schema*.

VLDB '08, August 24-30, 2008, Auckland, New Zealand

To create the required mappings, a data integration system uses a set of *semantic matches* (e.g., location = area) between the mediated schema and the source schemas. Creating such matches is well-known to be laborious and error prone. Consequently, many semi-automatic schema matching solutions have been proposed. Much progress has been made (see [21, 13] for recent surveys), and today schema matching has become a vibrant research area. No satisfactory solution however has yet been found, and the high cost of finding the correct semantic matches continues to pose a bottleneck for the widespread deployment of data integration systems.

To address this problem, in this paper we propose to open another attack direction, by considering the complementary problem of *revising the mediated schema to improve its matchability*. Specifically, when creating the mediated schema S, can a developer P analyze and revise S in such a way that preserves S's semantics, and yet makes it easier to match with in the future? The ability to do this can prove quite helpful in many common integration scenarios, such as those detailed below.

EXAMPLE 1.1. *A developer P often must add new data sources to an existing data integration system I. To do so, P must match the schemas of the new sources with S, the mediated schema of I, using a matching tool M. (Typically P must also elaborate the found matches into mappings, which are for example full-fledged SQL expressions, using a tool such as* Clio *[26]; however, this mapping-creation step is outside the scope of this paper.)*

As another example, following recent trends of providing Web-based services, many integration systems (especially those in scientific domains) are being "opened up", so that members of the user community can easily add new data sources via a GUI (e.g., [20]). To add a source T, a user U must eventually invoke a matching tool M (provided at the system site) to match T's schema with the mediated schema S, then sift through the results to fix the incorrect matches.

As yet another example, developers often "compose" integration systems, i.e., take an integration system I, treat it as a single source, then integrate it with a set of other sources to build a higher-level integration system. In such cases, the mediated schema S of I will often be matched with other mediated schemas.

In all of the above cases, if the target mediated schema can be designed to be more amenable to matching, then it can be matched with new schemas more accurately and quickly. The problem of improving the matchability of mediated schemas is therefore appealing. But it is unclear exactly how this problem should be attacked.

A key contribution of this paper is that we provide such an attack plan. Specifically, we decomposed the above problem

into three well-defined subproblems. For each subproblem we then identified the main challenges and provided initial solutions. Finally, we demonstrated the promise of the approach, using extensive experiments on real-world data sets. Our work therefore can help to motivate further research in this novel approach to schema matching.

The setting for our current work is as follows. First, we focus on improving 1-1 matching (e.g., location = address) for relational mediated schemas, a common scenario in practice [21]. Besides its conceptual simplicity, 1-1 matching allows us to focus on analyzing the fundamental reasons for matching errors and thus provides a good starting point. We leave more complex matches and data representations as future work.

Second, we observe that in practice, when designing a mediated schema, developers often design *multiple* schemas: an *internal schema* S_i, serving to capture all relevant aspects of the integration domain, and one or several *external schemas* $S_e^1, \ldots, S_e^n$, serving as user query interfaces. Thus, our goal is to improve only the matchability of S_i (against which developers match source schemas or higher-level target schemas). This way, we can improve the accuracy of the matching process while respecting the very different design goals for the external schemas (e.g., being easy to understand and query).

Within the above setting, as the first subproblem, we consider how to define the notion of *matchability score*, which quantifies how well an internal mediated schema S_i matches future schemas using a given matching tool. Such a score has not been proposed before, and estimating it is a difficult challenge. To address this challenge, we propose to employ a synthetic workload W that approximates the set of future schemas and is generated automatically from S_i.

Using the above notion of matchability score, we then define and address the second subproblem: analyze different types of matching mistakes, and show how to produce a report that identifies potential matching mistakes of S_i. Given this report, a developer P can already revise S_i to address the mistakes.

Manually finding good revisions, however, is difficult and tedious. Hence, in the final subproblem, we consider how to automatically discover a good set of revisions, which can then be presented to P in form of a revised schema S_i^*. Developer P is free to accept, reject, or modify further these suggested revisions.

In summary, we make the following contributions:

- *Introduce the novel problem of analyzing and revising mediated schemas to improve their matchability.*
- *Describe a clear decomposition of the above problem into three well-defined subproblems:* estimating "matchability" of a mediated schema, producing a report that identifies potential matching mistakes of a mediated schema, and automatically discovering a good set of schema revisions (to improve matchability).
- *Identify the key challenges underlying these subproblems, and provide initial solutions.* These include a way to approximate the set of future schemas, an analysis of reasons for incorrect matching, a method to identify these reasons, and an algorithm to efficiently search for the most effective schema revisions.
- *Establish the promise of the approach via extensive experiments over four real-world domains with several matching systems.* The results show that we can reveal fundamental reasons for incorrect matches and can revise mediated schemas to substantially improve their matchability.

2. PROBLEM DEFINITION

We now describe the problem considered in this paper.

Multiple Mediated Schemas: We begin by considering the process of creating a mediated schema S. A developer P often wants S to satisfy multiple design objectives [22, 25]. Since S functions as a query interface, P often wants S to be *concise* (i.e., containing relatively few attributes), so that users can quickly comprehend and pose queries over S. At the same time, P also wants S to be *comprehensive*, i.e., to meet user requirements. Other design objectives for S include "attribute names and values should be easy for users to understand" and "the ordering of attributes should roughly reflect the orderings at the source schemas" (see [25] for more details).

Clearly, it is difficult to create a *single* schema S that satisfies all these *conflicting* objectives. So in practice P often creates *multiple* mediated schemas: an *internal* schema S_i and several *external* ones $S_e^1, \ldots, S_e^n$ [22]. P designs the internal schema S_i to be comprehensive, and uses it to match with the source schemas. P designs the external schemas $S_e^1, \ldots, S_e^n$ to be user query interfaces, and often defines them as views over the internal schema S_i.

Revising the Internal Mediated Schema: In this paper we will consider the above setting of multiple mediated schemas. In this setting, since P matches source schemas with only the internal mediated schema S_i, *we will consider the problem of revising S_i to improve its matchability.* Specifically, let M be the tool employed by developer P to match schemas (or by the system site to match the schemas of the sources supplied by users; see Example 1.1). Then we will revise S_i to improve its matchability with respect to M.

It is important to note that revising the internal mediated schema S_i this way would not affect other traditional design objectives. First, it would not affect the comprehensiveness of S_i, because we do not propose to drop or add any new attribute when revising S_i (see Section 5). And second, such revisions may necessitate revising the view definitions of external schemas $S_e^1, \ldots, S_e^n$ (over S_i). But it should not affect these schemas themselves, as well as the important design objectives placed on them (e.g., being concise, easy to understand, etc.).

A Problem Decomposition: Suppose developer P has created an initial version of the internal mediated schema S_i. Then to help P revise S_i, we envision providing three services: computing a matchability score, generating a report of potential matching mistakes, and suggesting schema revisions. For simplicity, we will call a system that provides these services mSeer (shorthand for "match seer").

As a start, P can simply ask mSeer to compute a score that quantifies how well S_i can be matched in the future, using M. This requires relatively little effort from P (just supplying S_i and M), and yet can already prove quite useful. For example, if the matchability score is low, then P may consider replacing matching tool M, or allotting more time for matching activities (in anticipation of having to correct more matching mistakes than initially expected).

Next, P can ask mSeer to generate a report that describes the potential matching mistakes and makes high-level suggestions for fixing them. P can then use the report to revise S_i. At the minimum, the report can alert P of "obvious problems" (e.g., two attributes with almost identical names and very similar data values) that are hard to spot in a large mediated schema, thus allowing P to quickly fix them. But it can do much more. Section 6 shows how such reports can also identify non-obvious, yet important potential problems for matching.

Finally, even if P recognizes potential matching problems, it is often still far from obvious how best to revise S_i, given the large number of potential revisions, and the complex interaction among them. To address this problem, P can ask mSeer to suggest a revision of S_i. mSeer then searches a space of schemas judged to be semantically equivalent to S_i, to produce a schema S_i^* that has higher matchability than S_i. P can then accept, reject, or revise S_i^*.

We now describe the three mSeer services in detail.

3. SCHEMA MATCHABILITY

In this section we introduce *schema matchability* and show how to estimate it. Henceforth, for simplicity, we will use the phrase "schema S" to refer to the internal mediated schema S_i, whenever there is no ambiguity.

3.1 Defining Schema Matchability

Recall from Section 2 that our goal is to improve the matchability of the internal mediated schema S with respect to a matching tool M. A reasonable way to interpret this notion of matchability is to say it measures *on average* how well S can be matched with future schemas, using M.

Specifically, let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be the set of all the future schemas that will be matched against S (of course, we often do not know $\mathcal{T}$), and $m(S, \mathcal{T}, M)$ be the matchability score of S w.r.t. $\mathcal{T}$ and M. Then we can write

$$m(S, \mathcal{T}, M) = \big[\sum_{T_i \in \mathcal{T}} accuracy(S, T_i, M) \big] / n \qquad (1)$$

where $accuracy(S, T_i, M)$ is the accuracy of matching S with T_i using M.

While in principle any measure of matching accuracy can be used, we will use F_1, a popular measure [11, 21], to define $accuracy(S, T_i, M)$. Specifically, suppose that applying M to schemas S and T_i produces a set of matches O. Then the accuracy of matching S and T_i using M is $accuracy(S, T_i, M) = 2PR/(P + R)$, where precision P is the fraction of matches in O that are correct, and recall R is the fraction of correct matches that are in O.

In addition to matchability, we also define the notion of *matching variance*, denoted as $v(S, \mathcal{T}, M)$, to capture the variance in the accuracy of matching S with future schemas:

$$v(S, \mathcal{T}, M) = \big[\sum_{T_i \in \mathcal{T}} (m(S, \mathcal{T}, M) - accuracy(S, T_i, M))^2 \big] / n.$$

Our goal will be to revise S to maximize its matchability (breaking ties among revisions by selecting the one that produces the lowest variance). However, computing schema matchability and variance as defined above requires knowing the future schemas T_i as well as the *correct* matches between these schemas and S (without which we cannot compute precision P and recall R). This is rarely possible.

EMPLOYEES

id	first-name	last-name	salary
1	Mike	Brown	42,000
2	Jean	Laup	64,000
3	Bill	Jones	73,000
4	Kevin	Bush	36,000

(a)

EMP

id	name	salary
1	Mike Brown	42K
2	Jean Laup	64K
3	Bill Jones	73K
4	Kevin Bush	36K

(b)

Figure 1: An example of schema perturbation

Hence, we show how to estimate schema matchability and variance using synthetic matching scenarios.

3.2 Estimating Schema Matchability

We estimate schema matchability by adapting a technique proposed in the recent eTuner work [23]. eTuner attacks a very different goal, namely how to tune a matching system to maximize accuracy. It however also faces the problem of finding $\mathcal{T} = \{T_1, \ldots, T_n\}$, the future schemas that will be matched with S. eTuner solves this problem by applying a set of common *transformation rules* to the schema and data of S, in essence randomly "perturbing" S to generate a collection of synthetic schemas $V = \{V_1, \ldots, V_m\}$.

EXAMPLE 3.1. *Suppose that S consists of the sole table* EMPLOYEES *in Figure 1.a. Then* eTuner *can apply the rule "abbreviating a name to the first three letters" to change the table name* EMPLOYEES *to* EMP*, then the rule "merging two neighboring attributes that share a suffix, and renaming it with their common suffix" to merge the* first-name *and* last-name *attributes, and the rule "replacing* ,000 *with* K*" to the data values of column* salary *of the table. The resulting table is shown in Figure 1.b.*

The paper [23] describes an extensive set of such rules, including those that perturb (a) the set of tables (e.g., joining two tables, splitting a table), (b) the structure of a table (e.g., merging two columns, removing a column, and swapping two columns), (c) the names (e.g., abbreviating names, adding prefixes), and (d) the data (e.g., changing formats, perturbing values). We note that these rules are created only once by eTuner, independently of any schema S.

Since eTuner generates schemas $V = \{V_1, \ldots, V_m\}$ from S, clearly it can trace the generation process to infer the correct matches $\Omega = \{\Omega_1, \ldots, \Omega_m\}$ between these schemas and S. Hence, the set V, together with the correct matches, form a *synthetic matching workload* $W = \{(V_i, \Omega_i)\}_{1..m}$ that is an approximation of the true future workload $\mathcal{T}$.

The synthetic workload idea can be adapted directly to our current context. Given a schema S, we first perturb S to generate a synthetic workload $W = \{(V_i, \Omega_i)\}_{1..m}$ (see [23] for the detailed algorithm). Next, we use M to match S with each schema V_i in W. Since we know Ω_i, the correct matches between S and V_i, we can compute $accuracy(S, V_i, M)$. We then return the average of $accuracy(S, V_i, M)$ over all schemas in W as our estimate of the true matchability score of S. We estimate the matching variance of S in a similar fashion.

Discussion: At a first glance, the idea of deriving a set of synthetic schemas from schema S might seem counterintuitive. One may question if it can effectively approximate the future schemas.

We believe that in the absence of *any* additional information, this provides a reasonable way to do it. (It is unclear what other alternatives we can consider.) While synthetic workloads differ from real future workloads, they do capture common variations in schema design. Moreover, although matchability scores estimated with synthetic workloads vs.

real future workloads will differ, we only need the matchability rankings to be similar (and they often do, see Section 6), in order to revise S effectively. Of course, if developer P has additional knowledge about the future workload, then P can create additional transformation rules to capture those.

One may also question if the so-derived schema pairs would be easy to match. The answer is no, as it turns out that "reverse engineering" the process is quite difficult, given that the rules are *randomly* applied. We note that synthetic scenarios like these have recently also been used in competitions on ontology matching (*oaei.ontologymatching.org*).

Perhaps a useful perspective we can take on the use of synthetic schemas is that they provide a reasonable set of "test cases" to estimate how good our solutions are. In other words, if our solutions cannot even handle synthetic schemas, then how much confidence we would have that they can handle the real ones?

Finally, we note that the above matchability estimation process requires data instances for schema S. To maximize accuracy, schema matching systems increasingly make use of such data instances [21]. Hence, we want to analyze *both the schema and data* of S and propose changes to both. To do so, mSeer requires developer P to supply several sample data instances for S (as a part of the input). Section 6.6 shows that mSeer works well with only a few (3-5) instances, thus not imposing an excessive burden on developer P.

4. ANALYZING SCHEMA MATCHABILITY

We now describe the report generator, the second mSeer service. Given an internal mediated schema S, the generator produces a report that lists the matchability and variance of S and the main reasons for matching mistakes.

EXAMPLE 4.1. *Figure 2 shows such a report. The report first describes schema S and the matching system M (e.g.,* Product1 *and* iCOMA *in this case, see Section 6). Next, the report shows that S has a matchability 0.76 and variance 0.09 (over a synthetic workload of 20 schemas).*

Next, the report tries to explain why S obtains a somewhat low matchability of 0.76. A reasonable way to explain this is to list the attributes of S, together with their matchability scores (so that developer P can get a sense about which attributes of S are difficult to match).

The *matchability score of an attribute* can be defined in a similar fashion to that of a schema (see Section 3.1). Then it can be estimated as follows. Let s be an attribute of S. Suppose that when matching S with schemas $V_1, \ldots, V_n$ of a synthetic workload W using a matching system M we obtain a set K of matches that involve s (i.e., matches of the form $s = t$, $t \in V_i$, $i \in [1, n]$). Then s's estimated matchability with respect to W and M is $m(s, W, M) = 2PR/(P + R)$, where P is the fraction of matches in K that are correct, and R is the fraction of correct matches involving s (and between S and the V_i's) found in K.

The report shows the most-difficult-to-match attributes first, to help the developer P quickly identify those. For example, the report in Figure 2 shows that attribute discount is the most difficult to match, with matchability 0.47.

Still, just showing that discount is difficult to match is not very informative for P. Hence, the report goes one step further, trying to explain the common matching mistakes involving discount and make suggestions on how to fix them. In Figure 2, the report lists two reasons R_1 and R_3 for discount. Reason R_1 for example states that iCOMA predicted

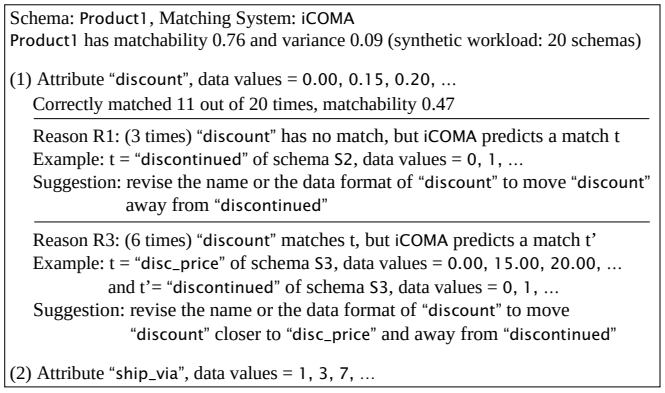

Figure 2: A sample matchability report

spurious matches for discount, such as discount = discontinued. To fix this mistake, the report suggests to pick a more distinctive name for discount. Section 6 provides examples of mistakes identified and suggestions made by the report on real-world schemas.

In the rest of this section we will first identify a set of common matching mistakes. Then we describe how to generate a report such as the above one.

4.1 Common Matching Mistakes

In what follows, we use the term *appearance* to refer to the name and the data format of an attribute. We divide matching systems into *local* and *global* ones, and start our analysis with the local ones.

4.1.1 Mistakes with Local Matching Systems

A *local* matching system M matches two attributes s and t by analyzing their appearances to compute a similarity score $sim(s, t)$, then declaring $s = t$, if $sim(s, t) \geq \epsilon$ for a pre-specified ϵ. M is local in that it decides if s matches t based solely on $sim(s, t)$, not on any other matches (as global systems that we describe later do). Examples of such systems include many of those from the COMA++ matching library [3], the LSD basic system (without the constraint handler) [12], and Semint [16].

Now consider applying M to schemas S and V, where V is a synthetic schema, and consider attribute $s \in S$. Matching mistakes involving s fall into three cases:

Case 1. Predict a Spurious Match: $s = none$, i.e., it has no match, but M predicts $s = t$, where $t \in V$. This implies that $sim(s, t) \geq \epsilon$. The fundamental reason is that

> R_1: the appearances of two non-matching attributes s and t are too similar.

To solve this problem, we should change the appearance of s to "move it away" from t. This can reduce $sim(s, t)$, thereby reducing the chance that M matches s with t. For example, if s has name "elec." (shorthand for "elective") with values "yes" and "no", and t has name "electricity" also with values "yes" and "no", then their appearances are too similar. To address this, we can expand s's name to "elective" and use values "1" and "0". As another example, if s has name "salary" with values "53000", "65500", etc., it can be easily confused with "zip code" (with values "53211", "60500", etc.), if in computing similarity scores M gives data value similarities a large weight. To address this, we can insert

into the data values of s characters that never occur in zip codes (e.g., change "53000" into "53,000") to "pry" these two attributes apart.

Case 2. Miss a Match: $s = t$, but M predicts $s = none$. This implies $sim(s,t) < \epsilon$. The fundamental reason is that

> R_2: the appearances of two matching attributes s and t are very different.

Examples include "yes/no" vs. "1/0", and "02.07.07" vs. "Feb 07, 2007". This is the reverse of Case 1. To solve this, we can change s's appearance to "bring it closer" to t. In many cases, however, this will not completely solve the problem. To see why, consider the following example.

EXAMPLE 4.2. *Suppose the synthetic workload W contains 100 attributes that match s: 60 attributes with data values "yes/no", and 40 with data values "1/0". Then no matter how we change s's data format, to "yes/no" or to "1/0", M will fail to match s in at least 40% of the cases.*

Fundamentally, the problem is that in the future schemas, the attributes that match s *can appear in many different formats*. Hence if s appears in just a single format, it may fail to match many such attributes. To address this problem, we propose a multi-appearance representation, which we will discuss shortly.

Case 3. Predict a Wrong Match: $s = t$, but M predicts $s = t'$. The mistake in this case is two-fold. First, M fails to predict the correct match $s = t$, which implies $sim(s,t) < \epsilon$. Second, M predicts instead a wrong match $s = t'$, which implies $sim(s,t') \geq \epsilon$. Thus the reason is that

> R_3: s is more similar to a non-matching attribute t', and less so to matching attribute t.

To avoid this, we should change the appearance of s such that it moves "closer" to t, to increase $sim(s,t)$, and "away" from t', to reduce $sim(s,t')$. This case thus in a sense combines Case 1 and Case 2.

Changing the appearance of s is relatively easy when t and t' are quite different. The more similar t and t' are, the more difficult this task becomes. In the extreme case, when t and t' are "almost identical" in their appearances, such changing may be impossible. For example, let s be "stime" (shorthand for "start time"). Suppose t and t' are "time1" and "time2", respectively, and suppose that all three attributes s, t and t' have very similar values (e.g., "3:05am", "4:00pm", etc.). Then t and t' are so similar that it is virtually impossible to change s so that it would have a higher chance of matching correctly. Fundamentally, this is because the future schema T is ill-designed, by having two almost identical attributes. In this case, there is not much we can do on schema S.

4.1.2 Mistakes with Global Matching Systems

A *global* system M matches two attributes s and t by examining not just their appearances, but also *external* information, such as domain constraints [12] and special filters [18]. M exploits such information to revise similarity scores and match selections.

With a global system M, matching mistakes involving s still fall into Cases 1-3 described earlier. However, the underlying reason for a mistake may be quite different. Consider for example Case 2: $s = t$, but M predicts $s = none$. If M is local, then by the definition of local systems, we know that $sim(s,t) < \epsilon$ and that this is the fundamental reason why M misses match $s = t$.

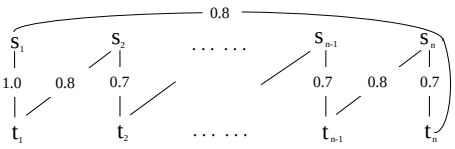

Figure 3: A matching scenario in a global system

However, if M is global, the reason for missing $s = t$ may be rather involved. It may even be the case that $sim(s,t) \geq \epsilon$ and yet M suppresses $s = t$, perhaps because t has been matched with another attribute s' and hence can no longer be matched with s, due to some constraint. In general, matches in a global system can influence one another in a rather complex fashion, as the following example illustrates:

EXAMPLE 4.3. *Figure 3 shows a matching scenario with attributes $s_1, \ldots, s_n$ and $t_1, \ldots, t_n$ of S and V, respectively. Here an edge $s_i - 0.7 - t_j$ denotes that $sim(s_i, t_j) = 0.7$; there is no edge between s_i and t_j if $sim(s_i, t_j) = 0$.*

Suppose that a global matching system M imposes the constraint that each attribute participates in at most one single match (e.g., [18]). Suppose further that M starts by selecting as a match the edge with the maximum score, and hence predicts $s_1 = t_1$. Since t_1 is already involved in this match, M has no choice for s_2 but to predict $s_2 = t_2$, and so on, until it predicts $s_n = t_n$. Now suppose that the correct matches are $s_n = t_{n-1}$, $s_{n-1} = t_{n-2}$, ..., $s_2 = t_1$, and $s_1 = t_n$. Then clearly the incorrect decision to match s_1 and t_1 has caused a chain of cascading matching errors, all the way to s_n and t_n.

Because of such cascading errors, pinpointing the exact reasons for matching mistakes of global systems can be very difficult. Consequently, we currently focus on identifying *some common reasons* for mistakes, rather than conducting a comprehensive mistake analysis for global systems.

Specifically, when Case 2 or Case 3 happens (i.e., $s = t$, but M predicts $s = none$ or $s = t'$), and $sim(s,t) \geq \epsilon$, clearly Reasons $R_1 - R_3$ do not apply. In this scenario, we have observed that a very common reason is that

> R_4: s is dominated by an attribute $s' \in S$.

By "dominating", we mean that $sim(s',t) \geq sim(s,t)$ (recall that t is the correct matching attribute for s). In this case, M often incorrectly matches s' with t. Then, due to a constraint such as "each attribute can participate in a single match", M can no longer match s with t. Consequently, it either declares $s = none$, leading to a mistake of Case 2, or $s = t'$, leading to a mistake of Case 3.

An extreme example of the domination scenario is when s and s' are "almost identical" (e.g., "time1" and "time2", with very similar data values "3:05am", "4:00pm", etc.). In this case, $s = t$ and $s' = t$ often have identical similarity scores, and M ends up guessing wrong 50% of the time.

To address the domination problem, we should change the appearances of s' and s so that s is "moved closer" to t and s' is "moved away" from t.

Summary: Table 1 briefly lists the conditions, likely reasons, and suggestions we have discussed so far, for both local and global systems. The first row of this table, for example, states that if $s = none$, but M predicts $s = t$, and

Conditions		Likely Reasons	Suggestions
s = none, predicts $s = t$	$sim(s, t) \geq \varepsilon$	R_1	(a) move s away from t
$s = t$, predicts s = none	$sim(s, t) < \varepsilon$	R_2	(a) bring s closer to t (consider multi-appearance representation (MAR) for s)
	$sim(s, t) \geq \varepsilon$, $\exists s'$ s.t. $sim(s', t) \geq sim(s, t)$	R_4	(a) move s closer to t (consider MAR) (b) move s′ away from t (c) check if ∃s′ such that s and s′ are highly similar
$s = t$, predicts $s = t'$	$sim(s, t) < \varepsilon \leq sim(s, t')$	R_3	(a) move s closer to t (consider MAR), and away from t′ (b) but, check if t and t′ are highly similar
	else if $\exists s'$ s.t. $sim(s', t) \geq \varepsilon > sim(s, t)$	R_4	(a) move s closer to t (consider MAR) (b) move s′ away from t (c) check if ∃s′ such that s and s′ are highly similar

Table 1: Conditions, reasons, and suggestions used in report generation

$sim(s,t) \geq \epsilon$, then R_1 is a likely reason, and developer P should consider changing the appearance of s to "move it away" from t. The report generator uses this table to identify likely matching mistakes (see Section 4.2).

4.1.3 Multi-Appearance Representation

We have seen from the discussion in Case 2 that in the future schemas the attributes that match $s \in S$ can appear in many different formats. Hence if s appears in just a single format (as is the case today), it may fail to match many such attributes. To address this problem, we experimented with a *multi-appearance representation (MAR)* for such an attribute s, by creating different *relational views* over s, and enforcing the constraint that any attribute that matches one of these views must also match s.

To illustrate, consider again Example 4.2. Suppose s is "waterfront" with values "1/0". Then we can create a view v_1 over s, with name $waterfront_1$ and data values "yes/no", then treat v_1 as another attribute of schema S. Next we enforce the constraint that any attribute t that matches v_1 must also match s, and vice versa. This ensures that no matter whether t takes values "yes/no" or "1/0", we can match t with s.

Creating such views in relational schemas should incur a moderate effort from developer P, and the views do not have to be kept up-to-date by the minute, for matching purposes. It is important to note that instead of creating views, P can also simply record in a text document that "s can also take "yes/no" values". However, no matching systems can exploit such *textual* information effectively today. Instead, virtually all of them have focused on exploiting the schema and data of attributes. Hence, we feel that capturing such information in views makes it more "understandable" to matching systems.

In theory, for an attribute s, we can create as many views as necessary, to capture all of s's possible future appearances. However, doing so can often make s "confusable" with other attributes, and hence can quickly decrease matching accuracy (e.g., by causing Case 1 or Case 3). Hence, developer P can propose such views for s, but P should let mSeer decide which views to keep. The experiment section shows that the use of such views as decided by mSeer can significantly improve matching accuracy.

4.2 Generating a Matchability Report

We are now in a position to describe the end-to-end working of report generation. Given a schema S, mSeer first generates a synthetic workload W. Next, mSeer applies the matching tool M to match S and schemas in W, then computes S's matchability and variance for the report.

Next, mSeer analyzes the above matching results to compute matchability scores for all attributes in S, and then displays these attributes in increasing order of their scores. For each attribute s, mSeer then generates an analysis as follows.

Let I be the set of all incorrect matches involving s (from workload W). mSeer finds the reason for each of these incorrect matches. Currently these reasons are $R_1 - R_4$ in Table 1 (or $OTHER$ if none applies). mSeer then groups matches in I based on their reasons, producing at most five groups. Next, mSeer reports each group as a triple (R, E, S): R is the reason, E is a concrete example to illustrate the reason, and S is a suggestion (to be described below). mSeer lists triples (R, E, S) in decreasing order of the corresponding group size (i.e., the number of matches in the group).

Within each group, mSeer selects as example E the incorrect match m that can be fixed most easily, since developer P seems likely to attempt to fix m first. Specifically, for group R_1, mSeer picks m with the lowest similarity score. For R_2, it picks m with the highest similarity score. For R_3, where $s = t$, but M predicts $s = t'$, it picks m that minimizes $[sim(s,t') - sim(s,t)]$. For R_4, where s is dominated by s', it picks m that minimizes $[sim(s',t) - sim(s,t)]$. mSeer then generates suggestion S by replacing variables in suggestion template with those in example E.

5. IMPROVING SCHEMA MATCHABILITY

Given a schema S, developer P can employ the report generator as described earlier to identify potential matching mistakes of S, then revise S to address these mistakes. Manually finding good revisions, however, is difficult and tedious. The revision advisor, the third mSeer service, addresses this problem. It automatically discovers a good set of revisions, then presents them to P, in form of a revised schema S^*. P is free to accept, reject, or modify further these suggested revisions.

We now describe the revision advisor. Clearly, the advisor can only suggest revisions that retain the *semantics* of S (e.g., it cannot suggest P to drop an attribute). Hence, we start by defining the notion of *semantically equivalent transformation rules* (or *SE rules* for short). Later we describe how the revision advisor finds a good set of SE rules to apply to S.

5.1 SE Transformation Rules

Let r be a transformation rule and $r(S)$ be the schema obtained by applying r to a schema S. Intuitively, we say that r is a *semantically equivalent (SE)* rule if for any schema

Categories	Sub-categories	Rules	Descriptions
Structure	*Schema-level*	merge-two-tables	Merges two tables based on their join path to create a new table.
		split-table	Splits a table into two, and duplicates key attributes in both tables.
	Table-level	merge-attributes	Merges multiple attributes into one (e.g., merging Day, Month, Year into Date).
Name	*Syntactic*	prefix-table-name	Adds the table name to the attribute name as prefix.
		drop-prefix	Drops the first token of the name (e.g., ContactName → Name).
		append-data-type	Appends the data type of the attribute to its name (e.g., appending Phone to attribute Office).
	Dictionary-based	expand-abbreviation	Expands common abbreviations (e.g., Qty → Quantity).
		expand-acronym	Expands acronyms (e.g., SSN → SocialSecurityNumber).
		use-synonym	Uses synonyms (e.g., PostalCode → Zip).
Data	*Numeric*	convert-unit	Converts the unit of the numbers (e.g., Price = "14,500" → Price = "14.5 K").
	Categorical	change-category-values	Converts categorical value representation (e.g., Fireplace = "yes/no" → Fireplace = "1/0").
	Special-type	change-data-format	Changes data formats of special data types (e.g., Date = "12/4" → Date = "Dec. 4").

Table 2: Classification of SE transformation rules

S, S and $r(S)$ are semantically equivalent, i.e., creator P can use $r(S)$ instead of S in his or her application.

SE rules fall into two categories: *domain-independent* and *domain-dependent*. Examples of domain-independent rules are "replacing data values "yes" with "1" and "no" with "0"", and "abbreviating a table name to its first three letters". Other examples include rules that cover special data types, such as "if s is a date attribute, then reformat s's values as "06/03/07"", and "if s is a price, then insert "$" to front of data values". To use such rules, we must recognize the type of an attribute (e.g., date, price, etc.). To do so, we employ a set of type recognizers, as described in [9]. Finally, an example of domain-dependent rules is "replacing attribute name "gName" with "gene-name"".

We have created a large set E of domain-independent rules, to be used in the current mSeer implementation and for our experiments. These rules are created only *once, when building* mSeer, not once per schema S. We omit a detailed description of E for space reasons, but show a high-level description in Table 2.

It is important to emphasize that this set of rules is not meant to be comprehensive. New rules can easily be added to the set, including domain-dependent ones supplied by P. However, the current set of rules is adequate as a starting point for us to examine the proposed mSeer approach, and to demonstrate mSeer's feasibility, a major goal of this paper.

5.2 Searching for Optimal SE Sequences

Let $E = \{r_1, \ldots, r_m\}$ be the set of SE rules fed into mSeer, as defined above. Abusing notation slightly, we will also use the term "rule r_i" to refer to a particular *application* of r_i to a schema S (i.e., r_i captures both the rule itself and an instance of applying it to an attribute of S), when there is no ambiguity.

Then given a schema S, we use $seq(S)$ to refer to the schema that results from sequentially applying rules $seq = (r_1, \ldots, r_n)$, where $r_i \in E$ for $i \in [1, n]$, to S. Note that SE rules are "transitive", in that $seq(S)$ is also semantically equivalent to S.

Intuitively, then, the goal of mSeer is to find a sequence seq^* that when applied to S yields a schema S^* with maximum matchability. Formally, $seq^* = \arg\max_{seq \in \mathcal{S}} m(seq(S))$, where $\mathcal{S}$ is the set of all sequences of SE rules in E and $m(seq(S))$ is the matchability of schema $seq(S)$. mSeer then faces two key challenges: how to estimate $m(seq(S))$ and how to find seq^* efficiently.

To address the first challenge, mSeer employs synthetic workloads, in the spirit of computing matchability that we have described so far. Recall from Section 3.2 that to estimate the matchability of S, mSeer employs a synthetic workload $W = \{(V_i, \Omega_i)\}_{1..m}$, where V_i is a synthetic schema obtained by perturbing S, and Ω_i is the set of correct matches between S and V_i. To estimate $m(seq(S))$, however, mSeer cannot simply employ W again, since the matching scenarios (S, V_i, Ω_i) do not involve S'. Instead, mSeer needs a new workload that approximates matching scenarios involving $seq(S)$. We show how to generate such a workload in Section 5.2.1.

To address the second challenge, mSeer employs look-ahead heuristics to cope with the exponential search space. The result is the algorithm RevSearcher, which approximates seq^*, and which we describe in detail in Section 5.2.2.

5.2.1 Estimating Matchability of Revised Schemas

After applying rules seq to schema S, we obtain a different but semantically equivalent schema $S' = seq(S)$. To determine whether applying seq is worthwhile, we need to estimate the matchability $m(S')$ of S'.

As discussed above, to compute $m(S')$, mSeer needs a workload W' that approximates matching scenarios involving S'. To achieve this, we augment mSeer as follows. When deriving the schemas in W, mSeer logs the applied SE rules R. These rules are then used to generate workload W' for S'. Specifically, mSeer generates W' by applying R to S', in the same way it generates W. After that, mSeer employs W' to compute $m(S')$. Applying the same rules R to generate W' ensures that W' is closest to W, compared with the workloads generated by randomly perturbing S'. This way, mSeer can compare the matchability scores of S' and S based on similar matching scenarios.

5.2.2 Algorithm RevSearcher

A simple algorithm H to approximate seq^* is to use the hill-climbing heuristic to find the best rule to apply at each step. First, H generates a synthetic workload W from S, and uses W to compute the matchability $m(S)$ of S. Then H generates all schemas $S_1, \ldots, S_m$ that can be obtained from S by applying a single SE rule in E.

Next, for each schema S_i, H computes its matchability $m(S_i)$ as described in Section 5.2.1. Let S_k be the schema with the highest matchability, i.e., $m(S_k) = \max_{i=1}^{m} m(S_i)$. If $[m(S_k) - m(S)] < \theta$ (currently set to 0.005), then H terminates, returning the schema S^* with the highest matchability it has found so far, together with the rule sequence that creates S^* from S. Otherwise, H sets S to S_k, sets S^* to S_k, and transforms the workload W to W_k. It then repeats the

search, starting with S_k.

In each search iteration, algorithm H finds and applies a *single* SE rule. Hence, it explores the search space rather "slowly", and at the same time is myopic. To address both problems, we develop algorithm RevSearcher. This algorithm works exactly like H, except that in each iteration it finds and applies *a set of SE rules*, instead of a single one (see the pseudocode in Figure 4). We now describe how RevSearcher finds this set of rules.

Compatible Rules: Let U be a set of SE rules. The result of applying U to S, denoted as $U(S)$, is meaningful only if the rules in U are *compatible*, in the sense that applying them *in any order* still results in the same schema $U(S)$. We say that two SE rules are *compatible* if they either apply to different attributes, or to different aspects of the same attribute (e.g., one applies to its name, and the other applies to its data values). Then we say that U is a *compatible set* if any two rules in U are compatible.

Finding the Best Set of Compatible Rules: In each search iteration, RevSearcher finds and applies a compatible set U^* of SE rules that maximizes matchability. Unlike H which enumerates all rules, RevSearcher cannot enumerate and evaluate all compatible sets, because there are often too many of them (if there are n SE rules, there may be up to 2^n such sets). Consequently, RevSearcher finds U^* greedily as follows.

Consider the first iteration, where RevSearcher starts with S. First, RevSearcher applies all SE rules to S and computes the matchability of all resulting schemas, just like H does, adding those rules that produce schemas with higher matchability than S to a set U. Next, RevSearcher computes the gain of each rule in U (defined below), adds the rule with maximum gain to U^* (which is initially empty), recomputes the gain of each remaining rule, then adds the rule that has maximum gain and that is *compatible* with all rules already in U^*, and so on. The iteration stops when U is empty or contains only rules that are either incompatible with some rules in U^* or of zero gain. This is the set of SE rules U^* that RevSearcher uses for the first iteration. Finding U^* for subsequent iterations is carried out in a similar fashion (see pseudocode in Figure 4).

Computing Gain of a Rule: All that remains is to describe computing the gain of a rule r, which measures the potential increase in matchability that applying r can bring. At first glance, it appears that this gain can be computed as $gain(r) = m(r(S)) - m(S)$, that is, the increase in matchability between S and the schema $r(S)$ obtained by applying r to S.

However, we found that applying this gain definition is not effective. For example, one might have two compatible rules, r_1 and r_2, that apply to the same attribute a of S (e.g., one to a's data values and one to a's name). Suppose they both increase the matchability of S. Then with the above gain definition, RevSearcher will add both of them to U^*. However, it may be the case that when applied together, they cancel the effects of each other. Consider a matching scenario where attribute $a = none$, but the matching system predicts $a = b$ (reason R_1 in Table 1). Both r_1 and r_2 reduce the errors in matching a by moving a away from b. In the meantime, however, they undesirably move a closer to some attribute c. Although applying either rule in isolation does not incur the incorrect match $a = c$, applying them both

Input: Schema S, set of SE rules $U = \{r_1, r_2, \cdots, r_n\}$
Output: maximal set of compatible rules U^*

1. Compute the matchability $m(S)$ of schema S;
2. For each r_i in U do
   2.1 Compute the matchability $m(r_i(S))$ of schema $r_i(S)$;
   2.2 If $m(r_i(S)) < m(S)$ then
       Remove r_i from U;
3. Compute the matchability $m(a_j, S)$ for each attribute a_j in S;
4. Let $m^*(a_j) = m(a_j, S)$, for each a_j in S;
5. $U^* = \phi$;
6. For each r_i in U do
   6.1 If r_i is compatible with all rules in U^* then
       Compute the matchability $m(a_j, r_i(S))$ for each a_j in $r_i(S)$;
       $gain(r_i) = \sum_j \max\{[m(a_j, r_i(S)) - m^*(a_j)], 0\}$;
7. $k = \arg\max_i(gain(r_i))$;
8. If $gain(r_k) > 0$ then
   8.1 Remove r_k from U, and add r_k to U^*;
   8.3 $m^*(a_j) = \max[m(a_j, r_k(S)), m^*(a_j)]$, for each a_j in S;
   8.4 Goto Step 6;
9 Return U^*;

Figure 4: The procedure that RevSearcher uses to find the best set of rules in each iteration)

might. This suggests that RevSearcher should select only one rule, which gives a higher matchability.

To alleviate this problem, we explore a different gain definition. Specifically, we define the gain of a rule r to be the total increase in matchability of the attributes $a_1, \ldots, a_n$ of S:

$$gain(r) = \sum_{i=1}^{n} \max\{[m(a_i, r(S)) - m^*(a_i)], 0\},$$

where $m(a_i, r(S))$ is the matchability of attribute a_i in schema $r(S)$ (if a_i does not exist in $r(S)$, then we set $m(a_i, r(S))$ to 0, indicating that r does not contribute to any gain on matchability of a_i). Furthermore, $m^*(a_i)$ is the maximal matchability that a_i has achieved so far. $m^*(a_i)$ is initially set to be $m(a_i, S)$, the matchability of attribute a_i in S. It is set to be $m(a_i, r(S))$ every time RevSearcher adds a rule r to U^* and $m(a_i, r(S))$ is higher than $m^*(a_i)$ at that point.

Note that $gain(r)$ is "conservative" in the sense that it "discourages" applying multiples rules to one attribute when subsequent changes to the attribute do not increase its matchability further. Also, it is "optimistic" in the sense that whenever $m(a_i, r(S))$ is lower than $m^*(a_i)$, this definition does not "punish" r; it simply sets the contribution of r to a_i to 0. Otherwise, it tends to underestimate the actual benefit of r, and RevSearcher ends up selecting fewer rules than it could.

6. EMPIRICAL EVALUATION

We now describe experiments with mSeer. First, we ranked a set of schemas according to matchability with (a) synthesized schemas, and (b) real schemas. The rankings strongly agree with one another. We thus conclude that for estimating matchability, synthesized schemas provide a promising proxy for using difficult-to-obtain real schemas.

Second, we provide anecdotal evidence that matchability reports produced by mSeer can help a schema designer identify likely matching problems.

Third, once we had revised a schema using the revisions suggested by mSeer, we matched it against a set of *real schemas*, and showed that we could improve matching accuracy by 1.3-15.2% for 17 out of 20 schemas across four

Domain	# schemas	# tables per schema	# attributes per schema
Course	5	3	13-16
Inventory	10	4	9-11
Real Estate	5	2	26-35
Product	5	2	46-50

Table 3: Real-world domains in our experiments

domains (while obtaining no improvement or minimally reducing the accuracy by 0.2-0.3% on the remaining 3). The results thus suggest that mSeer is robust and can revise schemas to improve their matchability across a range of domains.

Finally, we showed that (a) the multi-appearance representation could further improve matching accuracy by 7.1% on average, (b) mSeer is robust for small changes in the size of the synthetic workload, and (c) it requires only a few data instances to do well. We now describe the experiments in detail.

6.1 Experimental Setup

Domains: For research purposes, obtaining domains with a large number of realistic schemas is well-known to be difficult[1]. For our experiments, we obtained publicly available schemas [9, 12, 23] in four real-world domains, as shown in Table 3. "Course" contains university time schedules. "Inventory" describes business product inventories. "Real Estate" lists houses for sale, and "Product" stores product description of groceries.

Matching Systems: For experiments described in Sections 6.2-6.5, we employed a matching system called iCOMA, which consists of a name matcher, a decision-tree matcher, and a combiner. The name matcher compares names based on edit distance. The decision-tree matcher compares attributes based on their values, and the combiner combines the similarity scores of the matchers by taking their average. The name matcher and the combiner are taken from COMA++, a state-of-the-art matching library [3], and the decision-tree matcher is added to iCOMA from LSD [12], so that iCOMA can exploit data instances. For sensitivity analysis in Section 6.6, we also evaluated mSeer using a revised version of iCOMA, taken from COMA++.

Experimental Methodologies: We briefly describe the methodology employed for the main experiments (Section 6.4). In those experiments, for each domain in Table 3, we first randomly selected a schema to be the internal mediated schema S, then computed its average matching accuracy m with respect to the remaining schemas in the domain (treated as future schemas). Next, we applied mSeer to revise S into S^*. Then we computed the average matching accuracy m^* of S^*, again with respect to the remaining schemas in the domain. Finally, we report the difference $m^* - m$ as an estimate of the matchability improvement of S (using mSeer) in real-world scenarios.

6.2 Utility of the Matchability Concept

We first examine what matchability scores can tell us. Since we estimate such scores using synthetic workloads, it is unlikely that they will be roughly the same as the true scores (that can be computed if we know the true set of target schemas). However, we hoped that they would help us *rank* schemas, given that such ranking lies at the heart of schema revision.

Consequently, we want to know that if we rank a set of schemas using (a) synthetic workloads, and (b) real schemas, how strongly would such rankings agree. Toward this end, in each domain, say Course, we first selected a schema S, then perturbed it using SE rules one rule at a time, to obtain a set of schemas $\mathcal{S} = \{S_1, \ldots, S_{10}\}$.

Next, we ranked the schemas in $\mathcal{S}$ in decreasing order of their matchability scores, computed using a synthetic workload. Since matchability scores vary depending on the particularities of a workload, we ranked a schema $S_i \in \mathcal{S}$ higher than a schema $S_j \in \mathcal{S}$ only if their scores differ by at least ϵ (currently set to 0.005). We call the resulting ranked list SynList.

We then created TarList, a similar ranked list of the schemas in $\mathcal{S}$, except now we computed their matchability scores using *all schemas in Course other than S* as a real-world target workload.

Finally, we computed the distance between SynList and TarList as the ratio between the number of disagreeing pairs (with respect to their rankings) and the total number of pairs. This is the *Kendall distance*, a popular IR measure of the distance between two rankings [10], adapted to our context.

We repeated the above process for all other schemas S in Course, then computed the average Kendall distance. These distances, for Course, Inventory, Real Estate, and Product, are 0.28, 0.22, 0.19, and 0.27, respectively. For comparison purposes, the average Kendall distance between TarList (the ranking produced using real-world schemas) and a randomly generated list, again for the above four domains, are 0.43, 0.44, 0.39, and 0.45, respectively, roughly twice the distances produced using the synthetic schemas. This suggests that matchability scores computed by mSeer are indeed useful in helping rank schemas with respect to their matchability. The schema revision results in Section 6.4 further quantify this degree of "usefulness", in showing that by using such rankings (produced with synthetic workloads), mSeer was able to revise schemas to improve their matchability across all four domains.

6.3 Usefulness of Matchability Reports

We now provide anecdotal evidence that matchability reports produced by mSeer can help the schema creator identify likely matching problems. Table 4 shows snippets of matchability reports produced by mSeer (condensed and compiled in English, for exposition and space reasons). The reports cover two schemas: Product1 comes from Product domain, and TPCH is the publicly available schema of the well-known TPC-H benchmark (see www.tpc.org/tpch), which we also experimented with to broaden our range of experience with mSeer. (We did not include TPCH in our other experiments because we could not obtain a set of schemas comparable to TPCH.)

Part 1 of Table 4 reports that iCOMA failed to match discount and discounted, and incorrectly matched discontinued and discounted. It is clear from examining this part that attributes discount and discontinued of schema Product1 are "too similar" (Case 3, see Section 4.1.1). In particular, their names share the string "disco", which can confuse a name

[1]The largest domain that we are aware of is Thalia at www.cise.ufl.edu/research/dbintegrate/thalia. But its schemas are in XML, hence are not suitable for the current experiments.

(1) iCOMA failed to match "discount" of schema Product1 and "discounted" of schema Product1_S1 iCOMA incorrectly matched "discontinued" of schema Product1 and "discounted" of schema Product1_S1 Suggestion: revise the name or the data format of "discontinued" to move "discontinued" away from "discounted"
(2) iCOMA failed to match "P_MFGR" of schema TPCH and "P_MANUFACTURER_GROUP" of schema TPCH_S1 iCOMA predicted no match for "P_MFGR" of schema TPCH Suggestion: revise the name or the data format of "P_MFGR" to move "P_MFGR" closer to "P_MANUFACTURER_GROUP"
(3) iCOMA incorrectly matched "C_COMMENT" of schema TPCH and "P_COMMENT" of schema TPCH_S1 iCOMA incorrectly matched "C_COMMENT" of schema TPCH and "P_NOTES" of schema TPCH_S2 Suggestion: revise the name or the data format of "C_COMMENT" to move "C_COMMENT" away from "P_COMMENT" and "P_NOTES"

Table 4: Compilation of report snippets generated by mSeer

matcher (e.g., one using q-grams [3]). Given this, developer P can change the name, e.g., from "discontinued" to "terminated", then rerun mSeer, to see if the problem has been addressed.

Similarly, Part 2 of Table 4 reports that P_MFGR failed to match P_MANUFACTURER_GROUP. Here, the abbreviation "MFGR" may have caused the names not to match. Note that the knowledge "MFGR" is an abbreviation of "MANUFACTURER_GROUP" is highly domain specific. Since we simply cannot know if a particular matching system will possess such domain specific knowledge, it is better to revise the TPC-H schema to make it match aware by expanding such abbreviations.

Part 3 of Table 4 reveals a different problem. This part first reports that C_COMMENT matched P_COMMENT incorrectly. Given that both names share "COMMENT", this is not surprising. But then mSeer reports that C_COMMENT also incorrectly matched P_NOTES, despite the fact that their data values are quite different (one attribute records customer comments, while the other records product comments). A likely explanation for this is that the matching system knows "COMMENTS" is a synonym of "NOTES", and thus makes the latter incorrect match. To address this problem, it is important that the strings "C" and "P" in the names must be fully expanded (e.g., to "CUSTOMER" and "PRODUCT") to "push" the attributes away from each other as much as possible.

Other likely matching problems for the TPCH schema (that we found from the mSeer report) includes abbreviations such as "MK", the use of very short names for ID attributes (making all of them "confusable" with one another), and the merging of words without some separation characters, such as "RETAILPRICE" (instead of "RETAIL_PRICE") and "ORDERSTATUS".

From working with several mSeer reports, we found that a promising future work direction would be to produce aids in designing easily matched mediated schemas. Some can be "best practice" rules for humans, e.g., "avoiding short prefixes that carry crucial information (such as P_COMMENTS)", "avoiding very short names for ID attributes", etc. A richer direction would be to provide a library of idioms, to be used in constructing attribute names.

6.4 Automatic Schema Revision

Next, we examine how well mSeer can revise a schema to improve its matchability. Figure 5 shows the results for all four domains, five schemas in each domain (Inventory has 10 schemas, from which we randomly selected five). Consider the very first schema, Homeseekers of Real Estate (at the topmost left corner of the figure). Here, the two bars show the average matching accuracy of the original schema and that of the schema produced by RevSearcher, respectively. This average accuracy is computed by matching against the target workload of Homeseekers, i.e., the set of all remaining schemas in Real Estate. Note that this target workload consists of *real-world schemas*; it approximates the true set of target schemas that Homeseekers will be matched against in the future. We generated the bars for other schemas similarly.

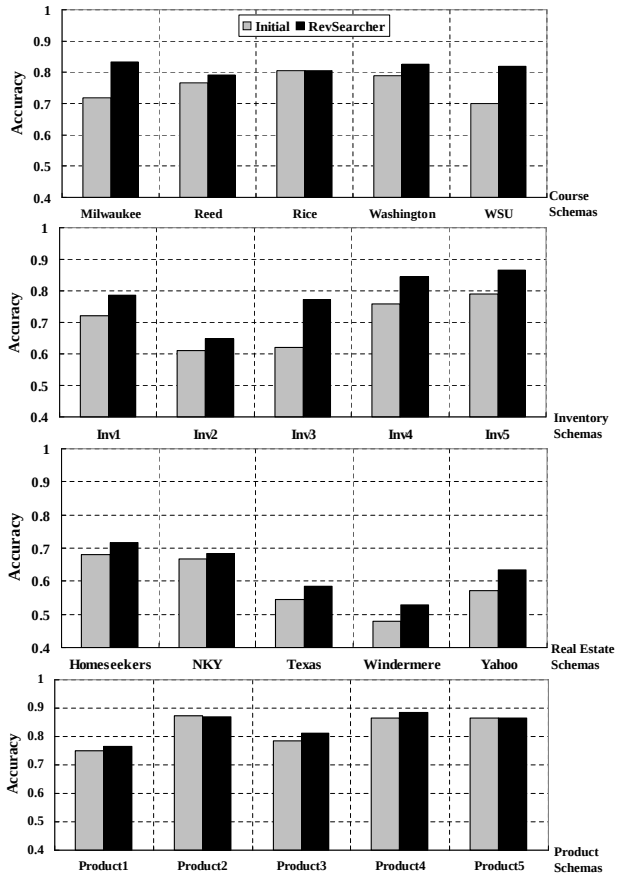

Figure 5: Matching accuracy of schemas produced by mSeer vs. that of the original schemas

The results show that RevSearcher was effective in improving matching accuracy. RevSearcher was able to revise schemas to achieve higher accuracy in 17 out of 20 cases, by 1.3-15.2%. It did not improve accuracy in one case (on Rice), and reduced accuracy minimally in two cases (on Product2 and Product5), by 0.2-0.3%. The results thus suggest that mSeer is robust and can revise schemas to improve their matchability across a range of domains. As an aside, the current unoptimized version of mSeer took 96-814 seconds to produce a revised schema in the experiments, spending most of time in searching for compatible rule sets (Section 5.2.2). To reduce runtime, we are exploring better search techniques and ways to reuse results across matching steps to compute matchability scores incrementally.

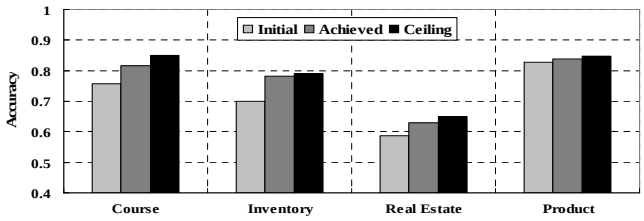

Figure 6: Improvement achieved vs. ceiling across all domains

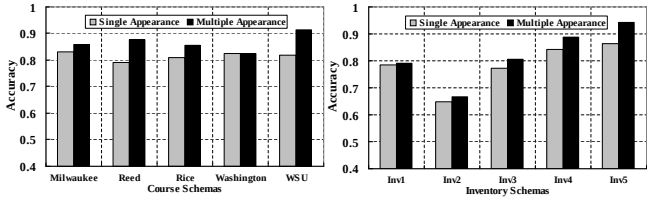

Figure 7: Accuracy with and without multi-appearance representation

Room for Improvement: How much better could mSeer revise a schema S if RevSearcher guided the search process using S's target workload, instead of a synthetic one? Figure 6 provides the answers for all four domains. In each domain, the three bars show the accuracies of the original schema, the schema produced by RevSearcher (using a synthetic workload), and the schema produced by a version of RevSearcher that uses the target workload, respectively. The accuracies are averaged over all sources in the domain.

The difference between the first and the third bar is the room for accuracy improvement. The results show that RevSearcher has done quite well. It achieves on average 69.7% of the improvement achievable with full knowledge (i.e., knowing the actual target workloads), demonstrating that its search strategy selects SE rules effectively. By expanding its set of SE rules, RevSearcher is likely to make inroads into the remaining 30%; and by pursuing an even better search strategy, RevSearcher can make further improvements, possibly beyond what is shown in the third bars.

6.5 Multi-Appearance Representation

Next, we examine the utility of multi-appearance representation (MAR) in schema revision (see Section 4.1.3). Figure 7 shows the results for the real-world schemas in Course and Inventory (experiments on other domains show similar results). For each schema, the two bars show the accuracy of mSeer in single-appearance and multi-appearance settings, respectively, measured using the real schemas as the target workloads.

The results show that using MAR significantly improves the matchability of schemas, increasing accuracy in 9 out of 10 cases, on average by 5% in Course, and 3% in Inventory. MAR failed to improve accuracy in only one case (on Washington). This suggests that MAR is quite promising as a way to revise a schema with modest effort and yet making it more match aware.

6.6 Sensitivity Analysis

Size of Synthetic Workload: Figure 8.a shows the accuracy of the revised schema that mSeer produces, as we vary the number of schemas in the synthetic workload W. In the figure the lines show average accuracies and the vertical bars show the maximum-minimum accuracy ranges. The results

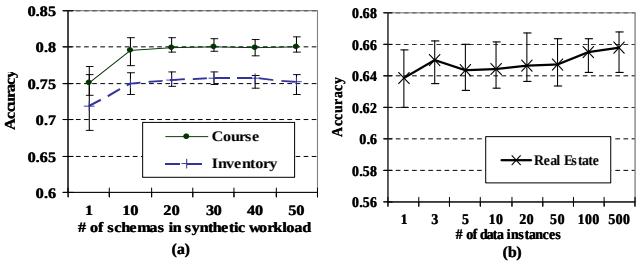

Figure 8: Change in matching accuracy with respect to (a) size of synthetic workload, and (b) number of data instances

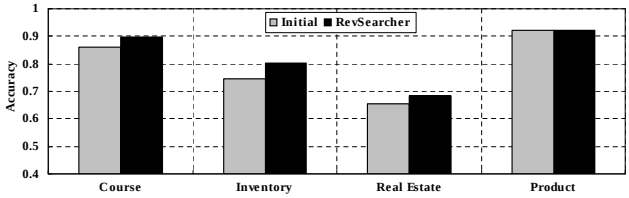

Figure 9: Matching accuracy with a new matching system

show that as W's size increases from 1 to 20, W captures the results of more transformation rules, thus better representing true target workload. Consequently, matching accuracy increases and the maximum-minimum fluctuations decrease. After size 30-35, however, all transformation rules have been captured in W, and as the size increases further, W's "distance" to the real workload increases, and its performance starts to decrease. This result is consistent with the observations in [23], for tuning matching systems. Overall, the results suggest an optimal workload size in the range of 20-30. The results also show no abrupt degradation of accuracy, thus demonstrating that mSeer is robust for small changes in the workload size.

Number of Data Instances: Figure 8.b plots the accuracy averaged over all sources in Real Estate, as we vary the number of data instances available to mSeer (i.e., to the decision-tree matcher). We chose Real Estate because it has the most of data instances available. The results show that more data instances led to a slow steady climb in accuracy. However, the accuracy is already quite high (within 2% of the maximum accuracy achieved) for 3-5 data instances. This suggests that mSeer requires only a few data instances to do well, and thus does not impose an unduly heavy burden on the schema creator.

New Matching System for mSeer: Next, we examine the performance of mSeer with respect to a different matching system. Instead of using system iCOMA described earlier (in Section 6), we employed a new system where the name matcher compares names using TF/IDF instead of edit distance, and the combiner takes the maximum of the similarity scores instead of the average (see COMA++ [3]).

Figure 9 summarizes the results with this new matching system, over all four domains. The results show that RevSearcher was able to revise schemas to improve accuracy in all four domains. RevSearcher for instance increased the average accuracy in Inventory by 5.7%. The results thus suggest that mSeer can be effective with more than one matching system.

7. RELATED WORK

Schema matching has received increasing attention over the past two decades (see [21, 13] for recent surveys). Many matching techniques have been developed, employing for example, machine learning [16, 12, 9], IR [7], and information theory [15]. Recent work has also explored incremental schema matching [5], self-organizing mapping [8], mapping debugging [6], mapping compilation [17], discovering mapping expression [2], information capacity in schema integration [19], data matching in ontology integration [24], hierarchy integration [27], and Web information integration [1, 4]. Once matches have been found and verified, they are typically elaborated into mappings [21] using a tool such as Clio [26].

A complementary problem (first raised in [14]) is then to revise schemas to make finding semantic matches easier. As far as we know, our work offers the first attack plan for this problem, placed in the context of revising mediated schemas of data integration systems. The work closest to ours is eTuner [23]. That work however attacks a very different goal, namely, given a schema S, how to tune a matching system M (i.e., selecting the right matching components to be executed and correctly adjusting their knobs) to maximize matching accuracy over future schemas. In a sense, our problem can be considered complementary: given a matching system M, how to "tune" (i.e., revise) a schema S, to maximize matching accuracy of S with future schemas.

8. CONCLUSION AND FUTURE WORK

We have described the novel problem of analyzing and revising mediated schemas to improve their matchability, and presented an initial promising solution. Our work can help to motivate further research in this novel direction to schema matching. A sample of important issues are the following. We have proposed a reasonable way to generate synthetic schemas. But what is the optimal way? What should be an optimal set of rules? Our analysis of matching mistakes and related experiments have achieved the goal of demonstrating that such a report of matching mistakes can be very useful to the schema creator. But can we improve the analysis further? In particular can we analyze better matching mistakes of global matching systems? Likewise, can we develop a better set of SE rules and a better search technique? Many more interesting challenges remain, such as developing an interactive environment (in which a creator can accept or revise a suggested schema revision on the fly, and can in general interact with the system in real time to revise the schema) and generalizing the work here to other data representations (e.g., XML) or problem contexts (e.g., revising schemas to facilitate record matching).

From a broader perspective, perhaps the most important conclusion drawn from this work, as well as the eTuner one, is that synthetic schemas can be very helpful for schema matching, and thus deserves more studies into their generation and usage.

Acknowledgment: We thank the reviewers for the insightful comments. This work is supported by NSF Career IIS-0347903, an Alfred Sloan fellowship, and an IBM Faculty award.

9. REFERENCES

[1] K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. The chatty web: Emergent semantics through gossiping. In *WWW-03*.

[2] Y. An, A. Borgida, R. J. Miller, and J. Mylopoulos. A semantic approach to discovering schema mapping expressions. In *ICDE-07*.

[3] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD-05*.

[4] L. Barbosa, J. Freire, and A. Silva. Organizing hidden-web databases by clustering visible web documents. In *ICDE-07*.

[5] P. A. Bernstein, S. Melnik, and J. E. Churchill. Incremental schema matching. In *VLDB-06*.

[6] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB-06*.

[7] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *DS-7*, 1997.

[8] P. Cudre-Mauroux, S. Agarwal, A. Budura, P. Haghani, and K. Aberer. Self-organizing schema mappings in the gridvine peer data management system. In *VLDB-07*.

[9] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering complex matches between database schemas. In *SIGMOD-04*.

[10] P. Diaconis. Group representation in probability and statistics. *IMS Lecture Series. Institute of Mathematical Statistics*, 11, 1988.

[11] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, 2002.

[12] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine learning approach. In *SIGMOD-01*.

[13] A. Doan, N. F. Noy, and A. Y. Halevy. Introduction to the special issue on semantic integration. *SIGMOD Record*, 33(4), 2004.

[14] A. Halevy and C. Li. Information integration research: The NSF IDM workshop breakout session. In *IDM-03*.

[15] J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *SIGMOD-03*.

[16] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *DKE*, 33, 2000.

[17] S. Melnik, A. Adya, and P. A. Bernstein. Compiling mappings to bridge applications and databases. In *SIGMOD-07*.

[18] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: a versatile graph matching algorithm. In *ICDE-02*.

[19] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *VLDB-93*.

[20] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Interactive query formulation over web service-accessed sources. In *SIGMOD-06*.

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

[22] A. Rosenthal and L. Seligman. Scalability issues in data integration. In *AFCEA Federal Database Conf*, 2001.

[23] M. Sayyadian, Y. Lee, A. Doan, and A. Rosenthal. Tuning schema matching software using synthetic scenarios. In *VLDB-05*.

[24] O. Udrea, L. Getoor, and R. J. Miller. Leveraging data and structure in ontology integration. In *SIGMOD-07*.

[25] W. Wu, A. Doan, and C. Yu. Merging interface schemas on the deep web via clustering aggregation. In *ICDM-05*.

[26] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data driven understanding and refinement of schema mappings. In *SIGMOD-01*.

[27] K. Zhao, R. Ikeda, and H. Garcia-Molina. Merging hierarchies using object placement. In *ICDE-08*.